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 them can help guide the student's study.

Answers are supplied for some questions in certain versions of this file. Some answers may contain typographical errors. Students are asked to question all results that seem incorrect or are hard to understand. Grading of all quizzes will be according to the *correct* answer, not the answer that has been provided in some list of answers.

Contents

Intro

1. Classes of problems

2. Verification of algorithms

3. Recurrences; brute force

4. Divide and conquer

5. Greedy and other efficient optimization algorithms

6. Complexity; approximation

7. Parallel, distributed, interactive computing

Summary

1. (a) ; (b) ; (c) ; (d) ; (e) none of these
2. (a) ; (b) ; (c) ; (d) ; (e) none of these
3. (a) ; (b) ; (c) ; (d) ; (e) none of these
4. (a) ; (b) ; (c) ; (d) ; (e) none of these
5. (a) ; (b) ; (c) ; (d) ; (e) none of these
6. (a) ; (b) ; (c) ; (d) ; (e) none of these
7. (a) ; (b) ; (c) ; (d) ; (e) none of these
8. (a) ; (b) ; (c) ; (d) ; (e) none of these
9. (a) ; (b) ; (c) ; (d) ; (e) none of these
10. (a) ; (b) ; (c) ; (d) ; (e) none of these
11. (a) ; (b) ; (c) ; (d) ; (e) none of these
12. (a) ; (b) ; (c) ; (d) ; (e) none of these
13. (a) ; (b) ; (c) ; (d) ; (e) none of these
14. (a) ; (b) ; (c) ; (d) ; (e) none of these

Study questions on Introduction

1. New ways of thinking about algorithms

1. An algorithm *lacks* which of these features? (a) computes a function; (b) is deterministic; (c) may take an unreasonably long time; (d) works in discrete steps; (e) may take forever
2. Algorithm specifications presuppose (a) that input has occurred; (b) that processing has occurred; (c) that output has occurred; (d) the meaning of input; (e) that loops time out
3. Algorithms solve problems that are associated with (a) services; (b) protocols; (c) irrational numbers; (d) functions; (e) none of these
4. A function is computed by a(n) (a) service; (b) interactive protocol; (c) multi-agent system; (d) algorithm; (e) event-driven program
5. Input to an algorithm is (a) necessarily atomic; (b) obtained before algorithm execution; (c) obtained during execution; (d) necessarily compound; (e) possibly infinite
6. An algorithm is a(n) (a) program; (b) plan; (c) structure; (d) service; (e) process

2. Data structures and container classes

1. Arrays are structures that are (a) linked; (b) branching; (c) linear; (d) dynamically allocated; (e) none of these
2. Linked lists are (a) static; (b) branching; (c) linear; (d) stack allocated; (e) none of these
3. Graphs are implemented as (a) trees; (b) matrices or arrays of lists; (c) queues; (d) stacks; (e) none of these
4. A tree is a kind of (a) list; (b) array; (c) graph; (d) all of these; (e) none of these
5. (T-F) A linked list is of a length specified in its declaration.
6. (T-F) A linked-list node type is a simple type.
7. (T-F) The physical order (in memory) of the nodes of a linked list is the same as the order in which nodes are accessed.
8. 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
9. (T-F) Insertion of a new value into an ordered linked list takes more steps than insertion into an ordered array.
10. 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
11. (T-F) A stack is a last-in, first out structure.
12. To add to a stack, we carry out a(n) ______ operation. (a) dequeue; (b) enqueue; (c) pop; (d) push; (e) traversal
13. The push operation is carried out on (a) lists; (b) trees; (c) arrays; (d) stacks; (e) queues
14. It is impossible to push data onto a(n) _______ stack. (a) full; (b) empty; (c) initialized; (d) array-implemented; (e) list-implemented
15. A queue is a specialized kind of (a) simple type; (b) array; (c) collection; (d) tree; (e) stack
16. To remove data from a queue, we carry out the ______ operation. (a) dequeue; (b) enqueue; (c) pop; (d) push; (e) traversal
17. The first-in, first-out structure is the (a) list; (b) array; (c) queue; (d) stack; (e) tree
18. Each item in priority queue has a value denoting its (a) address; (b) relative priority; (c) search key; (d) numeric weight; (e) none of these
19. (T-F) A binary search tree is generally faster to search than a linked list that has the same number of nodes.
20. 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
21. A binary search tree node has up to two (a) root nodes; (b) children; (c) paths to a given node; (d) parents
22. 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
23. 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
24. 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
25. 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
26. (T-F) Hashing arranges items by comparing them with each other.
27. 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
28. Address calculation in source code is associated with (a) hashing; (b) dynamic allocation; (c) all array accesses; (d) recursion; (e) searching
29. 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
30. Which might be used in task scheduling? (a) functions; (b) vertices; (c) undirected graphs; (d) directed graphs; (e) none of these
31. (T-F) In a connected graph an edge exists between each pair of vertices.

32. 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
33. 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.
34. 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
35. 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

3. Mathematical foundations of algorithm analysis

1. A graph is a (a) vertex; (b) edge; (c) vertex and edge; (d) set of vertices and an edge; (e) set of vertices and set of edges
2. A language is a (a) string; (b) number; (c) set of numbers; (d) sequence of strings; (e) set of strings
3. 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
4. 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
5. $\subseteq$ denotes (a) set membership; (b) union; (c) conjunction; (d) a relation between sets; (e) negation
6. $\wedge$ denotes (a) set membership; (b) union; (c) AND; (d) a relation between sets; (e) negation
7. $\neg$ denotes (a) set membership; (b) union; (c) AND; (d) a relation between sets; (e) logical negation
8. $\cup$ denotