Study questions on analysis of algorithms

Note: The intention in providing these questions is to show the student what sorts of problems we address in this course. With the multiple-choice questions, most of the problems are factual. Knowing how to answer the questions can raise students’ confidence and not knowing can help guide the student’s study.

Contents

Intro and background
1. Classes of problems
2. Verification of algorithms
3. Recurrences; brute force
4. Divide and conquer
5. Greedy and other efficient optimization algorithms
6. Intractability and approximation
7. Parallel, distributed, interactive computing
Summary
Study questions on Introduction

1. **What this course offers**
   1. This course claims to offer (a) facts; (b) a place to think about thinking; (c) a place to think about technology; (d) a place to think about thinking about technology; (e) answers

3. **How this course will deliver**
   1. The highest-priority objective of this course, of those below, is (a) memorizing facts; (b) agreeing with the instructor; (c) agreeing with the textbook; (d) working in a team; (e) writing programs
   2. The highest-priority objective of this course, of those below, is (a) memorizing facts; (b) agreeing with the instructor; (c) agreeing with the textbook; (d) presenting to a group; (e) writing programs
   3. The highest-priority objective of this course, of those below, is (a) memorizing facts; (b) agreeing with the instructor; (c) agreeing with the textbook; (d) documenting research; (e) writing programs
   4. Emphasis in this course is said to be on (a) lectures; (b) slides; (c) critical discussion and interaction; (d) right answers; (e) memorizing
   5. Central classroom guidelines are (a) taking notes and getting facts; (b) attending class and being on time; (c) focus and mutual respect; (d) listening and remembering; (e) none of these
   6. Attainment of learning objectives counts for about ____% of final grade in this course (a) 0; (b) 10; (c) 60; (d) 90; (e) 100

7. In academic work, direct use of text found on the Internet (a) is free without attribution; (b) requires a reference; (c) requires quoting and a reference; (d) costs money; (e) none of these
8. Signing work means that signer (a) requests course credit for the work; (b) has gathered the words from various sources; (c) claims to have written the words; (d) agrees with the words; (e) knows those who wrote the words
9. Plagiarism is (a) claiming another person’s work as one’s own; (b) copying exam answers; (c) copying exam questions; (d) unauthorized collaboration; (e) none of these
10. Cases of alleged academic dishonesty at FSU (a) stay between student and instructor; (b) are discussed among instructors; (c) may go to the office of the Dean of Students; (d) are referred to law enforcement; (e) none of these
11. In this course, exercises and project outside class (a) are the main form of assessment; (b) don’t matter; (c) are essential for learning; (d) are to be shown only to the instructor; (e) are always group work
12. Waiting until after mid-semester to do the assigned work is said to be (a) mandatory; (b) a suggested option; (c) very risky; (d) impossible; (e) a cause for automatic failure
13. All students are expected to attain all ____ objectives (a) recommended; (b) optional; (c) challenge; (d) priority; (e) essential
14. The growth mindset (a) is a technique of artificial intelligence; (b) tells us that only talent matters; (c) tells us that only effort matters; (d) tells us that effort and talent matter; (e) is a business strategy
Study questions on background (Data Structures, Finite Math)

0.0a Recall logic/proof concepts**M

1. **Logic and proofs**

1. \( \land \) denotes (a) set membership; (b) union; (c) \( \wedge \); (d) a relation between sets; (e) negation
2. \( \neg \) denotes (a) set membership; (b) union; (c) \( \neg \); (d) a relation between sets; (e) logical negation
3. A string is a (a) collection; (b) set; (c) tree; (d) sequence; (e) list
4. \( \lor \) denotes (a) set membership; (b) union; (c) \( \lor \); (d) OR; (e) implication
5. \( \implies \) denotes (a) set membership; (b) union; (c) \( \implies \); (d) OR; (e) implication
6. Propositional logic is associated with (a) boolean variables and logical operators; (b) quantifiers; (c) predicates; (d) numbers and relational operators; (e) none of these
7. An if-then assertion whose first clause is true is (a) never true; (b) sometimes true; (c) always true; (d) meaningless; (e) none of these
8. A rigorous demonstration of the validity of an assertion is called a(n) (a) proof; (b) argument; (c) deduction; (d) contradiction; (e) induction
9. A proof that begins by asserting a claim and proceeds to show that the claim cannot be true is by (a) induction; (b) construction; (c) contradiction; (d) prevarication; (e) none of these
10. A proof that proceeds by showing the existence of something desired is by (a) induction; (b) construction; (c) contradiction; (d) prevarication; (e) none of these
11. Proofs by contradiction (a) dismiss certain rules of logic; (b) misrepresent facts; (c) start by assuming the opposite of what is to be proven; (d) end by rejecting what is to be proven; (e) none of these
12. Induction is a(n) (a) algorithm; (b) program; (c) proof; (d) proof method; (e) definition
13. Contradiction is a(n) (a) algorithm; (b) program; (c) proof; (d) proof method; (e) definition
14. Construction is a(n) (a) algorithm; (b) program; (c) proof; (d) proof method; (e) definition
15. A proof that begins by asserting a claim and proceeds to show that the claim cannot be true is (a) direct; (b) by construction; (c) by contradiction; (d) by prevarication; (e) by contradiction
16. The induction principle makes assertions about (a) infinite sets; (b) large finite sets; (c) small finite sets; (d) logical formulas; (e) programs
17. A proof that proceeds by showing that a tree with \( n \) vertices has a certain property, and then shows that adding a vertex to any tree with that property yields a tree with the same property, is (a) direct; (b) by contradiction; (c) by induction; (d) diagonal; (e) none of these
18. A proof that shows that a certain property holds for all natural numbers is by (a) induction; (b) construction; (c) contradiction; (d) prevarication; (e) none of these
19. The principle of mathematical induction states that if zero is in a set \( A \) and if membership of any value \( x \) in \( A \) implies that \( (x + 1) \) is in \( A \), then (a) \( A \) is all natural numbers; (b) the proof is invalid; (c) \( A \) is the null set; (d) \( A \) is \( \{ x \} \)
20. In an inductive proof, showing that \( P(0) \) is true is (a) the base step; (b) the inductive step; (c) unnecessary; (d) sufficient to prove \( P(x + 1) \); (e) sufficient to prove \( P(x) \) for all \( x \)
21. In an inductive proof, showing that \( P(x) \) implies \( P(x + 1) \) is (a) the base step; (b) the inductive step; (c) unnecessary; (d) sufficient to prove \( P(x) \) for some \( x \); (e) sufficient to prove \( P(x) \) for all \( x \)
22. In an inductive proof, showing that \( P(0) \) is true, and that \( P(x) \) implies \( P(x + 1) \), is (a) the base step; (b) the inductive step; (c) unnecessary; (d) sufficient to prove \( P(x) \) for some \( x \); (e) sufficient to prove \( P(x) \) for all \( x \)
23. An inductive proof with graphs might proceed by (a) showing a contradiction; (b) showing a counter-example; (c) considering all graphs one by one; (d) starting with some simple graph and adding one vertex or edge; (e) none of these
24. The base step in an inductive proof might (a) show that \( P(0) \) is true, and that \( P(x) \) implies \( P(x + 1) \); (b) show that \( P(0) \) is true; (c) show that \( P(x) \) implies \( P(x + 1) \); (d) give a counterexample; (e) assume the opposite of what is to be proven
25. The inductive step in an inductive proof might (a) show that \( P(0) \) is true, and that \( P(x) \) implies \( P(x + 1) \); (b) show that \( P(0) \) is true; (c) show that \( P(x) \) implies \( P(x + 1) \); (d) give a counterexample; (e) assume the opposite of what is to be proven
26. An inductive proof might consist of (a) showing that \( P(0) \) is true, and that \( P(x) \) implies \( P(x + 1) \); (b) showing that \( P(0) \) is true; (c) showing that \( P(x) \) implies \( P(x + 1) \); (d) giving a counterexample; (e) assuming the opposite of what is to be proven, and proving a contradiction

0.0b Recall array concepts**M

2. Algorithms on arrays and lists

1. Arrays are structures that are (a) linked; (b) branching; (c) linear; (d) dynamically allocated; (e) none of these
2. For array \( A \), \( |A| \) is (a) the absolute value of the sum of \( A \)'s elements; (b) the absolute value of \( A \); (c) the smallest element of \( A \); (d) the number of elements in \( A \); (e) none of these
3. One member of a linked-list node must be a(n) (a) character; (b) integer; (c) node; (d) reference to a node; (e) none of these
4. To access a certain node in a singly-linked list (a) takes one step; (b) takes two steps; (c) requires that all its predecessors be visited; (d) requires that all its successors be visited
5. To delete a node a singly linked list, using the best-known algorithm, the address of which node must be known? (a) none; (b) the first; (c) the predecessor of the node to delete; (d) the node to delete; (e) the successor of the node to delete
6. Search of a linked list of \( n \) nodes may require visiting up to how many nodes? (a) \( 1 \); (b) \( 2 \); (c) \( n \); (d) \( n^2 \); (e) none of these
7. An advantage of linked lists is (a) low access speed; (b) optimal search speed; (c) zero overhead; (d) expandability; (e) none of these
3. To efficiently locate the number 9 in a sorted array of random integers, we would use a (a) linear search; (b) binary search; (c) Bubble sort; (d) Quick sort; (e) none of these

4. The analysis of algorithms is most concerned with (a) testing solutions; (b) the documentation of programs; (c) object orientation; (d) the time complexity of programs; (e) all of these

5. For a problem of size $n$, a solution of the worst time complexity would be (a) 1 (constant time); (b) $n$ (linear); (c) $n$ squared (quadratic); (d) $2^n$ to the $n$ power (exponential); (e) none is worst

6. A language is a (a) string; (b) number; (c) set of numbers; (d) sequence of strings; (e) set of strings

7. When $A$ and $B$ are sets, $(A \times B)$ is (a) a set of ordered pairs; (b) an arithmetic expression; (c) a sequence of values; (d) all of these; (e) none of these

8. $\subseteq$ denotes (a) a set membership; (b) union; (c) conjunction; (d) a relation between sets; (e) negation

9. $\cup$ denotes (a) a set membership; (b) union; (c) AND; (d) a set; (e) negation

10. $\emptyset$ denotes (a) a set membership; (b) union; (c) AND; (d) a set; (e) negation

11. $\in$ denotes (a) a set membership; (b) union; (c) AND; (d) a relation between sets; (e) negation

12. \{1,2,3\} $\cup$ \{2,4,5\} = (a) \{1\}; (b) \{1,2\}; (c) \{2\}; (d) \{1,2,3,4,5\}

13. A relation on set $A$ is (a) an element of $A$; (b) a subset of $A$; (c) an element of $A \times A$; (d) a subset of $A \times A$; (e) $A$

14. A function $f: \{1,2,3\} \rightarrow \{0,1\}$ is a set of (a) integers; (b) ordered pairs; (c) sets; (d) relations; (e) none of these

4. Trees and hashing

1. The height of a complete binary tree is (a) the number of nodes it contains; (b) the maximum path length between two leaf nodes; (c) the number of leaf nodes; (d) the maximum path length from the root to a leaf node; (e) infinite

2. Each step of the binary-search algorithm (a) reduces the size of the sub-array to be searched by about half; (b) finds the search key; (c) reports failure; (d) moves one array element; (e) compares two array elements

3. The time complexity of the binary search is in the order of (a) $O(1)$; (b) $O(n)$; (c) $O(n \log n)$; (d) $O(n)$ squared; (e) $O(n \log n)$

4. In Quicksort, the pivot is (a) a subscript; (b) an element value; (c) a turning point in the execution of the algorithm; (d) used to leverage performance; (e) none of these

5. A binary search tree node has up to two (a) root nodes; (b) children; (c) paths to a given node; (d) parents

6. The root of a tree node's left subtree is (a) the root of the tree; (b) the node's left child; (c) the node's right child; (d) the node's left or right child; (e) the node itself

7. The root node of a binary search tree contains (a) the lowest value in the tree; (b) a value not higher than that stored in its left child; (c) a value not higher than that stored in its right child; (d) the highest value stored in the tree; (e) no value

8. A new node is inserted into a binary search tree (a) at its root; (b) as a leaf; (c) as a parent of some other node; (d) in time proportional to the size of the tree; (e) none of these
9. Which is not an operation that can be performed on binary search trees? (a) insert item; (b) delete item; (c) display all items; (d) search for an item; (e) determine maximum possible size.

10. A hash function (a) is recursive; (b) is void; (c) returns a reference; (d) typically maps from a key value to an array subscript; (e) is a randomizer.

11. Address calculation in source code is associated with (a) hashing; (b) dynamic allocation; (c) all array accesses; (d) recursion; (e) searching.

12. To implement a priority queue, it is most time-efficient to use (a) a simple vector; (b) linked list; (c) heap; (d) binary search tree; (e) hash table.

13. A heap is (a) any array; (b) any tree; (c) a complete binary tree; (d) a binary tree in which no node has exactly one child; (e) none of these.

14. The depth of a heap of size n is close to (a) 1; (b) log₂n; (c) the square root of n; (d) n/2; (e) n².

**Recall graph concepts**

5. Graphs

1. A graph is (a) a set of integers; (b) a set of vertices; (c) a set of vertices and a set of edges; (d) a set of edges; (e) a set of paths.

2. Graphs are implemented as (a) trees; (b) matrices or arrays of lists; (c) queues; (d) stacks; (e) none of these.

3. A tree is a kind of (a) list; (b) array; (c) graph; (d) all of these; (e) none of these.

4. Which might be used in task scheduling? (a) functions; (b) vertices; (c) undirected graphs; (d) directed graphs; (e) none of these.

5. A tree is a graph that is (a) connected and cyclic; (b) connected and acyclic; (c) unconnected and cyclic; (d) unconnected and acyclic; (e) none of these.

6. A graph is defined in part by (a) exactly one ordered pair of vertices; (b) a relation; (c) a cycle; (d) one path joining each pair of vertices; (e) none of these.

7. A graph may be fully represented by (a) its vertices; (b) its edges; (c) an adjacency matrix; (d) the degrees of its vertices; (e) none of these.

8. A graph may be conveniently represented as (a) a vector of characters; (b) a two-dimensional array of Booleans; (c) a single linked list of Booleans.

9. The degree of a vertex in a graph is (a) the number of vertices in its graph; (b) the number of edges in its graph; (c) all array accesses; (d) the number of distinct connected subgraphs; (e) the number of other vertices adjacent to it.

10. A graph is defined in part by (a) exactly one ordered pair of vertices; (b) a relation; (c) a cycle; (d) one path joining each pair of vertices; (e) none of these.

11. A series of edges that connect two vertices is called (a) a path; (b) a cycle; (c) a connection; (d) a tree; (e) a collection.

12. To design a communications network that joins all nodes without excessive lines, we must find a (a) path; (b) connectivity number; (c) minimal spanning tree; (d) expression tree; (e) search tree.

13. A series of edges that form a path from a vertex to itself is (a) a spanning path; (b) a cycle; (c) a connection; (d) a tree; (e) an edge.

14. A weighted graph has an adjacency matrix that is (a) integers; (b) vertices; (c) real numbers and ∞; (d) boolean; (e) none of these.

15. The prerequisite relationships among required courses in the Computer Science major form a (a) binary tree; (b) linked list; (c) directed acyclic graph; (d) weighted graph; (e) spanning tree.
Questions to assess background (data structures, finite math)

0.1a Apply logical inference**
Use Modus Ponens, Modus Tollens, or the definition of implication to prove the assertion at right; name the rule used.
1. \((q \rightarrow r) \land q \rightarrow r\)
2. \(\neg r \lor q\)
3. \(p \land (\neg q \lor r) \rightarrow r\)
4. \(\neg p \land (q \rightarrow p) \rightarrow q\)
5. \(q \land (\neg q \lor r) \rightarrow p\)
6. \(\neg p \lor q \land p \rightarrow q\)

Below, you may abbreviate words with their initials; e.g., “c” = “clouds”.
7. Dark clouds mean it will rain; and I see dark clouds. Prove that it is raining.
8. There’s no class on holidays. There’s class today. Prove that it’s not a holiday.

0.1b Use a quantifier**
Use quantifiers and predicates to express the following in predicate logic.
1. Some athletes are fast.
2. All athletes are strong.
3. Some fast people are athletes.
4. All strong people are athletes.
5. Some athletes are not tall.
6. All tall athletes are strong.
7. All fast people are athletes.
8. Some strong people aren’t athletes.

0.1c Distinguish predicate from propositional logic**
1. What two features distinguish predicate logic from propositional logic?
2. Name and describe the sorts of assertions that predicate logic can express that propositional logic cannot.
3. Distinguish the meanings of “\((\forall x) P(x)\)” and “\((\exists x) P(x)\)” and name the logic that supports them.
4. Describe some limitations of propositional logic and state how another logic overcomes them.
5. Describe the quantifiers and how they address a limitation of propositional logic.
6. What are quantifiers, and which logic supports them?
7. Describe an alternative to propositional logic. Distinguish the two logics.

0.1d Write a direct proof**
Use direct proof to show that
1. the product of any natural number and any even natural number is even.
2. the difference between any two even natural numbers is even.
3. for any \(m > 3\), \(m^2 - 4\) is non-prime.
4. the sum of any even natural number and any odd one is always odd.
5. for any two consecutive integers \(a\) and \(b\), the difference between \(a^2\) and \(b^2\) is an odd number.
6. for any whole number \(x\), \((x^2 + 2x + 1)\) is not prime.
7. For any \(n\), \(2^{n+1} + 2^n + 1\) is not prime.

0.1e Describe mathematical induction**
1. Describe the two parts of an inductive proof.
2. What is the principle of mathematical induction?
3. What specific classes of theorems can the principle of mathematical induction be used to prove?
4. In an inductive proof, what must be shown, other than \(P(0)\)?
5. Explain the role of \(P(n) \Rightarrow P(n+1)\) in some mathematical proofs.
6. Describe the proof principle used in a proof that shows a property of some natural number and goes on to show that the same property holds for all natural numbers.
7. How is the following used? \(P(0) \land (P(n) \Rightarrow P(n+1)) \Rightarrow (\forall n \mathbb{N}) P(n)\)

0.1f Use induction to prove a theorem about numbers**
Prove by mathematical induction that for all natural numbers \(n\), greater than zero,
1. \(n^2 + n = (2 + 4 + 6 + \ldots + 2n)\)
2. \(\sum_{k=1}^{n} k = (n^2 + n)/2\)
3. \((n^2 + 2n)\) is divisible by 3
4. \(1 + 6 + 11 + \ldots + (5n - 4) = (5n^2 - 3n)/2\)
5. \(1 + 3 + 5 + \ldots + (2n - 1) = n^2\)
6. \(2^0 + 2^1 + 2^2 + \ldots + 2^n = 2^{n+1} - 1\)
7. For \(n > 4\), \(3 + n^2 < 2^n\)
8. \(\sum_{k=1}^{n} 2k = 2^{n+1} - 1\)
9. The sum of all whole numbers from 1 to \(n\) is \((n^2 + n)/2\)

0.2a Write and explain an algorithm to traverse an array**
Write pseudocode or program code to efficiently compute or determine, with respect to an array of integers:
1. the average
2. subscripts of the leftmost element that is same as its successor
3. number of elements that are the same as the first
4. smallest element
5. range (maximum minus minimum)
6. number of zeroes
7. subscripts of largest element
8. whether all values are the same
9. whether values are in ascending order
10. length of the longest sequence of adjacent elements that are in ascending order.

0.2b Explain the linear-search algorithm**
1. Describe the linear search algorithm.
2. Describe how you would tell whether an arbitrary array \(A\) of natural numbers contains the value 6, without calling a method.
3. Explain an algorithm that tells whether a file contains the word “wow”.
4. What is a procedure for searching a random array, and how many steps does it take?
5. Describe an algorithm to tell whether a string contains the letter ‘q’.

Describe an alternative to propositional logic. Distinguish the meanings of “\((\forall x) P(x)\)” and “\((\exists x) P(x)\)” and name the logic that supports them.

Describe the limitations of propositional logic. Name the logic that supports them.

Describe some limitations of propositional logic and state how another logic overcomes them.

Describe the quantifiers and how they address a limitation of propositional logic.

What are quantifiers, and which logic supports them? Describe an alternative to propositional logic. Distinguish the two logics.

Prove by mathematical induction that for all natural numbers \(n\), greater than zero,
1. \(n^2 + n = (2 + 4 + 6 + \ldots + 2n)\)
2. \(\sum_{k=1}^{n} k = (n^2 + n)/2\)
3. \((n^2 + 2n)\) is divisible by 3
4. \(1 + 6 + 11 + \ldots + (5n - 4) = (5n^2 - 3n)/2\)
5. \(1 + 3 + 5 + \ldots + (2n - 1) = n^2\)
6. \(2^0 + 2^1 + 2^2 + \ldots + 2^n = 2^{n+1} - 1\)
7. For \(n > 4\), \(3 + n^2 < 2^n\)
8. \(\sum_{k=1}^{n} 2k = 2^{n+1} - 1\)
9. The sum of all whole numbers from 1 to \(n\) is \((n^2 + n)/2\)

Write pseudocode or program code to efficiently compute or determine, with respect to an array of integers:
1. the average
2. subscripts of the leftmost element that is same as its successor
3. number of elements that are the same as the first
4. smallest element
5. range (maximum minus minimum)
6. number of zeroes
7. subscripts of largest element
8. whether all values are the same
9. whether values are in ascending order
10. length of the longest sequence of adjacent elements that are in ascending order.

Describe the linear search algorithm. Describe how you would tell whether an arbitrary array \(A\) of natural numbers contains the value 6, without calling a method. Explain an algorithm that tells whether a file contains the word “wow”. What is a procedure for searching a random array, and how many steps does it take? Describe an algorithm to tell whether a string contains the letter ‘q’.
6. Describe a way to search a linked list for a node containing a certain value, \( x \).

**0.2c Explain a sorting algorithm**

(1-3) Describe how the following sorts work.

1. Bubble
2. Selection
3. Insertion
4. Describe the loops required to sort an array, in an algorithm of your choice.
5. Describe what the following pseudocode does.
   Repeat
   \( \text{swapped} \leftarrow \text{false} \)
   for \( i \leftarrow 1 \) to \( \text{size}(A) - 1 \)
   if \( A[i] > A[i + 1] \)
   \( \text{swap}(A[i], A[i + 1]) \)
   \( \text{swapped} \leftarrow \text{true} \)
   until \( \text{swapped} \) is \text{false}
6. Describe the steps of an algorithm that arranges an array in descending order.

**0.2d Write an algorithm with nested loops**

Write pseudocode for an algorithm that
1. given input of \( n \), displays the first \( n \) prime numbers
2. puts an array in descending order
3. returns \text{true} iff any two elements in an array have the same value
4. finds the length of the longest series of values of elements in an array that are all the same
5. determines the winner in a vote for candidates coded by integers in an array; that is, finds the integer that is most common
6. finds all the divisors of integer \( n \).
7. finds the starting index of the longest ascending run of adjacent elements in an array

**0.2e Explain linked-list, stack, and queue operations**

Describe the operations on:
1. a linked list
2. a stack
3. a queue

Describe steps in the following operations:
4. prepending to a linked list
5. appending to a linked list
6. inserting an item in a linked list
7. adding an item to a stack
8. removing an item from a stack
9. adding an item to a queue
10. removing an item from a queue

**0.3a Explain the running time of an array traversal, versus array size**

Describe the running time the algorithm you wrote for 2.0d above, in relation to the size of the array. Show your reasoning.

**0.3b Explain or apply a concept in set theory**

1. What does the universal set refer to?
2. What is the complement of a set?

For sets \( A \) and \( B \) explain the value and meaning of
3. \( (B \subseteq A) \)
4. \( (B \supseteq A) \)
5. \( A \cap B \)
6. \( A \cup B \)
7. \( A \setminus B \)
8. \( A \cap \emptyset \)
9. \( A \cup \emptyset \)

**0.3c Describe a function**

1. Distinguish relations from functions.
2. What are the polynomial functions? Give an example.
3. What are the exponential functions? Give an example.
4. Distinguish partial from total functions.
5. Distinguish sets from functions.
6. Distinguish the domain of a function from its range.
7. What is the relationship of \( f : A \rightarrow B \to A \times B \)?
8. What are the domain and the range of the square-root function?

**0.4a Describe a basic concept of trees**

Draw a tree with
1. six vertices, at least two of which are neither a root nor a leaf.
2. six vertices, one of which is of degree 3
3. five vertices, at least one of which is neither a root nor a leaf
4. six vertices, four of which are leaf nodes
5. seven vertices that is a complete binary tree
6. five vertices, two of which are leaves
7. five vertices, at least two of which are internal

**0.4b Explain why a BST or heap enables efficiency**

In terms of the number of vertices or nodes, explain the running time of an efficient algorithm that
1. performs BST search (average case)
2. traverses a path in a heap from the root to a leaf.
3. performs BST insertion (average case)
4. deletes all items from a balanced BST of height \( n \)
5. descends from the root to a leaf of a balanced binary search tree with \( 2^n \) nodes
6. performs addition on pairs of binary numerals that range in value from 0 to \( n \)
7. swaps a value up from a leaf to the root in a heap
8. extracts a value from a heap

**0.4c Explain the binary search**

1. Describe the steps in the binary search.
2. Write pseudocode for the binary search.
3. Which values in an array are inspected in the binary search?
4. Why is it a precondition of the binary search that the array be sorted?
5. Describe how a binary search of an array sorted in descending order would be carried out.
6. How are the binary search and the search of a binary search tree alike?
0.5a Describe a basic concept of graph theory**

1. What is a path? Name a special classes of paths.
2. What is a cycle?
3. What is the degree of a vertex; of a graph?
4. When is \( G' = (V', E') \) a subgraph of \( G = (V, E) \)?
5. Describe adjacency the adjacency matrix of a weighted graph.
6. What is a connected graph?

0.5b Perform an operation on a graph**

1. (a) Write an adjacency matrix or linked-list representation of the graph below; (b) Give the sum of the degrees of the vertices

2. For the graph above, describe change that would cause it to be (a) acyclic; (b) unconnected.
3. Referring to the graph below, and showing your work, give the outdegree of vertex \( a \), and the least-weight cycle that includes three vertices.

4. With respect to the graph below, explain why it is or is not (a) cyclic; (b) connected; and (c) list, in order, the vertices in the longest path

5. Of the graphs below, (a) Which are directed? (b) Which are cyclic? (c) What is the degree of \( a \) in (i)?

6. (a) Give the connectivity components of the graph below; (b) Explain why it is cyclic or acyclic; (c) Give its adjacency matrix.

7. Describe an implementation of the graph data structure and a way to add an edge.
1.0a Recall basic problem concepts*<sup>M</sup>

1. Vectors, matrices, and sets
1. An example of an order statistic is (a) average; (b) mode; (c) standard deviation; (d) median; (e) none of these
2. The order statistic problem finds the _____ of an array
   (a) average element value; (b) median value; (c) kth largest element; (d) kth element; (e) sorted version
3. The minimum priority queue problem is (a) an optimization problem; (b) a data-storage problem; (c) a sorting problem; (d) a search problem; (e) none of these
4. String matching is a(n) _____ problem
   (a) vector; (b) optimization; (c) state-space search; (d) behavior-of-program; (e) interactive computation

2. Constraint and optimization problems
1. Optimization problems are ___ constraint satisfaction problems (a) equivalent to; (b) harder than; (c) easier than; (d) a subset of; (e) a superset of
2. A problem of finding values of several variables such that a certain condition holds is called (a) graph search; (b) tree traversal; (c) constraint satisfaction; (d) sorting; (e) optimization
3. Constraint satisfaction is a problem of (a) finding values of a set of variables such that a certain condition holds is called; (b) SAT; (c) finding a maximal or minimal value; (d) optimizing a path; (e) none of these
4. Bin packing a(n) _____ problem (a) non-optimization vector, matrix or graph; (b) optimization; (c) state-space search; (d) behavior-of-program; (e) interactive computation
5. Convex hull is an attribute of (a) arrays; (b) weighted graphs; (c) unweighted graphs; (d) sets of points; (e) none of these
6. Closest pair is an attribute of (a) arrays; (b) weighted graphs; (c) unweighted graphs; (d) sets of points; (e) none of these
7. Closest-pair is a(n) _____ problem (a) non-optimization vector, matrix or graph; (b) optimization; (c) state-space search; (d) behavior-of-program; (e) interactive computation
8. Graph coloring is a(n) _____ problem (a) non-optimization vector, matrix or graph; (b) optimization; (c) state-space search; (d) behavior-of-program; (e) interactive computation
9. Minimum-path search is a(n) _____ problem (a) non-optimization vector, matrix or graph; (b) optimization; (c) state-space search; (d) behavior-of-program; (e) interactive computation
10. Determining the largest number of items that fit in a knapsack is a(n) _____ problem (a) non-optimization vector, matrix or graph; (b) optimization; (c) state-space search; (d) graph; (e) interactive computation
11. Finding the minimum value that satisfies a certain constraint is a(n) _____ problem (a) constraint; (b) optimization; (c) state-space search; (d) behavior-of-program; (e) interactive computation
12. An optimization problem finds a maximum or minimum value that satisfies a certain (a) formula in predicate logic; (b) constraint; (c) time specification; (d) user; (e) protocol

13. A problem of finding a set of values that yields the highest or lowest return value when used as parameters to a function is (a) constraint satisfaction; (b) optimization; (c) maximization; (d) minimization; (e) central tendency
14. Satisfiability is a(n) _____ problem (a) graph; (b) optimization; (c) state-space search; (d) behavior-of-program; (e) interactive computation
15. Set partition is a(n) _____ problem (a) graph; (b) optimization; (c) state-space search; (d) behavior-of-program; (e) closest-pair
16. Hamiltonian path is a(n) _____ problem (a) vector; (b) optimization; (c) state-space search; (d) behavior-of-program; (e) interactive computation
17. A problem related to propositional-logic formulas is (a) order statistic; (b) closest pair; (c) topological sort; (d) satisfiability; (e) sorting
18. A problem related to graphs is (a) order statistic; (b) closest pair; (c) topological sort; (d) satisfiability; (e) sorting
19. A problem related to points on a plane is (a) order statistic; (b) closest pair; (c) topological sort; (d) satisfiability; (e) sorting
20. A problem related to arrays is (a) order statistic; (b) closest pair; (c) topological sort; (d) satisfiability; (e) convex hull

3. Graph problems
1. A topological sort is applied to (a) graph vertices; (b) an array; (c) a tree; (d) a logical formula; (e) none of these
2. Graph path search involves finding a (a) set of vertices; (b) sequence of vertices; (c) set of edges; (d) minimal set of edges; (e) none of these
3. The prerequisite relationships among required courses in the Computer Science major form a (a) binary tree; (b) linked list; (c) directed acyclic graph; (d) weighted graph; (e) spanning tree
4. Path search is a(n) _____ problem (a) graph; (b) optimization; (c) state-space search; (d) behavior-of-program; (e) interactive computation
5. Minimum spanning tree is an attribute of (a) arrays; (b) weighted graphs; (c) unweighted graphs; (d) sets of points; (e) none of these
6. Labeling vertices of a graph so that no two adjacent vertices have the same label is (a) search; (b) spanning tree; (c) shortest path; (d) graph coloring; (e) bipartite matching

4. Formal specification of algorithmic problems
1. A Hoare triple specifies (a) loop invariant and postcondition; (b) precondition, program and postcondition; (c) program and postcondition; (d) performance requirements; (e) none of these
2. A precondition is asserted to be true (a) before an algorithm executes; (b) at the beginning of every iteration of a loop; (c) after an algorithm executes; (d) all the above; (e) none of the above
3. A postcondition (a) should be true before an algorithm executes; (b) is asserted to be true at the beginning of every iteration of a loop; (c) should be true after an algorithm executes; (d) all the above; (e) none of the above
4. A graph whose vertices may be partitioned into $V_1$ and $V_2$ so that all edges in the graph have one end in $V_1$ and the other in $V_2$ is (a) a cover; (b) a spanning tree; (c) cyclic; (d) connected; (e) bipartite.

5. The independent-set problem is to ___ pairwise conflicts
(a) construct; (b) encourage; (c) avoid; (d) resolve; (e) count

6. A matching is what kind of problem? (a) numeric; (b) array; (c) graph; (d) list; (e) tree

7. A subset of edges in a graph s.t. no two edges share a vertex is (a) sort; (b) tree; (c) cover; (d) matching; (e) dag

8. A matching might represent (a) a tree; (b) common interests among people; (c) a hierarchy; (d) a path; (e) a logic formula

9. A flow network is a (a) tree with a cycle; (b) cover; (c) spanning tree; (d) graph with source and sink vertices; (e) graph with unlabeled edges

10. The maximum-flow problem maximizes (a) capacities of edges; (b) weights of vertices; (c) path length; (d) path weight; (e) cyclicity

11. Labeling vertices of a graph so that no two adjacent vertices have the same label is (a) search; (b) spanning tree; (c) shortest path; (d) graph coloring; (e) bipartite matching

12. Graph coloring is (a) finding shortest path; (b) ensuring that no two adjacent vertices have the same color; (c) ensuring that no two edges have the same color; (d) bipartite matching; (e) a graphics programming task

5. Interactive problems

1. Web pages in general (a) compute functions; (b) do sorts; (c) provide services; (d) compute traversals; (e) none of these

2. DBMS design is a(n) _____ problem (a) non-optimization problem; (b) optimization; (c) state-space search; (d) behavior-of-program; (e) interactive computation

3. Web-site design is a(n) _____ problem (a) non-optimization problem; (b) optimization; (c) state-space search; (d) behavior-of-program; (e) interactive computation

4. GUI design is a(n) _____ problem (a) non-optimization problem; (b) optimization; (c) state-space search; (d) behavior-of-program; (e) interactive computation

5. Protocol design is a(n) _____ problem (a) non-optimization problem; (b) optimization; (c) state-space search; (d) behavior-of-program; (e) interactive computation

6. Computation Tree Logic supports assertions about (a) algorithms; (b) interactive processes; (c) counted loops; (d) postconditions; (e) loop invariants

7. Temporal logic is used to verify (a) algorithms; (b) specifications; (c) batch programs; (d) reactive systems; (e) formulas in predicate logic

8. Model checking uses (a) propositional logic; (b) predicate logic; (c) Hoare triples; (d) temporal logic; (e) unit testing

9. A Kripke structure diagrams (a) an algorithm; (b) a reactive system; (c) a module hierarchy; (d) a web interaction; (e) a multi-agent system

10. Safety asserts that (a) that desired states will obtain; (b) that undesired states will never occur; (c) termination; (d) total correctness; (e) efficiency

11. Liveness asserts that (a) that desired states will obtain; (b) that undesired states will never occur; (c) termination; (d) total correctness; (e) efficiency

3. Graph problems (challenge)

1. Finding the minimum-weight collection of sets of weighted items whose union is a certain set $U$ is what problem? (a) graph coloring; (b) shortest path; (c) satisfiability; (d) set cover; (e) bipartite matching

2. A subgraph that contains all the vertices of its parent graph is the object of what problem? (a) graph coloring; (b) shortest path; (c) satisfiability; (d) vertex cover; (e) bipartite matching

3. A minimum cut is a set of (a) vertices; (b) graphs; (c) paths; (d) edges; (e) numbers
12. A Kripke structure is (a) a control structure; (b) a transition system; (c) a diagram of an algorithm; (d) a proof; (e) a set of objects and formulas.

13. Model checking is a method to verify (a) sorting; (b) algorithms; (c) proofs; (d) reactive systems; (e) efficiency.

14. Reactive systems may be verified formally by (a) quality assurance; (b) unit testing; (c) model checking; (d) Hoare triples; (e) predicate logic.

15. A computation tree is induced by (a) pseudocode; (b) program code; (c) machine code; (d) a Kripke structure; (e) a proof.

16. Verification of reactive systems may use (a) propositional logic; (b) predicate logic; (c) Computation Tree Logic; (d) fuzzy logic; (e) pseudocode.
**Terminology on topic 1 (problem classes)**

- array insertion
- array search
- bin packing
- closest pair
- convex hull
- function optimization
- graph coloring
- Hamiltonian cycle
- independent set
- knapsack problem
- matrix
- minimum spanning tree
- multi-stream interaction
- optimization
- order statistic
- principle of optimality
- reachability
- reducibility
- satisfiability
- sequential interaction
- service
- set partition
- shortest path
- sort
- string match
- string search
- topological sort
- traversal
- Formal specification
- assertion
- liveness
- model checking
- postcondition
- precondition
- predicate logic
- quantifier
- safety
- temporal logic

**Questions to assess topic 1 objectives**

**Topic objective:**
Describe, categorize, and formally specify a variety of computational problems.

**1.1a Describe an array problem**

Describe some formal aspects of the following problems:

1. string matching
2. bitwise OR
3. quicksort partition
4. longest common subsequence
5. closest pair
6. order statistic
7. bitwise complement
8. expression evaluation
9. priority queue
10. satisfiability
11. knapsack

**1.1b Solve a problem in combinatorics**

Show your work:

1. How many possible sum outcomes exist in the rolling of three dice, where order matters?
2. How many possible orderings exist of four distinct names?
3. How many combinations of six letters exist, taken three at a time?
4. If you flipped a coin forty times, recording results as 0 or 1, how many distinct binary numerals could be generated?
5. How many four-character passwords are possible, given the upper and lower case letters plus the digits?
6. How many games of tic-tac-toe are possible, given that the first player has nine options, the second has eight, and so forth?

**1.2a Distinguish constraint from optimization problems**

Distinguish optimization problems from constraint-satisfaction problems, giving examples.

**1.2b Evaluate a formula and determine its satisfiability**

(a) Evaluate the following expressions, given the variable assignments. (b) Find one expression that evaluates to false and show that it is satisfiable or not satisfiable.

1. \(-p \land q \lor r \land p\)  \(t, f, t\)
2. \((p \rightarrow q) \land p \land \neg q \land r\)  \(t, f, f\)
3. \(-p \land (-q \lor r) \land \neg (q \rightarrow r)\)  \(t, f, t\)
4. \(p \lor \neg (q \land \neg r)\)  \(t, f, t\)
5. \((p \lor \neg q) \land \neg r \lor p\)  \(t, f, t\)
6. \((p \land \neg q) \rightarrow r \land q\)  \(t, f, t\)

**1.3a Describe a graph problem**

Describe the following problem, including inputs and outputs:

1. path search
2. minimum spanning tree
3. Hamiltonian path
4. traveling salesperson
5. single-source shortest-path
6. reachability
7. cycle detection
8. topological sort of a dag
9. graph coloring
10. graph traversal

1.3b Describe an advanced graph problem
Describe the following problem, giving inputs and outputs:
1. minimum set cover
2. vertex cover
3. global minimum cut
4. find number of connected components
5. bipartiteness
6. bipartite matching
7. independent set
8. maximum matching
9. stable matching
10. maximum flow

1.4 Use preconditions and postconditions to specify a computational problem**
Using the languages of predicate logic (with quantifiers, if appropriate) and algebra, give the formal postcondition for an algorithm (but not the algorithm itself) that accepts an array A, or arrays A and B, and returns value y as stated below:
1. y = A but in ascending order.
2. y = the bitwise AND of A and B, where A and B are bit vectors.
3. y = the bitwise OR of A and B, where A and B are bit vectors.
4. y = true iff any consecutive elements of A are the same.
5. y = true iff all elements of A are the same.
6. y = true iff any two elements of A are the same.
7. y = the maximum value in A.
8. y = true iff the number of 0’s in bit vector A is greater than the number of 1’s.
9. y = the sum of the elements of A.
10. y = the index of the first occurrence of key value x in A, or 0 if not found.
11. y = true iff the first element is the smallest
12. y = true iff A has at least one pair of consecutive elements that are the same
13. y = true iff A consists of alternating 0s and 1s
14. y = true iff A has no pair of consecutive elements that are the same
15. y = true iff every element in A but the first and second is the sum of the previous two
16. y = true iff every element in A is divisible by 3
17. y = true iff some element in A is divisible by 3

1.5 Formally specify a property of an interactive system
Write and describe a Computation Tree Logic formula that applies to a given reactive system and asserts, for formulas φ and ϕ, that on this system:
1. φ will never hold.
2. There is a way for ϕ to become true
3. After the next state transition, φ will hold.
4. Only after φ holds can ϕ cease to be true.
5. There is no way to get into the state φ.
6. There is a way to get out of the state φ.
7. There is no way to avoid getting into state φ.
8. After some future state transition, φ will hold.
9. After some future state transition, φ will not hold.
10. φ will hold as long as ϕ is not true.
11. ϕ or φ will hold in all future states.

Coding projects
1. Write a program to solve any of the problems discussed above.
2. Design and implement data structures for a course schedule and for a set of prerequisite relationships among academic courses in a department. Express in predicate logic the following constraints on a four-year course plan. Use predicate logic to describe the optimal solution.
   a. Each course is to be offered at least once a year.
   b. Respect all prerequisite relationships

Miscellaneous
1. Describe the correspondence between a decision problem and a language.
2. Use the principle of optimality to describe finding the best batting average among players on teams in the major baseball leagues.
3. In what sense can one problem be said to be reducible to another?
4. Describe a verification method commonly used for reactive systems.
5. Relate the following: Computation Tree Logic; temporal logic; Kripke structures; reactive systems.
6. What is Computation Tree Logic used for?
7. What is temporal logic used for and how?
8. What is a Kripke structure?
9. In Computation Tree Logic (CTL), express the assertion that no path exists in which φ will never hold. (Practical example: In an operating system, no process is permanently blocked from access to a requested resource.)
10. What sort of logic is used in model checking?
11. Name the two quantifiers used in predicate logic and the corresponding ones used in CTL.
12. Translate to English:
   a. EFφ
   b. AG¬φ
   c. AXφ

(13-15) Describe formally the following problems (Kleinberg-Tardos, pp. 412ff):
3. Match staff members to holidays so that staffing requirements for holidays are exactly covered.
4. Distribute n injured persons to h hospitals, each within five miles of the injury location, such that each hospital receives a balanced load of persons not more than h / n.
5. Given a set of populated versus safe nodes on a map, determine whether a set of evacuation routes exists such that no two routes share any nodes.
2.0a Recall basic verification concepts* ^M

1. Loops and recurrences

1. For function \( f \), if \( f(n) = \uparrow \), it (a) returns 0; (b) returns an infinite quantity; (c) is defined for \( n \); (d) is undefined for \( n \); (e) is random

2. For function \( f \), if \( f(n) = \downarrow \), it (a) returns 0; (b) returns an infinite quantity; (c) is defined for \( n \); (d) is undefined for \( n \); (e) is random

3. Peano defined \( \mathbb{N} \) (a) by induction; (b) by contradiction; (c) by enumeration; (d) by encryption; (e) as a subset of \( \mathbb{R} \)

4. Any computable function can be defined (a) by induction; (b) by contradiction; (c) by enumeration; (d) by encryption; (e) as a subset of \( \mathbb{R} \)

5. A recurrence defines (a) a set of natural numbers; (b) a logical formula; (c) a computable function; (d) an undecidable problem; (e) none of these

6. A recursive function definition (a) uses a \textit{while} loop; (b) lists all possibilities; (c) contains a call to the function itself; (d) is impossible; (e) is inefficient

7. A recurrence defines a function in a way that suggests (a) selectiveness; (b) sequentiality; (c) exponentiality; (d) an algorithm; (e) redundancy

8. A claim that a counted-loop algorithm may be expressed recursively is (a) false; (b) provably false; (c) a conjecture; (d) a theorem; (e) an axiom

9. \[ f^2(a) = \begin{cases} 0 & \text{if } a \leq 1 \\ 1 + f(a-2) & \text{otherwise} \end{cases} \]

The above is (a) a method call; (b) method definition; (c) proof; (d) recurrence; (e) equation

10. We may define a sequence by relating an element to some of its predecessors using (a) enumeration; (b) concatenation; (c) inequality; (d) a recurrence relation; (e) a flowchart

11. Recurrences define (a) programs; (b) methods; (c) numbers; (d) strings; (e) functions

2. Correctness of algorithms

1. Which are sufficient conditions for algorithm correctness? (a) good programming methodology; (b) customer satisfaction; (c) approval by QA; (d) output is specified function of input; (e) program always halts and output is specified function of input

2. Total correctness is partial correctness plus (a) termination; (b) proof; (c) loop invariant; (d) postcondition; (e) efficiency

3. An assertion is (a) a comment that describes what happens in an algorithm; (b) a command; (c) a claim about the state of the computation; (d) an algorithm; (e) none of these

4. Syntax is (a) structure in pseudocode; (b) the form of a language expression; (c) the meaning of a language expression; (d) the key criterion for correctness; (e) IRS enforced

5. Semantics is (a) structure in pseudocode; (b) the form of a language expression; (c) the meaning of a language expression; (d) the key criterion for correctness; (e) outside computer science

6. An assertion is (a) a comment that tells what occurs in a program; (b) a comment that tells about values of variables and expressions; (c) always true throughout execution of a program; (d) an executable statement; (e) none of the above

7. Whether a program fits its specifications depends on (a) syntactic correctness; (b) semantic correctness; (c) logical provability; (d) QA’s opinion; (e) chance

8. The purpose of assertions in formal verification is to (a) help establish that code is correct; (b) describe what happens in a program; (c) guarantee that a program halts; (d) catch exceptions; (e) all the above

9. Preconditions are (a) if statements; (b) loops; (c) assertions; (d) test programs; (e) always true

10. Postconditions are (a) if statements; (b) loops; (c) assertions; (d) test programs; (e) always true

11. Termination is proven by (a) tests; (b) authority; (c) voting; (d) divergence; (e) convergence

12. Convergence may prove (a) total correctness; (b) termination; (c) computability; (d) time claims; (e) space claims

3. Inductive proofs of correctness

1. A loop invariant is asserted to be true (a) throughout the loop body; (b) at the beginning of every iteration of a loop; (c) is the same as the postcondition; (d) all the above; (e) none of the above

2. An assertion that is true at the start of each iteration of a loop is (a) a precondition; (b) a loop invariant; (c) a postcondition; (d) a loop exit condition; (e) none of these

3. Loop invariants are used in correctness proofs that are (a) by contradiction; (b) inductive; (c) diagonal; (d) constructive; (e) empirical

4. An appropriate loop invariant in an insertion sort is that (a) one particular element is less than its successor; (b) one particular element is the smallest; (c) the entire array is sorted; (d) part of the array is sorted; (e) none of these

5. A useful loop invariant to help prove correctness of a sorting algorithm could be (a) elements out of order are swapped; (b) elements are compared; (c) \( n \) passes occur; (d) the rightmost \( k \) elements are in ascending order; (e) the counter is incremented

6. A loop invariant asserts that (a) the precondition holds; (b) the postcondition holds; (c) a weaker version of the postcondition holds; (d) the algorithm terminates; (e) something occurs

7. Proving the correctness of a nested loop may involve (a) \textit{modus ponens}; (b) a construction; (c) one loop invariant; (d) two loop invariants; (e) null postcondition

8. Inductive proofs of correctness use (a) contradiction; (b) variants; (c) loop invariants; (d) branch invariants; (e) class invariants
4. Formal proof methods and Hoare triples

1. A Hoare triple consists of (a) precondition, loop invariant, postcondition; (b) program, loop invariant, postcondition; (c) precondition, program, postcondition; (d) proof, loop invariant, program; (e) none of these

2. $<\phi> P <\psi>$ is a (a) precondition; (b) loop invariant; (c) postcondition; (d) Hoare triple; (e) first-order logic formula

3. In $<\phi> P <\psi>$, $\phi$ is a (a) precondition; (b) loop invariant; (c) postcondition; (d) Hoare triple; (e) propositional-logic formula

4. In $<\phi> P <\psi>$, $\phi$ is a (a) precondition; (b) loop invariant; (c) postcondition; (d) Hoare triple; (e) propositional-logic formula

5. In $<\phi> P <\psi>$, $P$ is a (a) precondition; (b) loop invariant; (c) postcondition; (d) program; (e) propositional-logic formula

6. A precondition (a) is asserted to be true before an algorithm executes; (b) is asserted to be true at the beginning of every iteration of a loop; (c) is asserted to be true after an algorithm executes; (d) all the above; (e) none of the above

7. A postcondition (a) is asserted to be true before an algorithm executes; (b) is asserted to be true at the beginning of every iteration of a loop; (c) is asserted to be true after an algorithm executes; (d) all the above; (e) none of the above

8. Using a Hoare triple in algorithm verification entails using (a) syntax rules; (b) proof rules; (c) specification methodology; (d) design methodology; (e) research

9. Using $<\phi> P <\psi>$ in algorithm verification entails using (a) syntax rules; (b) proof rules; (c) specification methodology; (d) design methodology; (e) research

10. To show that $<\phi> P <\psi>$, we must show that (a) $P \Rightarrow \phi$; (b) $\phi \Rightarrow P$; (c) $P$ holds after $\psi$ runs, provided that $\phi$ holds before; (d) $\psi$ holds after $P$ runs, provided that $\phi$ holds before; (e) $\psi$ holds after $\phi$ runs, provided that $P$ holds before
Terminology for topic 2 (Verification)

<table>
<thead>
<tr>
<th>assertion</th>
<th>correctness</th>
<th>Hoare triple</th>
<th>induction principle</th>
<th>inductive proof</th>
<th>loop invariant</th>
<th>partial correctness</th>
<th>postcondition</th>
<th>quantifier</th>
<th>total correctness</th>
</tr>
</thead>
</table>

Questions to assess objectives for topic 2

**Topic objective:**
Explain and apply the method of proving correctness of an algorithm inductively, using postcondition and loop invariant.

2.1a Write a simple recurrence**

Write a recurrence to compute the following function:

1. factorial
2. exponentiation
3. sum of elements of an array
4. search of an array
5. true iff all elements of an array are identical
6. maximum element of an array
7. true if an array is in ascending order
8. smallest element of an array
9. product of all elements of array

2.1b Derive a while loop from a recurrence

Write iterative pseudocode or Java code that computes the functions defined below:

1. \( f(a, b) = \)
   \[
   \begin{cases} 
   0 & \text{if } a = 0 \\
   b + f(a - 1, b) & \text{otherwise} 
   \end{cases}
   \]

2.1c Write a recursive method that corresponds to a while loop**†

Write and test a recursive Java method that produces the same result as the loop below:

2.2a Explain the basic terminology of formal verification**

1. What is an assertion, about the state of a repetitive process, that holds at the start of the process and helps to establish that the process spec is satisfied? How is it used?
2. What are three classes of comments that help establish that the specification of an algorithm is satisfied? For each, state where the comment should appear in the code or pseudocode.
3. What is the likely relationship between a loop invariant and a postcondition?
4. Explain the relationship between loop invariants and induction.
5. Identify the components and meaning of \(<\phi> P <\psi>\) as discussed in this class.
6. Describe partial correctness and distinguish it from total correctness.
7. Describe a mathematical technique or notation, for defining a function, that converts straightforwardly into code or pseudocode.
2.2b Prove that an algorithm terminates*
Referring to an algorithm under objective 2.1c above, prove that the algorithm terminates.

2.3a Given an algorithm and a postcondition, write an appropriate loop invariant*
For an algorithm among the following:
(a) Write a valid postcondition (see topic 1)
(b) Write a valid loop invariant that helps establish that the postcondition holds
(c) Explain the proof of correctness.

1. Quotient (a, b)
   > Performs int division
   \[ y \leftarrow 0 \]
   \[ s \leftarrow b \]
   while \( s < a \):
   \[ s \leftarrow s + b \]
   \[ y \leftarrow y + 1 \]
   return \( y \)

2. Factorial (x)
   \[ y \leftarrow 1 \]
   \[ i \leftarrow 1 \]
   while \( i < n \):
   \[ y \leftarrow i \times y \]
   \[ i \leftarrow i + 1 \]
   return \( y \)

3. Allsame (A)
   \[ y \leftarrow true \]
   for \( i \leftarrow 2 \) to \( |A| \):
   \[ y \leftarrow false \]
   return \( y \)

4. Is-lower (A, test)
   For \( k \leftarrow 1 \) to \( \text{Size}(A) \):
   if \( A[k] < \text{test} \):
   return \( \text{false} \)
   return \( \text{true} \)

5. Largest-to-right (A)
   > Returns result of moving
   > the largest element of
   > \([A] \) into position \( n \) of \( A \).
   \[ \text{largest} \leftarrow 1 \]
   for \( i \leftarrow 2 \) to \( |A| \):
   if \( A[i] > \text{largest} \):
   \[ \text{largest} \leftarrow i \]
   swap \( A[\text{largest}] \) with \( A[i] \)
   return \( A \)

6. Index-of-largest (A)
   > Returns index of the
   > largest element of \( A \)
   \[ y \leftarrow 1 \]
   for \( i \leftarrow 1 \) to \( |A| - 1 \):
   \[ y \leftarrow i \]
   \[ i \leftarrow i + 1 \]
   return \( y \)

7. Search-stack (S, key)
   > Tells if \( S \) contains key
   > \( \text{false} \)
   > \( \text{true} \)
   \[ \text{found} \leftarrow \text{false} \]
   \[ \text{test} \leftarrow \text{Pop}(S) \]
   if \( \text{test} = \text{key} \)
   \[ \text{found} \leftarrow \text{true} \]
   return \( \text{found} \)

8. Max (A)
   > Returns largest elt of \( A \):
   \[ y \leftarrow A[1] \]
   \[ i \leftarrow 1 \]
   while \( i < |A| \):
   if \( y < A[i] \):
   \[ y \leftarrow A[i] \]
   \[ i \leftarrow i + 1 \]
   return \( y \)

9. Sum (A)
   > Computes sum of array elements
   \[ y \leftarrow 0 \]
   \[ i \leftarrow 1 \]
   while \( i \leq |A| \):
   if \( y < A[i] \):
   \[ y \leftarrow y + A[i] \]
   \[ i \leftarrow i + 1 \]
   return \( y \)

10. Product (x, y)
    \[ \text{result} \leftarrow 0 \]
    for \( i \leftarrow 1 \) to \( x \):
    \[ \text{result} \leftarrow \text{result} + y \]
    return \( \text{result} \)

11. Count-squares(s)
    > Returns the number of
    > spaces in a string, \( s \).
    \[ y \leftarrow 0 \]
    \[ i \leftarrow 1 \]
    while \( i \leq |s| \):
    if \( s[i] = \text{'}\text{' } \)
    \[ y \leftarrow y + 1 \]
    \[ i \leftarrow i + 1 \]
    return \( y \)

12. Which-sort (A)
    for \( i \leftarrow |A| \) down to 0:
    \[ A[i] \leftarrow \text{Lgst-to-rt}(A[1..i]) \]
    You may assume
    that \( \text{Lgst-to-rt} \) moves
    the largest element of its
    parameter to the rightmost position.

2.3b Given an algorithm and a loop invariant, explain why the algorithm is correct**
Use the loop invariant (LI) to
show that these algorithms satisfy their specifications.

1. Product (a, b)
   > Performs multiplication
   \[ y \leftarrow 0 \]
   For \( i \leftarrow 1 \) to \( a \):
   \[ y \leftarrow y \times i \]
   return \( y \)

2. Factorial (n)
   \[ y \leftarrow 1 \]
   For \( i \leftarrow 2 \) to \( n \):
   \[ y \leftarrow y \times i \]
   return \( y \)

3. Ascending (A)
   > Returns true iff \( A \) is sorted
   \[ y \leftarrow true \]
   For \( i \leftarrow 1 \) to \( |A| - 1 \):
   \[ y \leftarrow false \]
   return \( y \)

4. Pow (a, b)
   > Returns \( a^n \)
   \[ y \leftarrow 1 \]
   \[ i \leftarrow 0 \]
   while \( i < b \):
   \[ y \leftarrow a^{i+1} \]
   \[ i \leftarrow i + 1 \]
   return \( y \)

5. Sum (A)
   > Computes sum of \( A \)
   \[ y \leftarrow 0 \]
   \[ i \leftarrow 1 \]
   while \( i \leq |A| \):
   if \( y < A[i] \):
   \[ y \leftarrow y + A[i] \]
   \[ i \leftarrow i + 1 \]
   return \( y \)

6. Max (A)
   > Returns largest elt of \( A \)
   \[ y \leftarrow A[1] \]
   \[ i \leftarrow 1 \]
   while \( i < |A| \):
   if \( y < A[i] \):
   \[ y \leftarrow A[i] \]
   \[ i \leftarrow i + 1 \]
   return \( y \)

7. Search (A, key)
   \[ y \leftarrow false \]
   for \( i \leftarrow 1 \) to \( |A| \):
   if \( A[i] = \text{key} \):
   \[ y \leftarrow true \]
   return \( y \)

8. Log2 (x)
   > Returns \( \log_2(x) \)
   \[ y \leftarrow 0 \]
   \[ i \leftarrow x \]
   while \( i > 0 \):
   \[ y \leftarrow \log_2(x - i) \]
   \[ i \leftarrow \lfloor i/2 \rfloor \]
   \[ y \leftarrow y + 1 \]
   return \( y \)
2.3c Write an algorithm and prove its correctness*

Write an algorithm to compute the following function, and prove its correctness.

1. \( y = ab \), for inputs \( a, b \), using only addition or subtraction.
2. the sum of the elements of an input array, \( A \)
3. A version of the Quicksort Partition step in which first all the values less than the pivot are collected, then the pivot is appended, then the values greater than the pivot are collected.
4. Tell whether an array of integers is in ascending order;
5. prove correctness.
6. linear search
7. the maximum value in an array
8. the factorial function where \( \text{fact}(4) = 4 \times 3 \times 2 = 24 \)
9. summation of \( 1 \ldots x \)
10. the numeric value of a bit vector taken as a binary numeral
11. the smallest value in a sequence of integers
12. find the number of occurrences of a string, \( s \), in a text file, \( t \)
13. find the largest element of a subsequence in an array.

Signature: \( \text{Max}(A, \text{first}, \text{last}) \), where \( A \) is an array, \( \text{first} \) and \( \text{last} \) are subscripts. Algorithm should return the largest element of the sequence from \( A[\text{first}] \) to \( A[\text{last}] \), inclusive.

15. Write a method that accepts as parameters a sorted array of integers, its occupancy (current size), and a value to be inserted. The method should insert the value at its appropriate location. Write preconditions, postconditions, and loop invariants to argue that the method is correct.

2.4 Explain formal rules for algorithm correctness proofs

Identify and put in plain English:

1. \( \langle \phi \land b \rangle P \langle \phi \rangle \Rightarrow \langle \phi \rangle \text{ while } b \{ P \} \langle \phi \land \neg b \rangle \)
2. \( \langle \psi[E/x] \rangle x \leftarrow E \langle \psi \rangle \)
3. \( \langle \phi \rangle P_1 \langle \eta \rangle \land \langle \eta \rangle P_2 \langle \psi \rangle \Rightarrow \langle \phi \rangle P_1 P_2 \langle \psi \rangle \)
4. \( \langle \phi \land b \rangle P_1 \langle \psi \rangle \land \langle \phi \land \neg b \rangle P_2 \langle \psi \rangle \Rightarrow \langle \phi \rangle \text{ if } b \{ P_1 \} \text{ else } \{ P_2 \} \langle \psi \rangle \)
5. Describe the meaning of \( \langle \phi \rangle P < \psi \rangle \), and how a program may be formally verified using this.
6. Describe the formal process by which Hoare triples are used in program verification.
7. Explain how loop invariants are used with \( \langle \phi \rangle P < \psi \rangle \) to establish program correctness.

Coding problems

Write a program to solve any of the problems discussed above. Give the number of test items that would be required to assure correctness.
Multiple-choice questions on Topic 3 (Algorithm analysis)

3.0a Recall basic algorithm-analysis concepts*\textsuperscript{M}

1. **Algorithm analysis and Big-O notation**

1. Function $g$ is an upper bound on function $f$ iff for all $x$, (a) $g(x) \leq f(x)$; (b) $g(x) \geq f(x)$; (c) $g = O(f)$; (d) $f = \Omega(g)$; (e) $x \leq f(x)$

2. Function $g$ is a lower bound on function $f$ iff for all $x$, (a) $g(x) \leq f(x)$; (b) $g(x) \geq f(x)$; (c) $f = O(g)$; (d) $g = \Omega(f)$; (e) $x \leq f(x)$

3. Big-Omega notation expresses (a) tight bounds; (b) upper bounds; (c) lower bounds; (d) worst cases; (e) none of these

4. Big-O notation expresses (a) tight bounds; (b) upper bounds; (c) lower bounds; (d) best cases; (e) none of these

5. Theta notation expresses (a) tight bounds; (b) upper bounds; (c) lower bounds; (d) worst cases; (e) none of these

6. $T_\alpha(n) = \Theta(f(n))$ means that (a) algorithm $\alpha$ computes function $f$; (b) algorithm $\alpha$ produces a result in time at least $f(n)$ for inputs of size $n$; (c) algorithm $\alpha$ produces a result in time not greater than $f(n)$ for inputs of size $n$; (d) algorithm $T$ runs in time $\alpha$; (e) algorithm $f$ computes function $T$ on data $\alpha$

7. The theorm, $T_1(n) \in \Theta(g_1(n)) \land T_2(n) \in \Theta(g_2(n)) \Rightarrow T_1(n) + T_2(n) \in \Theta(\max\{g_1(n), g_2(n)\})$ says that (a) the slower and faster parts of an algorithm together set its running time; (b) the faster part of an algorithm dominates in determining running time; (c) the slower part of an algorithm dominates in determining running time; (d) Algorithm $T$ computes functions $g_1$ and $g_2$; (e) Algorithm $T$ finds the maximum of $g_1$ and $g_2$

8. $\log_2 \mathcal{N} \in O(\sqrt{n})$ means that the logarithm function grows as fast as; (a) grows no faster than; (c) grows as fast as; (d) is in a mapping of real numbers defined by; (e) regardless of parameter produces a result smaller than

9. **Empirical analysis of algorithms can be done using**

(a) theorems only; (b) system clock; (c) loop invariants; (d) reductions; (e) none of these

10. **Empirical analysis of algorithms can be done using**

(a) theorems only; (b) counter variables; (c) loop invariants; (d) reductions; (e) none of these

11. Quadratic time is faster than (a) $\Theta(1)$; (b) $\Theta(\log n)$; (c) $\Theta(n^2)$; (d) $\Theta(n^3)$; (e) none of these

12. Analysis (a) computes a function; (b) separates something into parts; (c) puts components together; (d) writes a program; (e) is the entire problem-solving process

13. **Best case for an algorithm (a) takes the same time for all data; (b) assumes the data that the algorithm handles in the greatest time; (c) assumes the data that the algorithm handles in the least time; (d) is the expected time considering all possible input data; (e) none of these**

14. **Worst case for an algorithm (a) takes the same time for all data; (b) assumes the data that the algorithm handles in the greatest time; (c) assumes the data that the algorithm handles in the least time; (d) is the expected time considering all possible input data; (e) none of these**

15. Average case for an algorithm (a) takes the same time for all data; (b) assumes the data that the algorithm handles in the greatest time; (c) assumes the data that the algorithm handles in the least time; (d) is the expected time considering all possible input data; (e) none of these

16. Bubble sort’s worst-case running time function is determined by (a) a single loop; (b) nested loops; (c) a series of loops; (d) the input data; (e) none of these

17. A loop nested to two levels, each with roughly $n$ iterations, has running time (a) $O(1)$; (b) $O(n)$; (c) $O(n^2)$; (d) $O(n \lg n)$; (e) $O(2^n)$

18. A loop nested to $n$ levels has running time (a) $\Theta(1)$; (b) $\Theta(n)$; (c) $\Theta(n^2)$; (d) $\Theta(n \lg n)$; (e) $\Theta(2^n)$

19. The running time function of an algorithm is determined by (a) the number of operations in a sequence structure; (b) the number of branches in a selection structure; (c) the time of the slowest of a series of loops; (d) the data; (e) none of these

2. **Recurrence relations and time functions**

1. Recurrences are used in (a) input specification; (b) proofs of correctness; (c) time analysis; (d) type checking; (e) none of these

2. Recurrences may help in time analysis if we find (a) count of iterations of while loop; (b) clock readings; (c) exit condition; (d) depth of recursion; (e) none of these

3. Recurrence relations enable us to use _____ to obtain running time (a) empirical tests; (b) loop nesting; (c) base-case running time; (d) depth of recursion; (e) base-case running time and depth of recursion

4. The more time-consuming part of the execution of an algorithm defined by a recurrence is (a) the base step; (b) the recursive step; (c) calculation of the time function; (d) proof of correctness; (e) design

5. Recurrences (a) are a form of pseudocode; (b) suggest algorithms but not running time; (c) suggest running time but not algorithms; (d) suggest running time and algorithms; (e) none of these

6. When base case is $O(1)$ and remaining work of an algorithm is cut in half at each step, the running time is (a) $O(1)$; (b) $O(\log n)$; (c) $O(n \log n)$; (d) $O(n)$; (e) $O(n^2)$

7. When the running time for the base case of a recursive algorithm is $O(n)$ and the remaining part of input to process is reduced by one at each recursive step, the total running time is (a) $O(1)$; (b) $O(\log n)$; (c) $O(n \log n)$; (d) $O(n)$; (e) $O(n^2)$

8. In a recursive algorithm, when the running time for the base case is $O(1)$ and remaining work of an algorithm is reduced by one at each step, the running time is (a) $O(1)$; (b) $O(\log n)$; (c) $O(n \log n)$; (d) $O(n)$; (e) $O(n^2)$

9. In a recursive algorithm, when the running time for the base case is $O(n)$ and remaining part of input to process is reduced by one at each recursive step, the total running time is (a) $O(1)$; (b) $O(\log n)$; (c) $O(n \log n)$; (d) $O(n)$; (e) $O(n^2)$

10. What is the solution for $T(n) =$

\[
\begin{cases}
2 & \text{if } n = 1 \\
2 + T(n-1) & \text{otherwise}
\end{cases}
\]

(a) $O(1)$; (b) $O(\log n)$; (c) $O(n \log n)$; (d) $O(n)$; (e) $O(n^2)$
11. What is the solution for $T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ n + T(n-1) + 3 & \text{otherwise} \end{cases}$ 
(a) $O(1)$; (b) $O(\log n)$; (c) $O(n \log n)$; (d) $O(n)$; (e) $O(n^2)$

12. What is the solution for $T(n) = \begin{cases} 1 & \text{if } n \leq 2 \\ 2 + T((n-1)/2) & \text{otherwise} \end{cases}$ 
(a) $O(1)$; (b) $O(\log n)$; (c) $O(n \log n)$; (d) $O(n)$; (e) $O(n^2)$

13. What is the solution for $T(n) = \begin{cases} 1 & \text{if } n = 0 \\ n + ((n-1)/2) & \text{otherwise} \end{cases}$ 
(a) $O(1)$; (b) $O(\log n)$; (c) $O(n \log n)$; (d) $O(n)$; (e) $O(n^2)$

3. Brute force and nested loops

1. The approach to algorithm design that addresses combinatorial problems in the most straightforward way is (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic

2. Performing an $O(1)$ operation on each of a series of $4n$ values is $O(\text{___})$ (a) $1$; (b) $\log n$; (c) $n$; (d) $n^2$; (e) $4n$

3. Decrease-by-one algorithms use (a) divide and conquer; (b) brute force; (c) the greedy approach; (d) dynamic programming; (e) a probabilistic approach

4. Adding the elements of an array has an efficient ______ algorithm (a) brute-force; (b) divide-and-conquer; (c) greedy; (d) dynamic-programming; (e) probabilistic

5. Finding the maximum element of an unsorted array has an efficient ______ algorithm (a) brute-force; (b) divide-and-conquer; (c) greedy; (d) dynamic-programming; (e) probabilistic

6. Matrix multiplication by brute force requires (a) $O(n)$ time; (b) $O(1)$ time; (c) $O(\log n)$ time; (d) nested loops; (e) recursion

7. Brute-force algorithms make use of (a) a straightforward solution to a problem by examining all possible solutions; (b) the fact that an optimal solution can be constructed by adding the cheapest next component, one at a time; (c) the fact that data is arranged so that at each step, half the remaining input data can be disposed of; (d) effort can be saved by saving the results of previous effort in a table; (e) none of these

8. The linear search is what kind of algorithm? (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic

9. Vector traversal is $O(\text{___})$ (a) 1; (b) $\log n$; (c) $n$; (d) $n^2$; (e) $2^n$

10. Inserting an element in a sorted array so that it stays sorted is (a) $O(1)$; (b) $O(\log n)$; (c) $O(n)$; (d) $O(n \log n)$; (e) $O(n^2)$

11. An good array insertion algorithm uses (a) divide and conquer; (b) brute force; (c) the greedy approach; (d) dynamic programming; (e) a probabilistic approach

12. String search using string matching employs (a) divide and conquer; (b) brute force; (c) the greedy approach; (d) dynamic programming; (e) a probabilistic approach

13. Linear search is $O(\text{___})$ (a) 1; (b) $\log n$; (c) $n$; (d) $n^2$; (e) $2^n$

14. A recursive-case running time of $(1 + T(n-1))$ indicates _____ time (a) constant; (b) logarithmic; (c) linear; (d) quadratic; (e) exponential

15. What is the running time of a brute-force algorithm that tells whether an array is sorted? (a) $O(1)$; (b) $O(\log n)$; (c) $O(n)$; (d) $O(n \log n)$; (e) $O(n^2)$

16. What is the running time of a brute-force algorithm that finds the minimum element of an array? (a) $O(1)$; (b) $O(\log n)$; (c) $O(n)$; (d) $O(n \log n)$; (e) $O(n^2)$

17. What is the running time of a brute-force algorithm that number of occurrences of “1” in an array? (a) $O(1)$; (b) $O(\log n)$; (c) $O(n)$; (d) $O(n \log n)$; (e) $O(n^2)$

18. What is the running time of a brute-force algorithm that finds the intersection of two arrays? (a) $O(1)$; (b) $O(\log n)$; (c) $O(n)$; (d) $O(n \log n)$; (e) $O(n^2)$

19. What is the running time of a brute-force algorithm that tells whether all element of an array are the same? (a) $O(1)$; (b) $O(\log n)$; (c) $O(n)$; (d) $O(n \log n)$; (e) $O(n^2)$

20. Decrease-by-one algorithms exploit the relationship between (a) input and output; (b) loops and branches; (c) $f(1)$ and $f(n)$; (d) $f(n)$ and $f(n-1)$; (e) $f(n)$ and $f(\log n)$

21. Computing the factorial function has an efficient ______ algorithm (a) brute-force; (b) divide-and-conquer; (c) greedy; (d) dynamic-programming; (e) probabilistic

22. Insertion sort is (a) $O(1)$; (b) $O(\log n)$; (c) $O(n)$; (d) $O(n^2)$; (e) $O(2^n)$

23. The Bubble sort is what kind of algorithm? (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic

24. The insertion sort is what kind of algorithm? (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic

25. Selection sort is $O(\text{___})$ (a) 1; (b) $\log n$; (c) $n$; (d) $n^2$; (e) $2^n$

26. Selection sort uses (a) divide and conquer; (b) brute force; (c) the greedy approach; (d) dynamic programming; (e) a probabilistic approach

27. A recursive-case running time of $(n + T(n-1))$ indicates _____ time (a) constant; (b) logarithmic; (c) linear; (d) quadratic; (e) exponential

28. Bubble sort is (a) $O(1)$; (b) $O(\log n)$; (c) $O(n)$; (d) $O(n^2)$; (e) $O(2^n)$

3.0b Recall advanced algorithm-analysis concepts

4. Exhaustive search

1. The simpler-to-design solution to the closest-pair problem discussed in this course is by (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic

2. Solving the convex-hull problem by generating all sequences of points and checking each sequence to see if it satisfies the convex-hull constraint is (a) divide and conquer; (b) brute force; (c) the greedy approach; (d) dynamic

3. Solving the closest-pair problem by generating all pairs of points and selecting the pair that is closes together is (a) divide and conquer; (b) brute force; (c) the greedy approach; (d) dynamic programming; (e) a probabilistic approach

4. Exhaustive search is what kind of algorithm? (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic

5. Solving the Traveling Salesperson Problem by comparing costs of all paths is (a) divide and conquer; (b) exhaustive search; (c) the greedy approach; (d) dynamic programming; (e) a probabilistic approach
6. A recursive-case running time of \((1 + 2T(n-1))\) indicates ___ time (a) constant; (b) logarithmic; (c) linear; (d) quadratic; (e) exponential

7. Brute force may often use tactics designed for efficiency; (a) exhaustive search; (b) tree search; (c) binary search; (d) shortcuts; (e) none of these

8. In state-space search, state may be (a) a geographical entity; (b) a temperature; (c) a set of values; (d) a choice among five alternatives; (e) none of these

9. The running time of a brute-force solution to the satisfiability problem is (a) \(\Theta(1)\); (b) \(\Theta(n)\); (c) \(\Theta(n^2)\); (d) exponential in \(n\); (e) none of these

10. A known brute-force solution to SAT (a) \(O(1)\); (b) \(O(n)\); (c) \(O(n^2)\); (d) \(O(n \lg n)\); (e) \(O(2^n)\)

11. Independent-set may be solved by (a) sorting; (b) searching; (c) comparing each element with each other one; (d) an \(O(n^2)\) algorithm; (e) an \(O(2^n)\) algorithm
# Terminology for topic 3 (Algorithm analysis)

- **algorithm analysis**
- **asymptotic behavior**
- **average case**
- **best case**
- **big-O notation**
- **big-Omega notation**
- **brute force**
- **brute-force closest-pair**
- **brute-force convex hull**
- **brute-force knapsack**
- **brute-force Traveling Salesperson**
- **double recursion**
- **empirical analysis of algorithms**
- **exhaustive search**
- **expected running time**
- **extra-large time O(2n)**
- **Insert-ascending insertion sort**
- **depth of recursion**
- **linear search**
- **linear time O(n)**
- **lower bound**
- **matrix multiplication**
- **quadratic time O(n^2)**
- **recurrence relation**
- **running time**
- **running-time recurrence**
- **theta notation**
- **tight bound**
- **time function**
- **upper bound**
- **worst case**
- **Towers of Hanoi Brahma**

## Questions to assess objectives for Topic 3

### 3.1 Define and use the big-O, theta, and big-omega notations**

(1-3) Define
1. O notation
2. \( \Omega \) notation
3. \( \Theta \) notation

(4-14) Explain why each of the following is true or false:
4. \( f(n) = 1 \in O(n) \)
5. \( f(n) = \log n \in O(n^2) \)
6. \( f(n) = n^2 \in O(1) \)
7. \( f(n) = n^2 \in O(n^5) \)
8. \( f(n) = n^2 \in O(n) \)
9. \( f(n) = 4n^2 \in O(n^2) \)
10. \( f(n) = 3n + 10 \in O(\log n) \)
11. \( f(n) = n \log n \in O(n^2) \)
12. \( f(n) = n^2/8 \in O(n \log n) \)
13. \( f(n) = 100 \in O(n^3) \)
14. \( f(n) = n \in O(n^5) \)

15. In mathematical notation, list some of the most widely discussed categories that classify algorithms by performance.

16. Use two simple formulas to show the relation among O, \( \Theta \), and \( \Omega \) notations. (Hint: if \( f(n) \in \Omega(g(n)) \), what follows?)

17. What is the main notation for expressing the complexity of algorithms as tight bounds, and what is the meaning of this notation in terms of the other commonly used notations?

18. What is a lower bound on a function \( f \)? A tight bound?

Distinguish big-O, \( \Omega \), and \( \Theta \), and relate to algorithm analysis.

19. For function \( f \), \( O(f) \), \( \Omega(f) \), and \( \theta(f) \) are sets of functions. What are the set containment relationships among the following:
   (a) \( O(n^3) \), \( O(\log n) \)
   (b) \( \Omega(n^2) \), \( \Omega(\log n) \)
   (c) \( \theta(n^3) \), \( \theta(\log n) \)

20. Tell which are true:
   (a) \( O(n^3) = O(2n^2) \)
   (b) \( O(n^3) = O(100n) \)
   (c) \( O(50n) = O(n / 2) \)
   (d) \( O(n) = O(\log n) \)
   (e) \( O(\log n) = O(\log n + 1000) \)

21. Using the definition of big-O notation, explain why \( f(n) = 4n^2 \in O(n^5) \) is true or false.

22. Using the definition of \( \Theta \), explain why
   (a) \( \Theta(n / 2) = \Theta(n) \)
   (b) \( \Theta(3n) = \Theta(n) \)
   (c) \( \Theta(n^2 - 5) = \Theta(n^2) \)

23. Express in big-O notation:
    \( T(n) = 500n + n^2 / 10 + 80 \log n \)

24. Give an example of a lower bound on the function \( f(x) = x^3 \)

25. Referring to other notations and their meanings, explain why \( \theta(f) \) called a tight bound on \( f \).

26. What is the main notation for expressing the complexity of algorithms as lower bounds, and what is the intuitive meaning of this notation?

27. What is the main notation for expressing the complexity of algorithms as upper bounds, and what is the intuitive meaning of this notation?

28. Give the names of five asymptotic efficiency classes, in ascending order of time complexity.

29. Group the following time functions by efficiency class:
   (a) \( t(n) = 6 \log n \)
   (b) \( t(n) = 5n^3 \)
   (c) \( t(n) = n / 4 \)
   (d) \( t(n) = 2^n \)
   (e) \( t(n) = 10n \log n + 100 \)
   (f) \( t(n) = 80n + 1000 \)

30. Simplify (a) \( O(2n) \); (b) \( O(n^8) \); (c) \( O(4n^2 + 2n) \)

### 3.2 Write and solve a recurrence for the time function of a linear-time algorithm**

Given a recurrence shown in a problem in 2.1b, write a recurrence for the time function.
3.3a Explain the brute-force approach to algorithm design*

1. Describe brute force, as discussed in this course.
2. Name and describe the algorithm design approach used in the basic sorting algorithms.
3. Describe the simplest algorithm-design approach, in which the design follows in a straightforward way from the problem definition.
4. Name and describe a simple design approach that is easily used in converting a simple description to pseudocode or programming code.
5. Describe a brute-force approach to searching or sorting an array. Give its main features, applicable to other problems.
6. Describe exhaustive search and associate it with an algorithm-design approach.

3.3b Describe the running time of a nested-loop algorithm*

Write pseudocode, and describe the running time, of an algorithm that, without using utility methods

1. given input of n, displays the first n prime numbers
2. puts an array in descending order
3. returns true iff any two elements in an array have the same value
4. solves the closest-pair problem
5. solves the convex-hull problem
6. finds the length of the longest series of values of elements in an array that are all the same
7. determines the winner in a vote for candidates coded by integers in an array
8. finds all the divisors of integer n.
9. finds the starting index of the longest ascending run of side-by-side elements in an array
10. finds the intersection of the sets stored in two arrays of integers.
11. tells whether a given destination vertex in a graph is reachable from a given source vertex
12. finds the longest common subsequence of two arrays
13. finds the mode (most frequent value) of an array
14. finds the index of the longest ascending subsequence in an array
15. uses a loop that inserts values successively into a sorted initial sequence until this sequence is the whole array.
16. finds the median (middle) value in an array
17. given unsorted array A, returns the subscripts of the start and end of the longest run of adjacent array elements off A that are in ascending order
18. finds the index of the longest subsequence of identical elements in an array
19. performs the bubble sort
20. tells whether string s1 is a substring of s2
21. tells whether a given destination vertex in a graph is reachable from a given source vertex.
22. finds the disjunction (OR) of two arrays of integers, without sorting.

3.3c Write and solve a time recurrence for a nested-loop algorithm*

(1-20) Write a time recurrence for the algorithm you wrote for objective 3.3b above. Solve the recurrence, explaining your solution.

21. (a) Express the algorithm below as a recurrence.
(b) Based on (a), write a recurrence to express the time function of the algorithm, and solve the recurrence using big-O or \( \Theta \) notation.

```plaintext
num_passes \leftarrow 0
Repeat
    swapped \leftarrow false
    for i \leftarrow 1 to size(A) - 1
        if A[i] > A[i + 1]
            swap(A[i], A[i + 1])
            swapped \leftarrow true
    num_passes \leftarrow num_passes + 1
until swapped
```

22. Levitin, p. 76, Ex. 1 (b, c) (solve recurrences)
23. Levitin, p. 76, Ex. 3 (solve recurrences)

3.4a Describe an algorithm that performs an exhaustive search

Describe a brute-force solution to the following (described in topic 1):

1. the independent-set problem
2. partitioning a set of numbers into two subsets, each with the same sum
3. whether two strings have a common subsequence (not necessarily of contiguous characters)
4. whether a given set of natural numbers may be partitioned into two subsets of equal sum.
5. the stable-marriage problem
6. the knapsack problem (placing items, each with weight and value, in a container so as to maximize total value without exceeding maximum weight)
7. satisfiability (determining whether a formula in propositional logic has any variable assignment that makes it true)
8. finding the closest pair of points among a set of points on a coordinate axis
9. problem of finding a path in a graph that passes through each vertex exactly once
21. convex hull (finding a set of points, in a larger set of points that would form a polygon enclosing the larger set)
22. Levitin, pp. 59-60, Ex. 1, 2(b, c), 3 (b, c, d), 5, 8(a). p. 67. Ex. 1 (g) (summations; show work), 2 (d) (summation; show work), 4 (mystery algorithm)

3.4b Code and test an exhaustive-search algorithm†

Write and test a program to solve any of the problems listed under 3.4a above. Give the number of test items that would be required to assure correctness.
Multiple-choice questions on **Topic 4 (Divide and conquer)**

### 4.0a Recall basic divide-and-conquer concepts

#### 1. Very fast algorithms

1. The approach to algorithm design that uses decrease by a constant factor is (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic

2. The approach to algorithm design that often enables a quickly converging search of the solution space by ruling out a large part of the space at each step is (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic

3. The binary search uses which approach to algorithm design? (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic

4. After each step of the binary search, the quantity of remaining data to be searched is on average (a) 1; (b) lg n; (c) n + 2; (d) n; (e) 2n

5. Binary search can be shown to be $\Theta(lg n)$ by (a) Hoare triples; (b) temporal logic; (c) predicate logic; (d) the Master Theorem (Main Recurrence Theorem); (e) induction

6. Divide-and-conquer algorithms make use of (a) a straightforward solution to a problem by examining all possible solutions; (b) the fact that an optimal solution can be constructed by adding the cheapest next component, one at a time; (c) the fact that data is arranged so that at each step, half the remaining input data can be disposed of; (d) effort can be saved by saving the results of previous effort in a table; (e) none of these

7. When a running time of the base step of a recursive algorithm is constant, and the remaining data to be processed at each step is reduced by half, the algorithm’s running time is $O(\ldots)$ (a) 1; (b) lg n; (c) n; (d) $n^2$; (e) $2^n$

8. A recursive-case running time of $(1 + T(n+2))$ indicates (a) time (a) constant; (b) logarithmic; (c) linear; (d) quadratic; (e) exponential

9. Decrease by constant factor is consistent with $T(n) =$ (a) $T(n - 1) + O(1)$; (b) $T(n - 1) + O(n)$; (c) $T(n/b) + O(1)$; (d) $T(n/b) + f(n)$ is consistent with (a) decrease by constant; (b) decrease by one; (c) decrease by constant factor; (d) $O(n^2)$; (e) $O(n)$

10. $T(n) = T(n/b) + f(n)$ is consistent with (a) decrease by constant; (b) decrease by one; (c) decrease by constant factor; (d) $O(n^2)$; (e) $O(n)$

11. Binary search is of the algorithm time pattern (a) decrease by constant; (b) decrease by one; (c) decrease by constant factor; (d) $O(n^2)$; (e) $O(n)$

12. The Master Theorem (Main Recurrence Theorem) (a) is used to prove correctness of algorithms; (b) gives general solutions to time recurrences for divide-and-conquer algorithms; (c) gives intractability results; (d) is proven by temporal logic; (e) none of these

13. The height of the decision tree for an algorithm expresses its (a) correctness; (b) problem class; (c) running time; (d) space requirement; (e) data arrangement

14. A structure that shows possible outcomes of all steps of a computation is a (a) flowchart; (b) module hierarchy; (c) binary tree; (d) decision tree; (e) none of these

15. A recursive-case running time of $(n + T(n+2))$ indicates (a) time (a) constant; (b) $n lg n$; (c) linear; (d) quadratic; (e) exponential

16. When base case is $O(n)$ and remaining work of an algorithm is cut in half at each step, the running time is (a) $O(1)$; (b) $O(lg n)$; (c) $O(n lg n)$; (d) $O(n)$; (e) $O(n^2)$

17. The approach to algorithm design that enables faster solutions to the convex hull and closest-pair problems is (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic

#### 2. Vector problem solutions

1. Quicksort is of the algorithm type (a) decrease by constant; (b) decrease by one; (c) decrease by constant factor; (d) $O(n^2)$; (e) $O(n)$

2. The Quicksort partition step is (a) $O(1)$; (b) $O(lg n)$; (c) $O(n)$; (d) $O(n^2)$; (e) $O(2^n)$

3. In average case, Quicksort is (a) $O(1)$; (b) $O(lg n)$; (c) $O(n lg n)$; (d) $O(n^2)$; (e) $O(2^n)$

4. In worst case, Quicksort is (a) $O(1)$; (b) $O(lg n)$; (c) $O(n lg n)$; (d) $O(n^2)$; (e) $O(2^n)$

5. Which sort has an execution time proportional to $n \times \log_2(n)$? (a) insertion sort; (b) bubble sort; (c) selection sort; (d) insertion sort; (e) Quicksort

6. Quicksort uses which approach to algorithm design? (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic

7. The solution to convex-hull that recursively finds triangles defined by farthest points is an instance of (a) brute force; (b) divide and conquer; (c) greedy algorithms; (d) dynamic programming; (e) probabilistic algorithms

8. Finding closest pair by recursively finding closest pair in each of two vertical areas, then choosing closest, is an instance of (a) brute force; (b) divide and conquer; (c) greedy algorithms; (d) dynamic programming; (e) probabilistic algorithms

9. Finding the third-highest element of an array is which problem? (a) average; (b) search; (c) mode; (d) order statistic; (e) maximization

10. Finding the median of an array is which problem? (a) average; (b) search; (c) mode; (d) order statistic; (e) maximization

#### 3. Tree-based algorithms

1. The BST search uses which approach to algorithm design? (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic

2. After each step of the BST search, the quantity of remaining data to be searched is on average (a) 1; (b) lg n; (c) $n + 2$; (d) n; (e) $2n$

3. An example of the efficiency of divide and conquer is (a) linear search; (b) bubble sort; (c) merge; (d) BST search; (e) exhaustive search

4. The height of a BST is on average $O(\ldots)$ (a) 1; (b) lg n; (c) n; (d) $n lg n$; (e) $n^2$

5. The complexity of a search of a balanced binary search tree of $n$ nodes is $O(\ldots)$ (a) log n; (b) n; (c) $n log n$; (d) $n^2$
6. Name a kind of binary tree in which all leaves are at most one level apart (a) module hierarchy; (b) complete; (c) shallow; (d) deep; (e) random
7. What is a binary tree in which the bottom-row nodes are filled at the left? (a) module hierarchy; (b) complete; (c) shallow; (d) deep; (e) random
8. To implement a priority queue, it is considered most time-efficient to use a (a) simple vector; (b) linked list; (c) heap; (d) stack; (e) hash table
9. The figure below is a (a) linked list; (b) minimum heap; (c) binary search tree; (d) hash table; (e) maximum heap

```
2
3
6
```
10. The common implementation of a heap is (a) array; (b) linked list; (c) doubly-linked structure; (d) multi-linked structure; (e) none of these
11. The root of a heap implemented as array A is (a) the first element of A; (b) the last element; (c) a pointer; (d) the node pointed to by a certain pointer; (e) inaccessible
12. The depth of a heap of size \( n \) is close to (a) \( \log n \); (b) \( \log^2 n \); (c) the square root of \( n \); (d) \( n/2 \); (e) \( n^2 \)
13. If a heap node’s subscript is 4, then the subscript of its left child is (a) 1; (b) 2; (c) 3; (d) 4; (e) 8
14. A heap is (a) any array; (b) any tree; (c) a complete binary tree; (d) a binary tree in which no node has exactly one child; (e) none of these
15. The running time for \( \text{heap-extract-min} \) is (a) \( O(1) \); (b) \( O(\log n) \); (c) \( O(n \log n) \); (d) \( O(n) \); (e) \( O(n^2) \)
16. The running time for the algorithm presented in class as \( \text{heapiify} \) is (a) \( O(1) \); (b) \( O(\log n) \); (c) \( O(n \log n) \); (d) \( O(n) \); (e) \( O(n^2) \)
17. The running time for \( \text{heap-sort} \) is (a) \( O(1) \); (b) \( O(\log n) \); (c) \( O(n \log n) \); (d) \( O(n) \); (e) \( O(n^2) \)
18. The running time for \( \text{build-heap} \) is (a) \( O(1) \); (b) \( O(\log n) \); (c) \( O(n \log n) \); (d) \( O(n) \); (e) \( O(n^2) \)
19. A precondition of the \( \text{Heapify} \) operation on node \( x \) is (a) \( x \) is the root of a heap; (b) \( x \)'s left and right children are roots of heaps; (c) \( x \)'s parent is the root of a heap; (d) \( x \) has no parent; (e) \( x \) has no child
20. The \( \text{Heapify} \) operation (a) deletes a node; (b) inserts a node; (c) restores the heap property; (d) sorts a heap; (e) creates a heap from an array
21. The time of \( \text{Heapify}(x) \) is proportional to the (a) size of the heap; (b) depth of the tree; (c) value of \( x \); (d) number of siblings of \( x \); (e) number of children of \( x \)
22. Search of a degenerate tree is (a) constant; (b) logarithmic time; (c) linear time; (d) quadratic time; (e) none of these
23. The heap data structure makes use of an efficient algorithm of what kind? (a) brute-force; (b) divide and conquer; (c) greedy; (d) exhaustive search; (e) probabilistic
24. A full binary tree with \( k \) nonleaves has how many vertices? (a) 1; (b) \( \log_2 k \); (c) \( k/2 \); (d) \( 2k + 1 \); (e) \( k^2 \)

4.0b Recall advanced divide-and-conquer concepts

4. Graph algorithms
1. Graph path search involves finding a (a) set of vertices; (b) sequence of vertices; (c) set of edges; (d) minimal set of edges; (e) none of these
2. To find a path from one vertex to another, we may use (a) breadth-first search; (b) connectivity number; (c) minimal spanning tree; (d) expression tree; (e) search tree
3. Breadth-first search finds (a) an adjacency matrix; (b) a cycle; (c) a tree; (d) a path; (e) a reachability matrix
4. Depth-first search finds (a) an adjacency matrix; (b) a cycle; (c) a tree; (d) a path; (e) a reachability matrix
5. The breadth-first search (a) uses a queue; (b) uses a stack; (c) searches an array; (d) searches a tree; (e) none of these
6. The depth-first search (a) uses a queue; (b) uses a stack; (c) searches an array; (d) searches a tree; (e) none of these
7. An efficient algorithm for the topological sort of a dag is an instance of (a) brute force; (b) divide and conquer; (c) greedy algorithms; (d) dynamic programming; (e) probabilistic algorithms
8. Breadth-first search is \( O(____) \) (a) 1; (b) \( \lg n \); (c) \( n \); (d) \( n^2 \); (e) \( 2^n \)
9. Depth-first search is \( O(____) \) (a) 1; (b) \( \lg n \); (c) \( n \); (d) \( n^2 \); (e) \( 2^n \)
10. Topological sort is a(n) ____ problem (a) matrix; (b) optimization; (c) state-space search; (d) behavior-of-program; (e) interactive computation
11. The number of connected components of a bipartite graph \( G = (V,E) \) is (a) \( |V| \); (b) \( |V|/2 \); (c) \( |E| \); (d) \( |V|/2 \); (e) \( 2|E| \)
12. An ordering of courses that follows prerequisite requirements is a (a) search; (b) traversal; (c) array sort; (d) topological sort; (e) spanning tree
13. A prerequisite diagram for an academic major is a (a) heap; (b) array; (c) tree; (d) dag; (e) cycle
Terminology for topic 4 (Divide and conquer)

<table>
<thead>
<tr>
<th>binary search</th>
<th>dag</th>
<th>directed acyclic graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary-tree traversal</td>
<td>decrease by constant factor</td>
<td>divide and conquer</td>
</tr>
<tr>
<td>breadth-first search</td>
<td>depth-first search</td>
<td>extract-min</td>
</tr>
<tr>
<td>BST search</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Topic objective:
Explain the divide-and-conquer approach and apply it to a variety of algorithm-design problems, giving the appropriate time analysis for each.

4.1 Relate logarithmic time to very fast algorithms*
1. What running time do algorithms that reduce the size of the remaining data to be processed by half, at each step, have, and why?
2. Explain why decrease-by-constant-factor is associated with \( \lg n \) as a factor in running-time expressions.
3. In divide and conquer, what is divided?
4. What is it about divide-and-conquer that sometimes offers fast running time?
5. What is worst-case time for binary search, and why?
6. Give the complexity of the following algorithms and justify your answer: (a) Heapify (b) Heap-insert; (c) Heap-sort
7. Compare BST search and binary search, explaining similarities or differences in running times.
8. What are the running times of heapify, extract-min, and insert for heaps? Relate this to the data structure used.
9. How does the BST data structure support a divide-and-conquer algorithm design strategy?
10. Describe the merge sort, including complexity analysis.
11. Compare the worst-case and average-case performances of BST insertion and binary search. Explain.
12. Explain:
\[
T_{\text{Quick}}(n) = \begin{cases} 
1 & \text{if } n \leq 1 \\
O(n) + 2T_{\text{Quick}}(\lceil (n - 1) / 2 \rceil) & \text{otherwise}
\end{cases}
\]

4.2a Write a divide-and-conquer algorithm*
Write Java code or pseudocode for these divide-and-conquer algorithms:
1. Given Boolean array \( A \), arranged so that all 1’s are after any of the zeroes, return the number of 0’s.
2. Suppose you know that for array \( A \), there is some \( k \) s.t. \( A[1..k-1] \) is ascending, and \( A[k..|A|] \) is constant. Write an algorithm to find \( k \) in \( \lg n \) time.
3. Quicksort
4. BST insertion
5. Extract-min for heaps
6. An algorithm that determines whether a given binary tree has the BST property.
7. An algorithm that determines whether a given binary tree has the heap property.

4.2b Write a recurrence for a divide-and-conquer algorithm*
(1-7) Write recurrences for the algorithms that are your solutions in objective 4.2a.
(8-10) Write recurrences for the following:
8. Search \( (A, \text{first}, \text{last}, \text{key}) \)
   while first \( \leq \) last do
   \[
   \text{middle} \leftarrow (\text{first} + \text{last}) / 2 \\
   \text{if key} < A[\text{middle}] \\
   \text{last} \leftarrow \text{middle} - 1 \\
   \text{else if key} > A[\text{middle}] \\
   \text{first} \leftarrow \text{middle} + 1 \\
   \text{return true}
   \]
   return false
9. BST-search \( (\text{root}, \text{key}) \)
   If root is null, return false
   If root’s data matches key return true
   otherwise
   if root’s data \( > \) key
   return BST-search (left (root), key)
   otherwise
   return BST-search (right (root), key)
10. Heapify \( (i) \)
   \[
   \text{L} \leftarrow \text{Left}(i) \\
   \text{R} \leftarrow \text{Right}(i) \\
   \text{if L} \leq \text{Heap-size}(A) \text{ and A[L] < A[i]} \\
   \text{smallest} \leftarrow L \\
   \text{else}
   \text{smallest} \leftarrow i \\
   \text{if R} \leq \text{Heap-size}(A) \text{ and A[R] < A[smallest]} \\
   \text{smallest} \leftarrow R \\
   \text{if smallest \( \neq \) i}
   \]
   exchange \( A[i] \) with \( A[\text{smallest}] 
   \text{Heapify}(A, \text{smallest})
\]

Questions to assess objectives for topic 4

8. An economic recovery is an upswing in gross domestic product from a low point to a high point. Let an array store \( n \) monthly GDP figures. Describe a \( \Theta(\lg n) \) algorithm to detect the start and end of a recovery, of one has occurred. Assume that all series are of the form of a U-shaped, bell-shaped, down-up-down, or up-down-up curve.
9. Describe an implementation of a 3-heap, which uses a complete tree where each parent has not more than three children, in contrast to the heap we have described, which is based on a binary tree. Discuss complexity of the 3-heap.
10. Choose from Levitin, p. 128, Ex. 1 (array max); p. 139, Ex. 6 (binary search for elements in a range); p. 143, Ex. 2 (leaves in a binary tree), p. 143, Ex. 8 (internal path length in binary tree); p. 154, Ex. 3: (closest pair); p. 170-171, Ex. 1 (DFS)
4.2c Write and solve a time recurrence for a divide-and-conquer algorithm*

Write and solve a recurrence for the time function of an algorithm whose recurrence is

\((1-10) \ldots \text{a solution to objective 4.2b.}\)

\((11-13) \text{as follows:}\)

11. \(OS(k, A) =\)
    \[
    \begin{cases}
    A[pivloc(A)] & \text{if } k = \text{pivloc}(A) \\
    OS(\text{pivloc}(A) - k, A[1..\text{pivloc}(A)-1]) & \text{if } k < \text{pivloc}(A) \\
    OS(k - \text{pivloc}(A), A[\text{pivloc}(A)+1, |A|]) & \text{otherwise}
    \end{cases}
    \]

12. \(f(a, b) =\)
    \[
    \begin{cases}
    0 & \text{if } a = 0 \\
    1 + f(\lfloor a / 2 \rfloor, b) & \text{otherwise}
    \end{cases}
    \]

13. \(f(n) =\)
    \[
    \begin{cases}
    0 & \text{if } n = 0 \\
    1 + f(\lfloor n / 2 \rfloor) & \text{otherwise}
    \end{cases}
    \]

4.3 Explain or use traversal methods for trees

1. List the order of visiting vertices for an inorder and a postorder traversal of the tree below. Explain running time and explain how this is divide and conquer.

2. Explain inorder (preorder, postorder) tree traversal and give the design approach and running time.
3. Name and describe two tree-traversal methods.
4. Contrast the preorder, inorder, and postorder tree traversals. Explain their running times.
5. Give the nodes visited in the following tree, in the order visited, in a preorder (postorder, inorder) traversal. Give the algorithm’s complexity.

6. Give the nodes visited in the following tree, in the order visited, in a postorder (preorder, inorder) traversal. Give the algorithm’s complexity.

4.4a Explain breadth-first and depth-first search

1. Distinguish the breadth- and depth-first searches. What do they search in and what do they search for?
2. What structure do the depth-first and breadth-first search algorithms search? In what different ways do use the divide-and-conquer concept? What are their complexities?
5.0a Recall basic greedy concepts* M

1. When the next step is easy to choose

1. Greedy algorithms chiefly solve ______ problems
   (a) constraint; (b) vector; (c) sorting; (d) intractable;
   (e) optimization
2. Optimization problems are often solved efficiently by
   expanding a partial solution until the problem is solved, using
   ____ algorithms (a) divide-and-conquer; (b) brute-force;
   (c) greedy; (d) dynamic-programming; (e) probabilistic
3. A simple approach that is efficient, but does not work for all
   problems, is (a) divide-and-conquer; (b) brute-force;
   (c) greedy; (d) dynamic-programming; (e) probabilistic
4. Many optimization problems have efficient ____ solutions
   (a) divide-and-conquer; (b) brute-force; (c) greedy;
   (d) dynamic-programming; (e) probabilistic
5. Generating each element of a problem domain, and selecting
   the best-valued element that satisfies a certain constraint, is
   (a) divide and conquer; (b) exhaustive search; (c) the greedy
   approach; (d) dynamic programming; (e) a probabilistic
   approach
6. The approach to algorithm design for optimization problems
   that makes direct use of the fact that the most apparent
   next component of a solution is part of the optimal solution is
   (a) divide and conquer; (b) greedy; (c) brute force;
   (d) dynamic programming; (e) probabilistic
7. The best-known solution to the coin-changing problem is of
   the algorithm type (a) divide and conquer; (b) greedy;
   (c) brute force; (d) dynamic programming; (e) probabilistic
8. Huffman’s algorithm is (a) divide and conquer; (b) greedy;
   (c) brute force; (d) dynamic programming; (e) probabilistic
9. An algorithm to find an efficient compression encoding tree
   is (a) Dijkstra’s; (b) Prim’s; (c) Kruskal’s; (d) Huffman’s;
   (e) none of these
10. Which is an easy approach to designing a solution to the
    knapsack problem? (a) divide and conquer; (b) exhaustive
    search; (c) the greedy approach; (d) dynamic programming;
    (e) a probabilistic approach
11. Sorting an array of items by value-to-weight ratio in order to
    pack them in a maximum-valued knapsack enables a ______
    algorithm (a) brute-force; (b) divide-and-conquer; (c) greedy;
    (d) dynamic-programming; (e) probabilistic
12. Hill climbing is a(n) ____ approach (a) divide-and-conquer;
    (b) brute-force; (c) greedy; (d) dynamic-programming;
    (e) probabilistic
13. The optimal substructure property asserts that (a) if S solves
    an optimization problem, then any component of S is an
    optimal solution to a subproblem; (b) if some component of S
    is an optimal solution to a subproblem, then S solves the
    larger problem; (c) all parts of a problem have suboptimal
    solutions; (d) the sum of the parts is greater than the whole;
    (e) none of these

14. Greedy algorithms make use of (a) a straightforward solution
    to a problem by examining all possible solutions; (b) the fact
    that an optimal solution can be constructed by adding the
    cheapest next component, one at a time; (c) the fact that data
    is arranged so that at each step, half the remaining input data
    can disposed of; (d) the fact that effort can be saved by saving
    the results of previous effort in a table; (e) none of these
15. Greedy solutions exist for (a) no problems;
    (b) no optimization problems; (c) some problems; (d) all
    problems; (e) none of these
16. Greedy algorithms tend to be (a) O(lg n); (b) O(n); (c) O(n^2);
    (d) O(2^n); (e) none of these
17. In problems that feature local optima that are not global
    optima, the ______ algorithm approach is not
    effective (a) brute-force; (b) divide-and-conquer; (c) greedy;
    (d) dynamic-programming; (e) probabilistic
18. State-space search seeks (a) a local optimum; (b) a global
    optimum; (c) all values that satisfy a constraint; (d) the
    solution to a recurrence; (e) none of these
19. Local optima are (a) candidate solutions sought in state-space
    search; (b) the same as global optima; (c) solutions to be
    avoided; (d) harder to find than global optima; (e) none of
    these
20. All problems with greedy solutions have
    (a) recursive structures; (b) many local optima;
    (c) the optimal-substructure property; (d) tree structures;
    (e) constraint satisfaction as the goal
21. Making change with the denominations (1, 5, 10, 25) takes
    how many steps? (a) 1; (b) 3; (c) 6; (d) O(n); (e) O(n^2)

2. Greedy algorithms on graphs

1. Sorting a list of edges in a weighted graph enables use of a
   ______ algorithm (a) brute force; (b) divide and conquer;
   (c) greedy; (d) dynamic-programming; (e) probabilistic
2. Building a spanning tree by finding at each step the minimal
   edge adjacent to a partial subtree is an instance of ______
   algorithms (a) brute-force; (b) divide and conquer; (c) greedy;
   (d) dynamic-programming; (e) probabilistic
3. A structure that is connected and contains all the vertices in a
   weighted graph is (a) a coloring; (b) a path;
   (c) a spanning tree; (d) a single-source shortest path;
   (e) a depth-first traversal
4. A minimal spanning tree can be found by (a) subtracting
   edges greedily; (b) adding edges greedily; (c) seeking the
   shortest path; (d) recursive traversal; (e) none of these
5. The single-source shortest path problem has an efficient
   ______ solution of the type (a) brute force; (b) greedy;
   (c) depth-first search; (d) divide-and-conquer; (e) none of
   these
6. The observation that the subpaths of a shortest subpath are
   minimal is an instance of which property?
   (a) excluded middle; (b) reduction; (c) optimal substructure;
   (d) rationality; (e) order statistic
7. The Dijkstra algorithm finds (a) minimal spanning tree;
   (b) single-source shortest path; (c) a traversal;
   (d) maximal path; (e) a compression encoding tree
8. Labeling vertices of a graph so that no two adjacent vertices have the same label is (a) search; (b) spanning tree; (c) shortest path; (d) graph coloring; (e) bipartite matching
9. Graph coloring is (a) finding shortest path; (b) ensuring that no two adjacent vertices have the same color; (c) ensuring that no two edges have the same color; (d) bipartite matching; (e) a graphics programming task
10. A map of a geographic region may be represented as a (a) string; (b) number; (c) vector; (d) tree; (e) graph

11. The fastest known solution to the graph-coloring problem is (a) brute-force; (b) divide-and-conquer; (c) greedy; (d) exhaustive search; (e) approximate
12. There is a greedy algorithm that solves which problem efficiently? (a) sorting; (b) minimal spanning tree; (c) SAT; (d) traveling salesperson; (e) independent set
13. All the subpaths of a shortest path in a graph are (a) of length 1; (b) maximal; (c) minimal; (d) cyclic; (e) wandering
14. The stable-marriage problem is about (a) paths; (b) cycles; (c) matchings; (d) trees; (e) sorting

**5.0b Recall advanced greedy concepts**

3. Space/time tradeoffs and dynamic programming

1. Hashing solutions are associated with (a) space-time tradeoffs; (b) brute force; (c) intractability; (d) probabilistic algorithms; (e) approximation
2. The Boyer-Moore algorithm’s use of a table of shift values makes use of (a) divide and conquer; (b) the greedy approach; (c) brute force; (d) dynamic programming; (e) a probabilistic algorithm
3. B trees with m-ary nodes make use of (a) divide and conquer; (b) the greedy approach; (c) brute force; (d) space-time tradeoffs; (e) a probabilistic algorithm
4. Dynamic programming uses (a) the optimal-substructure property; (b) a divide-and-conquer approach; (c) hill climbing; (d) tables; (e) none of these
5. Space-time tradeoffs are associated with (a) brute force; (b) divide and conquer; (c) greediness; (d) dynamic programming; (e) probabilistic methods
6. Hashing raises the issue of (a) brute force; (b) divide and conquer; (c) greediness; (d) space-time tradeoffs; (e) approximation
7. B trees make use of (a) brute force; (b) divide and conquer; (c) greediness; (d) space-time tradeoffs; (e) approximation
8. The approach to algorithm design that reuses part of the solution search by storing values in memory is (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic
9. Dynamic programming solutions are associated with (a) space-time tradeoffs; (b) brute force; (c) intractability; (d) probabilistic algorithms; (e) approximation
10. Warshall’s algorithm for finding a reachability matrix is associated with (a) dynamic programming; (b) brute force; (c) intractability; (d) probabilistic algorithms; (e) approximation

11. An algorithm that generates a table of estimates and refines it progressively is associated with (a) dynamic programming; (b) brute force; (c) intractability; (d) probabilistic algorithms; (e) approximation
12. An efficient Fibonacci algorithm uses (a) the optimal-substructure property; (b) a divide-and-conquer approach; (c) hill climbing; (d) a table; (e) none of these
13. Computing the Fibonacci function has an efficient ______ algorithm (a) brute-force; (b) divide-and-conquer; (c) greedy; (d) dynamic-programming; (e) probabilistic
14. A solution to the knapsack problem that uses a table to store evolving estimates of solution values uses (a) the optimal-substructure property; (b) a divide-and-conquer approach; (c) hill climbing; (d) dynamic programming; (e) none of these
15. Dynamic-programming algorithms make use of (a) a straightforward solution to a problem by examining all possible solutions; (b) the fact that an optimal solution can be constructed by adding the cheapest next component, one at a time; (c) the fact that data is arranged so that at each step, half the remaining input data can disposed of; (d) the fact that effort can be saved by saving the results of previous effort in a table; (e) none of these

4. Transform and conquer

1. The AVL tree is an instance of (a) dynamic programming; (b) brute force; (c) intractability; (d) probabilistic algorithms; (e) transform-and-conquer
2. Instance simplifications, representation change, and problem reduction are used in ______ algorithms (a) divide and conquer; (b) the greedy approach; (c) transform and conquer; (d) dynamic programming; (e) a probabilistic algorithm
3. Verifying uniqueness of array elements by presorting is a case of (a) divide and conquer; (b) the greedy approach; (c) transform and conquer; (d) dynamic programming; (e) a probabilistic algorithm
4. Finding the mode (most common element) of an array by presorting is a case of (a) divide and conquer; (b) the greedy approach; (c) transform and conquer; (d) dynamic programming; (e) a probabilistic algorithm
5. BST balancing is a case of (a) divide and conquer; (b) the greedy approach; (c) transform and conquer; (d) dynamic programming; (e) a probabilistic algorithm
6. AVL trees are a case of (a) divide and conquer; (b) the greedy approach; (c) transform and conquer; (d) dynamic programming; (e) a probabilistic algorithm
7. A kind of transformation used in transform-and-conquer (a) adds complexity; (b) adds running time; (c) changes representation; (d) changes syntax; (e) changes semantics
8. Finding the mode of an array is facilitated by (a) searching; (b) sorting; (c) merging; (d) partitioning; (e) adding
9. Finding the median of an array is facilitated by (a) searching; (b) sorting; (c) merging; (d) partitioning; (e) adding
10. Transform-and-conquer always uses (a) sorting; (b) searching; (c) reducibility; (d) optimal substructures; (e) monotonicity
### Terminology for topic 5 (greedy algorithms)

<table>
<thead>
<tr>
<th>acyclic graph</th>
<th>adjacency matrix</th>
<th>AVL tree</th>
<th>binomial coefficient</th>
<th>BST balancing</th>
<th>B-trees</th>
<th>Build-Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dijkstra’s algorithm</td>
<td>dynamic programming</td>
<td>Kruskal’s algorithm</td>
<td>Extract-min</td>
<td>Fibonacci</td>
<td>greedy algorithm</td>
<td>hash function</td>
</tr>
<tr>
<td>hashing</td>
<td>least common multiple</td>
<td>linear probe</td>
<td>global optimum</td>
<td>local optima</td>
<td>minimal spanning tree</td>
<td>minimum heap</td>
</tr>
<tr>
<td>minimum heap</td>
<td>optimal BST</td>
<td>optimal-substructure property</td>
<td>optimalization problem</td>
<td>Prim’s algorithm</td>
<td>reachability matrix</td>
<td>sequence matching</td>
</tr>
<tr>
<td>reachability matrix</td>
<td>single-source</td>
<td>shortest path</td>
<td>time/space tradeoff</td>
<td>transform and conquer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Questions to assess objectives for topic 5 (Greedy algorithms)

**5.1a Explain greedy algorithm design**

1. Describe an algorithm that finds the single-source shortest path in a graph. Refer to the design approach used.
2. Describe two algorithms find minimal spanning trees in graphs. Refer to the design approach used.
3. Give a lower time bound for the minimal spanning tree problem and name an algorithm approach that achieves this time.
4. What does Huffman’s algorithm find, and how?
5. Explain the greedy approach to algorithm design, giving an example related to coins, graphs, or knapsacks.
6. Describe an efficient solution to any of the following problems, name the design method, and give a time analysis: graph coloring, knapsack packing, change making.

**5.1b Explain the optimal-substructure property**

1. What is the optimal-substructure property, and what kind of algorithm does it enable?
2. Defend or refute, defining the relevant terms as part of your answer: The optimal-substructure property asserts that every problem has some greedy algorithm that solves it.
3. What is an optimal substructure, and what does it have to do with algorithms?
4. Explain the relation among the optimal substructure property, global maxima, and local maxima.
5. Show by counter examples that neither of the following greedy solutions to the knapsack problem yield optimal results: (a) choose items in ascending order of weight; (b) choose items in descending order of weight.
6. Show that the optimal substructure property applies to searches for minimal paths in weighted graphs.

**5.2a Describe a greedy algorithm for graphs**

1. Describe an algorithm to find the shortest path from a certain vertex in a graph to a certain other vertex. Name the design approach used.
2. Show that the minimal spanning tree (Prim, Kruskal) is not necessarily the same as the tree of single-source shortest paths (Dijkstra).
3. Briefly describe, and analyze the running time:
   a. Prim’s algorithm
   b. Dijkstra’s algorithm
   c. Kruskal’s algorithm
   d. A minimal spanning tree algorithm
   a. A single-source shortest path algorithm
   b. A fast solution to the continuous-knapsack problem

**5.2b Apply a greedy algorithm**

Use the Dijkstra algorithm to find the tree of shortest paths for the graphs below.

2. Washington 4. Chicago

**5.8 For the graph below, starting at**

1. a 2. b 3. c
4. d 5. e

(7-9) How does the optimal substructure property apply to:

7. making change
8. climbing some hills
9. continuous knapsack problem, but not discrete version
5.3 Explain an instance of dynamic programming

1. What is dynamic programming about? Describe an algorithm of this kind.
2. Consider the following recurrence:
   
   $\text{Binomial}(n, k) = \begin{cases} 
   1 & \text{if}\ n = k \text{ or } k = 0 \\
   \text{Binomial}(i - 1, j - 1) + \text{Binomial}(i - 1, j) & \text{otherwise}
   \end{cases}

   (a) Give a time analysis of the algorithm induced by the recurrence, showing your work.
   (b) Write a dynamic-programming algorithm that computes the defined function and is much more efficient than the algorithm induced by the recurrence; give a time analysis.
3. Write a recurrence that defines the Fibonacci function in the standard way and analyze running time. Write a faster, dynamic-programming algorithm and analyze.
4. Analyze the following, comparing it to the standard Fibonacci algorithm:
   
   $Fib(x)
   
   F[0] \leftarrow 1, F[1] \leftarrow 1
   
   \text{For } i \leftarrow 2 \text{ to } x \text{ do}
   
   F[i] \leftarrow F[i-1] + F[i-2]
   
   \text{Return } F[x]

   What is the design approach, and what resource trade-offs are involved?
5. Analyze:
   
   $\text{Closure}(M[n, n])
   
   \text{for } i \leftarrow 1 \text{ to } n \text{ do}
   
   \text{for } j \leftarrow 1 \text{ to } n \text{ do}
   
   \text{for } k \leftarrow 1 \text{ to } n \text{ do}
   
   \text{if } M[i, j] \land M[j, k]
   
   M[i, k] \leftarrow \text{true}
   
   \text{Return } M

   How does the hashing process transform the problem of search, and how does it affect search running time?

5.4 Explain an instance of transform-and-conquer

1. Describe an algorithm that finds the mode (most frequent element) of an array by sorting it first. What design method is used? What is its complexity?
2. Describe an algorithm that finds the median (middle element) of an array by processing it first. What design method is used? What is the algorithm’s complexity and why?

3. Compare the time complexities of double-recursive Fibonacci and Fibonacci with dynamic programming.
4. Describe the resource that defines the main overhead of using the dynamic-programming method, with an example.
5. Compare the time complexities of the search algorithms for hash tables stored with open addressing (linear probe algorithm) and buckets (lists) and explain. If load factor is the ratio of occupancy to capacity in a hash table, state the range of possible load-factor values for each implementation and explain.
6. Describe the binomial-coefficient problem and a dynamic-programming solution.
7. What is the transitive closure of the adjacency relation? Describe an algorithm to find it.
8. How does Warshall’s algorithm compute the reachability matrix of a graph?
9. Compare how Dijkstra’s and Floyd’s algorithms solve the shortest-path problem. Should these two be categorized both as greedy and as dynamic-programming algorithms?
10. See slide on the knapsack problem algorithm. If the definition of the value of $V[i,j]$ were expressed as a recurrence, would the resulting algorithm’s time efficiency be different from the complexity of the algorithm shown?
11. What are the space and time considerations raised by B trees, and what design method was used to develop them?
12. Describe a transform-and-conquer algorithm to find the mode (most frequent element) of an array. Analyze.
13. What are AVL trees used for?
14. Write an algorithm to solve sorting-by-counting problem (see slide).
15. Code and analyze time complexity of Boyer-Moore search.
16. Write an algorithm to decide path of descent of a B tree.
17. Find complexities of algorithms that compute binomial coefficient, $C(n, k)$, that use recursion (brute force) and dynamic programming.
18. Levitin, p. 271, Ex. 7; p. 276, Ex. 3; p. 283, Ex. 4(a); p. 292, Ex. 1 (See D. Keil C++ code, available in course docs); p. 303, Ex. 2(a); p. 350, Ex. 4(a); p. 362, Ex. 4(a); p. 369, Ex. 1(a); p. 375, Ex. 2; p. 375, Ex. 5(a); p. 201, Ex. 2(a) (closest pair of numbers in array); p. 229, Ex. 2 (heap recognition)

Coding problems
Write a program to solve any of the problems discussed above. Give the number of test items that would be required to assure correctness. Obtain test results to describe its expected running time.
Multiple-choice questions on Topic 6 (Intractability)

6.0a Recall basic tractability concepts* $^M$

1. **Undecidability and intractability**

1. Uncomputable functions correspond to problems that are called (a) undecidable; (b) intractable; (c) P-time; (d) optimization; (e) none of these
2. The Halting Problem is (a) undecidable; (b) intractable; (c) NP-complete; (d) optimization; (e) none of these
3. The information-theoretic lower bound for a problem is given by the ____ height of the decision tree for the best algorithm that solves the problem (a) maximum; (b) minimum; (c) average; (d) absolute; (e) relative
4. The problem of sorting by comparisons is (a) $O(n)$; (b) $O(\log n)$; (c) $O(n^2)$; (d) $\Omega(n \log n)$; (e) $\Omega(n^3)$
5. Exponential time is closely associated with (a) tractability; (b) combinatorial explosion; (c) constraint problems; (d) sorting problem; (e) interaction
6. Intractable problems (a) are undecidable; (b) lack acceptable approximate versions; (c) are decidable but take an unacceptably long time; (d) lack solutions; (e) none of these

7. $\emptyset$ is used in the course material to denote (a) a problem; (b) an algorithm; (c) the function computed by program $P$; (d) the time function of program $P$; (e) none of these

8. \( f(x) \downarrow \) means (a) $f(x)$ is descending as $x$ rises; (b) $f(x)$ is undefined for parameter $x$; (c) $f$ is defined for parameter $x$; (d) $f$ is undefined for parameter $x$; (e) none of these
9. \( f(x) \uparrow \) means (a) $f(x)$ is descending as $x$ rises; (b) $f(x)$ is undefined for parameter $x$; (c) $f$ is defined for parameter $x$; (d) $f$ is undefined for parameter $x$; (e) none of these

10. A lower bound may give the ____ of a problem solution (a) maximum running time of; (b) average running time of; (c) minimum running time of; (d) minimum value output by; (e) shortest version of
11. The Hamiltonian-cycle problem is considered (a) intractable; (b) decidable; (c) tractable; (d) polynomial-time; (e) polymorphic
12. The problem of evaluating a formula in propositional logic, given an interpretation, is (a) intractable; (b) undecidable; (c) tractable; (d) $\Omega(2^n)$; (e) polymorphic
13. Deciding whether a formula in propositional logic is satisfiable is considered (a) intractable; (b) undecidable; (c) tractable; (d) decidable; (e) polymorphic
14. SAT is the problem of deciding whether a formula in propositional logic (a) holds; (b) has a set of variable assignments that make it true; (c) is not a contradiction; (d) is syntactically correct; (e) is probably true
15. The set of formulas in propositional logic that can evaluate to true values under some set of variable assignments is (a) SAT; (b) finite; (c) undecidable; (d) decidable in $O(n)$ time; (e) none of these
16. The set-partition problem is (a) undecidable; (b) decidable in $O(n^2)$ time; (c) a vector problem; (d) a tree problem; (e) intractable
17. The Traveling Salesperson problem is (a) undecidable; (b) decidable in $O(n^2)$ time; (c) a vector problem; (d) a tree problem; (e) intractable
18. The independent-set problem is (a) undecidable; (b) decidable in $O(n^2)$ time; (c) a vector problem; (d) a tree problem; (e) intractable
19. The 3SAT problem is (a) undecidable; (b) decidable in $O(n^2)$ time; (c) a vector problem; (d) a tree problem; (e) intractable

2. **Complexity classes of problems**

1. $\text{TIME}(f(n))$ is a set of (a) functions on natural numbers; (b) time functions; (c) problems; (d) algorithms; (e) times of day
2. $\text{DTIME}$ is a complexity class of (a) algorithms; (b) data structures; (c) problems; (d) theorems; (e) none of these
3. $\text{EXPTIME} = \text{DTIME}(\_\_\_\_)$ (a) $1$; (b) $n$; (c) $\log n$; (d) $n^2$; (e) $2^n$
4. *Tractable* problems are associated with (a) polynomial time; (b) $O(n)$; (c) $O(2^n)$; (d) $O(\log n)$; (e) none of these
5. $P$ is the set of (a) algorithms that execute in $O(n)$ time; (b) problems decidable in $O(n^2)$ time for some constant $k$; (c) problems not decidable in $O(n^2)$ time; (d) intractable problems; (e) exponential-time problems
6. If checking a given candidate solution to a problem is tractable, then the problem is in (a) $P$; (b) $NP$; (c) $NPC$; (d) $EXP$; (e) $SPACE$
7. $NP$ is the set of (a) algorithms that execute in $O(n^2)$ time for some constant $k$; (b) problems decidable in $O(n^2)$ time; (c) problems for which possible solutions may be checked in $O(n^k)$ time; (d) intractable problems; (e) exponential-time problems
8. Problems for which no polynomial-time solutions are known are called (a) undecidable; (b) intractable; (c) $NP$; (d) optimization; (e) none of these
9. $NPC$ is the set of all (a) problems that execute in $O(2^n)$ time; (b) problems decidable in $O(n^2)$ time for some constant $k$; (c) problems for which possible solutions may be checked in $O(n^k)$ time; (d) intractable problems; (e) exponential-time problems
10. Problems to which $\text{SAT}$ or similar problems are reducible are called (a) $P$; (b) $NP$; (c) $NP$-complete; (d) $NP$-hard; (e) undecidable
11. $NP$-complete problems are widely believed to have (a) polynomial-time solutions; (b) no polynomial-time solutions; (c) no exponential-time solutions; (d) no solutions checkable in polynomial time; (e) none of these
12. The set of intractable problems is associated with (a) $P$; (b) divide-and-conquer algorithms; (c) greedy algorithms; (d) $NP$; (e) $NPC$ and EXPTIME
13. Problem $A$ is reducible to problem $B$ if there is an algorithm that can (a) transform any instance of $B$ to an instance of $A$; (b) transform any instance of $A$ to an instance of $B$; (c) solve $A$; (d) solve $B$; (e) solve $A$ as fast as $B$
14. Finding the maximum value in an array is ______ the sorting problem (a) reducible to; (b) harder than; (c) a subproblem of; (d) equivalent to; (e) none of these
15. When we show that solving problem $A$ efficiently would give us an efficient solution to problem $B$, and problem $B$ is polynomial-time, we are using (a) construction; (b) contradiction; (c) induction; (d) reduction; (e) none of these
### 6.0b Recall advanced tractability concepts

#### 3. Approximate algorithms

1. An approach to algorithm design often used to address intractable problems is (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic algorithms
2. To find an exact solution to an intractable optimization problem may require (a) heuristics; (b) rules of thumb; (c) brute force; (d) approximation; (e) probabilistic algorithms
3. One way to find an adequate though inexact solution to an intractable optimization problem may be (a) brute force; (b) approximation; (c) divide and conquer; (d) greedy algorithm; (e) none of these
4. For ______ problems, sometimes global maxima/minima differ from local ones (a) optimization; (b) O(n); (c) BST search; (d) sorting; (e) none of these
5. Finding a feasible solution to an optimization problem under constraints, and refining it by successive steps, is (a) brute force; (b) dynamic programming; (c) iterative improvement; (d) intractable; (e) none of these
6. The iterative-improvement solution to the maximal-flow problem uses (a) brute force; (b) divide and conquer; (c) greediness; (d) dynamic programming; (e) approximation
7. Augmenting paths are used in (a) brute force; (b) divide and conquer; (c) iterative improvement; (d) dynamic programming; (e) probabilistic methods
8. The stable-marriage problem has a tractable solution by (a) brute force; (b) divide and conquer; (c) iterative improvement; (d) dynamic programming; (e) probabilistic methods
9. Iterative improvement starts with an _______ solution (a) greedy; (b) uncomputable; (c) intractable; (d) feasible; (e) unacceptable
10. Iterative improvement uses (a) brute force; (b) divide and conquer; (c) greediness; (d) dynamic programming; (e) approximation
11. Iterative improvement may fail to quickly find an optimal solution because (a) there is none; (b) “optimal” is undefinable; (c) global maxima may differ from local ones; (d) local maxima are wanted and only global ones are available; (e) “improvement” is undefinable
12. If a solution to the maximal-flow problem repeatedly finds a series of paths that increase the flow more and more, then it uses (a) transform and conquer; (b) greediness; (c) iterative improvement; (d) brute force; (e) divide and conquer
13. If a solution to the maximal-matching problem repeatedly finds paths that yield larger matches, then it uses (a) transform and conquer; (b) greediness; (c) iterative improvement; (d) brute force; (e) divide and conquer
14. If a solution to the stable-marriage problem involves a series of proposals to preferred partners and acceptances if proposal is by a more preferred proposer, then it uses (a) transform and conquer; (b) greediness; (c) iterative improvement; (d) brute force; (e) divide and conquer

### 4. State-space search

1. Generate-and-test methods explore (a) trees; (b) arrays; (c) graphs; (d) state spaces; (e) none of these
2. Iterated local search is (a) hill climbing; (b) O(lg n); (c) deterministic; (d) exact; (e) none of these
3. A state space is (a) part of RAM; (b) one set of variable assignments; (c) a set of possible arrangements of values; (d) a graph; (e) none of these
4. The knapsack problem searches (a) a tree; (b) a graph; (c) an array; (d) a state space; (e) none of these
5. Games and puzzles are simple examples of (a) embodied intelligence; (b) state-space search; (c) inference; (d) agent interaction; (e) adaptation
6. The minimax algorithm emerged from (a) recursive function theory; (b) calculus; (c) game theory; (d) symbolic logic; (e) none of these
7. The principle of rationality states that an agent will pursue (a) knowledge; (b) a goal; (c) another agent; (d) a minimal value; (e) none of these
8. *Rationality* in economic theory is (a) the use of predicate logic; (b) the use of rational numbers; (c) acting in one’s best interests using heuristics; (d) acting in one’s best interests using perfect information; (e) following instructions
9. *Hill climbing* is (a) a problem; (b) heuristic strategy; (c) best-first search; (d) expression of consciousness; (e) form of representation
10. A drawback of hill climbing is (a) very long running time; (b) undecidability; (c) tendency to become stuck at a local maxima; (d) the absence of goals; (e) none of these
11. A set of hard problems in interactive environments are (a) state-space search; (b) one-player game; (c) planning action sequences; (d) POMDPs; (e) none of these

### 5. Randomized and evolutionary algorithms

1. An adequate though inexact solution to an intractable optimization problem may be (a) brute force; (b) probabilistic; (c) divide and conquer; (d) O(2^n); (e) exponential time
2. Evolutionary computation uses the technique of maximizing (a) fitness; (b) reward; (c) performance; (d) quantity of output; (e) time
3. Evolutionary computation (a) is deterministic; (b) seeks optimal solutions; (c) was developed in the 19th century; (d) is probabilistic; (e) is O(lg n)
4. Evolutionary computation is modeled on (a) brute force; (b) divide and conquer; (c) greediness; (d) natural selection; (e) fractals
5. Evolutionary computation is (a) a brute-force method; (b) state-space search one state at a time; (c) path optimization; (d) population based; (e) DNA computing
6. Function optimization searches for (a) a function; (b) parameter values; (c) a return value; (d) an algorithm; (e) a time analysis
7. Genetic algorithms are (a) greedy; (b) brute-force; (c) a way to compute fitness; (d) a form of evolutionary computation; (e) used in the human genome project
8. Ant computing is (a) greedy; (b) brute-force; (c) a way to compute fitness; (d) a form of evolutionary computation; (e) used in the human genome project
Terminology for topic 6 (Intractability)

approximation algorithm \hspace{1cm} DTIME \hspace{1cm} generate-and-test algorithm \hspace{1cm} local maxima
backtracking \hspace{1cm} evolutionary computation \hspace{1cm} global maxima \hspace{1cm} NP-hard problem
bounded rationality \hspace{1cm} EXPTIME \hspace{1cm} halting problem \hspace{1cm} rationality
branch-and-bound \hspace{1cm} function \hspace{1cm} independent set \hspace{1cm} reducibility
complexity class of problems \hspace{1cm} optimization \hspace{1cm} intractable problem \hspace{1cm} space search
decision tree \hspace{1cm} game theory \hspace{1cm} iterative improvement \hspace{1cm} stable marriage

Questions to assess objectives for topic 6 (Intractability)

**Topic objective:**
Describe the notion of intractability mathematically, including complexity classes, and ways to overcome this obstacle.

6.1 Describe an intractable problem**
1. Distinguish satisfiability, validity, and evaluation of propositional-logic formulas. Are some harder than others?
2. Describe the correspondence between a decision problem and a language.

*Explain the following problems, describing their complexities.*
3. telling whether a formula in propositional logic is satisfiable
4. telling whether a formula in propositional logic is a contradiction
5. telling whether a formula in propositional logic is a tautology
6. finding a path through a graph that includes each vertex exactly once
7. telling whether a set of natural numbers can be partitioned in such a way that the sum of the elements in each partition is the same
8. the traveling-salesperson problem
9. the stable marriage problem
10. telling whether a set of vertices in a graph contains any pair that is adjacent
11. factoring a product of two n-digit primes
12. finding a maximum matching in a bipartite graph, and one solution.

6.2a Describe the main complexity classes for tractable and intractable problems*
1. Define the two main complexity classes that distinguish tractable and intractable problems. Describe associated complexity classes.
2. Describe some classes of intractable problems discussed in the course.
3. Define $P$ mathematically, describe it in plain English, and relate it to a set of problems.
4. Give an example of a decidable but intractable problem and an example of an undecidable set.
5. Explain and compare complexities of TSP and Hamiltonian path.
6. Explain the complexity of the problem of printing all permutations of a sequence, $A$.
7. Compare backtracking to depth-first search.

8. What is the (very short) name and a definition of the class of problems that are decidable in time that is a polynomial function of the input size? Give examples of problems in this class and problems that are not believed to be in it.
9. Explain the set-partition problem and describe its complexity.
10. What bound on what resource is given by the minimum height of a decision tree representing execution of an algorithm?
11. Explain the complexity of the problem of printing all subsets of a set, $A$.
12. What is wrong with the following? It is easy to solve the Halting Problem. Just compile the code in question and see if it halts. If it does, output “yes”, otherwise “no”.
13. What bound on what resource is given by the minimum height of a decision tree representing execution of an algorithm?
14. What common-sense evidence is there that the Hamiltonian Cycle problem is very hard?
15. What sorts of running times is intractability associated with?
16. Argue for or against: the satisfiability problem for formulas in propositional logic is intractable.
17. Distinguish the problems of validity and satisfiability of propositional-logic formulas, referring to problem specification and complexity.
18. Concerning the following formula in propositional logic $(p \lor q \lor r) \land (p \lor \neg q \lor r) \land (\neg p \lor \neg q \lor \neg r)$:
   a. State whether the formula is satisfiable, showing your work (you may write a truth table);
   b. State and explain what is the time necessary to answer the question for arbitrary formulas with $k$ variables.
19. What are the complexities of these problems w.r.t. formulas $\phi$ in propositional logic?
   a. $\phi$ is a tautology
   b. $\phi$ is satisfiable
   c. $\phi$ is a contradiction
   d. $\phi$ holds for a given set of variable assignments
20. Explain NP-completeness.
21. Write program code to generate a truth table that verifies the assertion that $p \lor q \lor r \lor s = (r \lor q \lor s) \land (r \lor \neg p \lor y) \land (s \lor \neg y \lor \neg x)$
   *(Note: This will help establish that any CNF formula can be rewritten as a 3-CNF one.)*
22. If there is an algorithm that can transform any instance of problem $A$ to an instance of problem $B$, then what can be said about the relationship of $A$ to $B$?
23. Briefly, what are the classes P, EXPTIME, NP, NP-Hard, and NPC? Give several containment relationships among them. Describe the connection between intractability and one of these classes.
24. Give reasons why it makes sense to associate P with tractability, and give reasons why this association has limited value.
25. What complexity class (P, NP, NPC, NP-hard, EXP) are the following problems in?
   a. Searching an array
   b. Tree traversal
   c. Hamiltonian path
   d. Knapsack
   e. Sorting
   f. Single-source shortest path

6.2b Use a reduction to explain why a problem is intractable
1. What is a way to show that computational problem A is at least as hard as problem B?
2. If the SAT problem is reducible to problem π, then what can be said about problem π? Explain.
3. If there is an algorithm that can transform any instance of problem A to an instance of problem B, then what can be said about the relationship of A to B? How is this notion used to prove the complexities of the same problems?
4. Suppose someone found an O(n^k) solution to the satisfiability problem.
   a. What implications would this have for other problems reducible to TSP in polynomial time, and why?
   b. What implications would it have for other problems to which TSP is reducible in polynomial time, and why?
5. Suppose someone found an O(n^k) solution to the Traveling Salesperson problem.
   a. What implications would this have for other problems reducible to TSP in polynomial time, and why?
   b. What implications would it have for other problems to which TSP is reducible in polynomial time, and why?
6. The SAT problem is reducible to that of 3-satisfiability, and 3SAT is reducible to the independent-set problem. The independent-set problem consists of telling whether a graph contains a set of n or more vertices, none of which have an edge in common. What can be said of the complexity of the independent-set problem?
7. Discuss the possible reducibility relationship of the following with the problem of finding a path through a graph that passes through k or fewer vertices. Explain a possible case.
   a. Single-source shortest path
   b. Search for a vertex in a graph
   c. Convex hull
   d. Minimal spanning tree
   e. SAT
   f. Sorting

6.3 Explain an approximation algorithm
1. When is the exhaustive search for a path through a graph likely to take a short amount of time? A huge amount of time? Suggest an alternative in the latter case.
2. Give two cases we have discussed where approximation algorithms are the only reasonable ways to solve a problem, and explain why that is true.
3. What is considered a reasonable alternative to the goal of optimality and what are ways to achieve it?

4. For what sorts of problems are approximation algorithms, randomization, and heuristics used?
5. Why is the algorithm shown for approximately solving the stable-marriage problem in O(n^2)?
6. What does searching state spaces have to do with solving problems?
7. When is the generate-and-test algorithm unlikely to find an optimal solution?
8. Why does hill climbing often find approximate rather than exact optima?
9. What is the complexity of minimax, relative to the number of moves ahead it looks in a game?
10. What principle is described by saying that an agent will choose an action if the agent knows that the solution will lead to a chosen goal?

6.4 Describe the notion of state-space search
1. Relate state-space search to hard computational problems.
2. Describe steps in a state-space search.
3. What are the states in a state-space search?
4. Describe the state-space search of a maze.
5. Describe a state-space search that may occur when a wholesale buyer chooses items and quantities to order for a retail outlet.

6.5 Explain a randomized approach to solving intractable problems
1. What sorts of problems does evolutionary computation address and how?
2. In your own words, relate fitness to function optimization.
3. What sort of algorithm uses Select, Alter, and Evaluate functions?
4. Describe a probabilistic method for computing the approximations of functions, inspired by a phenomenon found in nature.
5. How does evolutionary computation use the technique of randomization to address intractable problems?

Coding problems
Write a program to solve any of the problems from objective 6.1 above. Obtain test results to describe its expected running time.

Other problems
Levitin, p. 385, Ex. 3(a), 3(c) (Lower bound classes for maximum and subset problems); p. 393, Ex. 9(a)-(b) (Tournament tree, number of games and rounds); p. 402, Ex. 7(a) (State the decision version of knapsack problem); p. 423, Ex. 3(a) (Backtracking for n-queens problem; pseudocode plus statement of rules is OK); p. 432, Ex. 1 (Data structure to keep track of live nodes in best-first branch-and-bound algorithm)
Multiple-choice questions on Topic 7 (Concurrency)

7.0a Recall basic concurrency concepts*

1. Concurrent computing
   1. Parallel computing is a kind of (a) concurrency; (b) Von Neumann architecture; (c) serial paradigm; (d) single-core computing; (e) none of these
   2. Distributed computing is (a) single core; (b) concurrency; (c) distributed computing; (d) intractable; (e) rare
   3. A thread simulates (a) ownership of all resources; (b) ownership of a CPU; (c) ownership of data; (d) an array; (e) none of these
   4. Serial computing is the same as (a) concurrency; (b) parallelism; (c) distributed computing; (d) single-core; (e) none of these
   5. Parallel computing is (a) single core; (b) concurrency; (c) distributed computing; (d) intractable; (e) slow
   6. Process algebras assume communication by (a) packet; (b) indirect interaction; (c) bus; (d) message passing; (e) parameters
   7. PRAM is (a) a model of serial computing; (b) a kind of memory; (c) a model of parallel computing; (d) a greedy algorithm; (e) none of these
   8. PRAM assumes that memory is (a) all distributed; (b) shared; (c) infinite; (d) magnetic; (e) none of these
   9. Concurrent write is as opposed to ___ write (a) serial; (b) automatic; (c) shared; (d) exclusive; (e) none of these
   10. The standard model for parallel computing is (a) CRCW; (b) RAM; (c) PRAM; (d) Von Neumann; (e) none of these
   11. Under SIMD, all CPUs execute ___ instructions (a) the same; (b) different; (c) pipelined; (d) cached; (e) complex
   12. Under SIMD, all CPUs operate on ___ data (a) the same; (b) different; (c) pipelined; (d) cached; (e) disk

2. Parallel algorithms
   1. When memory is shared in parallel computing, a related issue may be (a) data uniformity; (b) cache coherency; (c) disk latency; (d) intractability; (e) none of these
   2. Amdahl’s Law states that maximum speedup (a) is O(1); (b) is O(lg n); (c) is limited by the fraction of code that is parallelizable; (d) is unlimited; (e) none of these
   3. What is the top parallel-computing speedup? (a) 2x; (b) n, for n processors; (c) n^2 for n processors; (d) limited by fraction of code that may be executed in parallel
   4. A parallel algorithm exists to add all the elements of an array in parallel time (a) Θ(1); (b) Θ(lg n); (c) Θ(n); (d) Θ(n^2); (e) none of these
   5. A parallel algorithm exists to assign the same value to all elements of an array in parallel time (a) Θ(1); (b) Θ(lg n); (c) Θ(n); (d) Θ(n^2); (e) none of these
   6. A parallel algorithm exists to search an array in parallel time (a) Θ(1); (b) Θ(lg n); (c) Θ(n); (d) Θ(n^2); (e) none of these
   7. A parallel algorithm exists to apply any associative operator to an array in parallel time (a) Θ(1); (b) Θ(lg n); (c) Θ(n); (d) Θ(n^2); (e) none of these
   8. Parallel prefix computation operates in parallel time (a) Θ(1); (b) Θ(lg n); (c) Θ(n); (d) Θ(n^2); (e) none of these

7.0b Recall advanced concurrency concepts*

3. Distributed algorithms
   1. A distributed algorithm runs on (a) a single processor; (b) a parallel-processing system; (c) multiple independent processors; (d) web pages; (e) none of these
   2. In distributed systems, as opposed to serial or parallel systems, a significant factor in determining scalability is (a) cache coherency; (b) recursion; (c) communication time; (d) brute force; (e) none of these
   3. Ring network broadcast is (a) a distributed algorithm; (b) parallel algorithm; (c) intractable problem; (d) multitasking primitive; (e) none of these
   4. Leader selection for a ring network is a (a) distributed algorithm; (b) parallel algorithm; (c) intractable problem; (d) multitasking primitive; (e) none of these
   5. An example of a distributed algorithm is (a) Quicksort; (b) parallel-prefix; (c) Java threading; (d) ring broadcast; (e) serial search
   6. An example of a distributed algorithm is (a) Quicksort; (b) parallel-prefix; (c) Java threading; (d) leader selection for ring; (e) serial search
   7. Scalability for concurrent systems is defined as (a) algorithms are O(n^2); (b) problems are tractable; (c) time is constant as problem size rises; (d) productivity is maintained as scale rises; (e) none of these
   8. Quality of service must be maintained for (a) scalability; (b) tractability; (c) decidability; (d) correctness; (e) O(n) time performance
   9. Scalability of distributed systems is heavily dependent on (a) parallel fraction; (b) serial fraction; (c) memory access time; (d) communication time; (e) none of these
   10. The part of an algorithm that can be distributed to different processor is the ___ fraction (a) efficient; (b) serial; (c) parallel; (d) scalable; (e) hard-working
   11. Parallel work is ___ parallel time (a) less than; (b) equal to; (c) somewhat greater than; (d) exponentially greater than; (e) none of these
   12. If speedup is linear, a parallel system and algorithm are said to be (a) serial; (b) distributed; (c) scalable; (d) efficient; (e) quadratic

9. In parallel computing, work is greater than (a) serial time; (b) parallel time; (c) amount of data; (d) best-case time; (e) none of these
10. Parallel speedup is the (a) parallel time; (b) parallel work; (c) increase in speed of processors; (d) serial time divided by parallel time; (e) none of these
11. Problems considered tractable in parallel computing are in class (a) P; (b) NP; (c) NC; (d) NPC; (e) none of these
12. NC problems are ___ on parallel systems (a) log-time; (b) constant-time; (c) linear-time; (d) polylog-time; (e) intractable
13. The serial fraction is the ___ part of an algorithm (a) efficient; (b) parallelizable; (c) unparallelizable; (d) intractable; (e) none of these
4. **Sequential interaction**

1. A feature of algorithmic computation is (a) alternation of input and output; (b) processing before input; (c) output before processing; (d) input, then processing, then output; (e) none of these

2. A feature of algorithmic computation is finite (a) input; (b) output; (c) processing; (d) input, output, and processing; (e) none of these

3. Algorithms (a) compute functions; (b) provide services; (c) accomplish missions in multi-agent systems; (d) may execute indefinitely; (e) none of these

4. **Reactive systems** (a) compute functions; (b) provide services; (c) accomplish multi-agent missions; (d) execute only finitely; (e) none of these

5. I/O in reactive systems is (a) static; (b) dynamically generated; (c) finite; (d) constrained; (e) none of these

6. Interaction is distinguished from algorithmic computation by the presence of (a) finite input; (b) persistent state; (c) input; (d) processing; (e) none of these

7. The environment is an ongoing active partner in ____ computation (a) algorithmic; (b) parallel; (c) distributed; (d) interactive; (e) none of these

8. A mutual causal effect between two agents occurs in all (a) interaction; (b) algorithms; (c) communication; (d) computing; (e) none of these

9. **Synchrony** entails (a) communication; (b) taking turns; (c) input; (d) autonomy; (e) none of these

10. **Stream I/O** characterizes (a) interaction; (b) algorithms; (c) functions; (d) O(1) processes; (e) none of these

11. Interaction may be sequential or (a) algorithmic; (b) O(n); (c) multi-stream; (d) data-driven; (e) none of these

12. **Persistent state** denotes (a) parameters; (b) loop invariants; (c) memory of past interactions; (d) algorithmic computation; (e) none of these

13. The form of input/output in interaction is (a) static; (b) finite; (c) streams; (d) strings; (e) none of these

14. A service is characteristic of (a) an algorithm; (b) an interactive process; (c) a multi-agent system; (d) a parallel system; (e) none of these

15. UML, as opposed to pseudocode, is required to specify (a) functional problems; (b) interaction; (c) algorithms; (d) algorithmic inputs; (e) none of these

16. **Mutual causality** characterizes (a) algorithms; (b) only directly-interacting entities; (c) all interaction; (d) functions; (e) none of these

17. In interactive environments, a agent requires (a) a reflex mapping; (b) an action sequence; (c) planning under uncertainty; (d) a policy; (e) none of these

5. **Multi-stream interaction**

1. A mission is characteristic of (a) an algorithm; (b) an interactive process; (c) a multi-agent system; (d) a parallel system; (e) none of these

2. Indirect interaction requires (a) mutual causality between entities that do not interact directly; (b) message passing; (c) synchrony; (d) static I/O; (e) none of these

3. The problem solved by a multi-agent system is called a(n) (a) algorithm; (b) function; (c) service; (d) mission; (e) process

4. Sequential-interactive agents offer a(n) (a) algorithm; (b) function; (c) service; (d) mission; (e) process

5. Self-organization is (a) algorithmic; (b) sequential-interactive; (c) decentralized; (d) centralized; (e) shared-memory

6. Stigmergy makes use of (a) the environment; (b) first-order logic; (c) adaptive learning; (d) Bayesian reasoning; (e) shared memory

7. Coordination via the environment is (a) minimax; (b) stigmergy; (c) centralized; (d) a heuristic; (e) none of these

8. Indirect interaction is in contrast to (a) deduction; (b) distributed AI; (c) stigmergy; (d) message passing; (e) none of these

9. Stigmergy uses (a) indirect interaction; (b) language; (c) reasoning; (d) a knowledge base; (e) none of these

10. Self-organization is observed in (a) English grammar; (b) garden design; (c) social insects; (d) proofs; (e) none of these

11. Indirect interaction features (a) anonymity; (b) synchronization; (c) use of predicate logic; (d) fuzzy reasoning; (e) none of these

12. Indirect interaction features (a) space decoupling; (b) synchronization; (c) use of predicate logic; (d) fuzzy reasoning; (e) none of these

13. Ant foraging by use of chemical trails is an example of (a) inference; (b) linguistic processing; (c) stigmergy; (d) evolution; (e) none of these

14. A policy is a mapping of (a) natural numbers to words; (b) income to premium; (c) states to actions; (d) percepts to outputs; (e) problems to algorithms

15. A policy is (a) a mapping from states to actions; (b) a planned action sequence; (c) an algorithm; (d) a mapping of percepts to actions; (e) none of these

16. The scalability of the _____ model is questioned by the course materials (a) algorithmic; (b) sequential-interactive; (c) message-passing; (d) distributed; (e) parallel

17. Time efficiency of systems designs can be expressed as (a) processor speed; (b) parallelism; (c) scalability; (d) intractability; (e) ease of use

18. Intractable problems are (a) solvable with efficient algorithms; (b) scalable; (c) unsolvable; (d) unsolvable; (e) none of these

19. Message passing models are considered (a) uncomputable; (b) intractable; (c) unsolvable; (d) scalable; (e) none of these

20. If time performance degrades quickly as the number of computing agents rises, the system is (a) worthy; (b) incorrect; (c) unsolvable; (d) synchronous; (e) efficient
**Terminology for topic 7 (concurrency)**

- Amdahl’s Law
- autonomous agent
- broadcast cache coherency circuit model communication time concurrency concurrent read/write data parallel direct/indirect interaction distributed algorithm

- distributed-memory architecture efficient parallel algorithm exclusive read/write interactive computation leader election load balancing message complexity message passing mission


- persistence of state policy search polylog(n) process quality of service scalability sequential interaction serial computing serial fraction service

- shared-memory architecture single instruction single data (SIMD) speedup synchronous interaction thread Von Neumann architecture work

**Questions to assess objectives for topic 7 (concurrency)**

**Topic objective:**
Describe design and performance issues in parallel, distributed, and interactive computation.

### 7.1 Describe two forms of concurrency*

1. Briefly state the common and distinguishing features of distributed computing, multi-tasking, and parallel processing.
2. Describe two forms of concurrency.
3. What is an ongoing series of responses to inputs, queries, or requests?
4. What sort of problem requires servicing multiple input streams, possibly under time constraints?
5. Describe parallel prefix computation
6. What are the two memory architectures for parallel computing?
7. Briefly state the common and distinguishing features of distributed computing, multi-tasking, and parallel processing.
8. Describe two models for concurrent computing.
9. What term that incorporates multi-tasking, multi-threading, networking computing, and parallel computing. Distinguish two of these

### 7.2a Design a parallel algorithm

**(1-7) Using parallel for, write a parallel algorithm to find, for an array of n integers:**

1. the sum
2. the maximum value
3. the OR, if the integers are all 0 or 1
4. the minimum value
5. the number of 0’s
6. whether the array is sorted
7. whether the array contains a given search key
8. Write a parallel algorithm to compute the AND of an array of bits in time \(O(\lg n)\).

### 7.2b Characterize the performance of a parallel algorithm*

Give time complexity for solutions to the problems in objective 7.2a.

1. Define the parallel fraction and related terms.
2. What limits exist on the possible speedup offered by parallel computing for a program?
3. Distinguish work from parallel time.

9. Explain why the following pseudocode would fail, and describe how to correct it:

   ```
   Sum (A) 
   y ← 0 
   forall processors PID ← 1 to |A| 
   y ← y + A[PID] 
   return y 
   ``

10. Explain why the following pseudocode would fail, and describe how to correct it:

    ```
    And (A) 
    y ← true 
    forall processors PID ← 1 to |A| 
    if ¬(y ∧ A[PID]) 
    y ← false 
    return y 
    ```

11. Explain why the following pseudocode would fail, and describe how to correct it:

    ```
    Max (A) 
    y ← A[1] 
    forall processors PID ← 2 to |A| 
    if A[PID] > y 
    y ← A[PID] 
    return y 
    ```

12. Explain why the following parallel pseudocode would fail, and describe how to correct it:

    ```
    Count-true (A) 
    count ← 0 
    forall processors PID ← 1 to |A| 
    if A[PID] = true 
    count ← count + 1 
    return count 
    ```
4. What factors other than those considered in serial algorithm analysis must be considered in performance analysis of parallel and distributed systems?

5. How does work relate to parallel time?

6. Discuss work, parallel time, speedup, and efficiency.

7. Give a definition of scalability in your own words.

8. What parallel algorithms apply associative operators efficiently, and in what parallel time?

9. Discuss the idea that many NPC problems, e.g., playing chess, are tractable for parallel systems because brute force algorithms on exponentially many CPUs will solve the problems in PTIME.

10. Argue for or against the possibility of super-linear speedup; i.e., parallel algorithms whose running times are less than \( O(t/n) \), where \( t \) is the running time of the best sequential algorithm to solve the problem and \( n \) is the number of processors in the parallel system.

11. What factors other than length of input affect the performance of parallel and distributed systems?

7.3 Describe a distributed algorithm

1. Describe an efficient algorithm to send the same message to all nodes of a distributed system.

2. (a) Describe a distributed algorithm to send the same message to all nodes of a distributed system organized in (ring, binary-tree) configuration.
   (b) Give its complexity and justify your answer.

3. Describe the ring broadcast algorithm

4. Describe the leader-election algorithm

7.4 Distinguish algorithms from interactive processes*

1. What measures of efficiency are applied in interactive computing?

2. What sort of computation is associated with computing functions, and what sort of computation is associated with providing services? Give another distinction.

3. Distinguish a service from an algorithmic problem.

4. What performance issues are associated with types of computation that provide services?

5. Discuss the time complexity of sequential interactive processes.

6. Describe what a mission is, for this course.

7.5 Describe features of multi-stream interaction

1. Distinguish two types of interaction by the number of active agents involved.

2. Distinguish two types of interaction by the use or non-use of the environment in interaction.

3. In multi-agent systems, what scalability issues are raised by the choice between message passing and shared access to storage?

4. Discuss the scalability of multi-agent systems.
Study questions on multiple topics

1. Describe several approaches to algorithm design, with examples, and with discussion of running times achievable by these approaches.
2. Discuss problems related to propositional or predicate logic, referred to in this course, and discuss uses for logic in verification and analysis of algorithms and interactive processes.
3. Describe the main notations for expressing the complexities of algorithms.
4. Describe how recurrences are used in algorithm analysis. Give examples of three time recurrences that yield three different complexity classes.
5. Name the approaches to algorithm design that are illustrated by:
   (a) Prim’s algorithm
   (b) the use of tables
   (c) selection sort
   (d) Quicksort
   (e) evolutionary computation
6. Why are trees important in this course?
7. Discuss exponential-time algorithms.
8. Discuss tractable and intractable problems, defining some complexity classes.
9. For a problem from problem set A below,
   1. Write an algorithm to solve the problem.
   2. Using a postcondition and a loop invariant, prove that your algorithm is correct.
   3. Use a recurrence to define the function computed by the algorithm.
   4. Write the corresponding time recurrence and solve it.

Problem set A
Given an array A of integers, return
1. true iff all elements are 1
2. true iff any element is 1
3. true iff the array is in ascending order
4. true iff any consecutive elements of A are the same
5. true iff any consecutive elements of A differ
6. the sum of the elements
7. the maximum element
10. For a problem from problem set B below,
   a. Write an algorithm to efficiently solve the problem.
   b. Write a recurrence that defines the function computed by the algorithm
   c. Write the corresponding time recurrence and solve it, for worst case and for average case.

Problem set B
1. for sorted array A, return true iff A contains key value x
2. given bit vector A, sorted so that all 1’s are before any of the zeroes, return the number of 1’s.
3. for binary search tree T, return true iff T contains key value x
4. suppose you know that for array A, there is some k s.t. A[1 .. k – 1] is ascending, and A[k .. |A|] is constant. Write an efficient algorithm to find k.
11. Compare some search algorithms discussed in this course, with respect to at least three different data structures, and their time complexities.
12. Compare the analysis of the running times of serial algorithms, the characterization of the complexity of algorithmic problems, and the analysis of parallel algorithms.
13. For topic __, see your group-work submission.
   (a) What were the main concepts presented in the course that this group work exercised or reinforced for you?
   (b) Discuss briefly any errors in your group work for this topic.
   (c) Critique the group-work submission that appears just after yours on the Discussion Board. (If yours is last, critique the first submission.)
14. Referring to chapter __ of Johnsonbaugh-Schaefer,
   (a) What are the main concepts presented?
   (b) What are the main concepts presented in the topic(s) of this course for which the chapter was assigned?
   (c) Compare and contrast the textbook presentation with what was presented in the classroom and the slides.
15. Relate the eleven topics discussed in this course. Focus on the way that some material occurred again and again in later topics. Contrast different approaches taken, problems addressed, and objects analyzed. Include discussion of ways to mathematically formalize some of the different notions addressed in the course. While covering all the main topics, you may emphasize ones that interested you the most.
16. How have your perspectives on the course material changed as a result of
   (a) group work,
   (b) comments on your work from other students,
   (c) comments from the instructor?
17. What ideas or passages in the textbook most engaged you or most changed the way you thought?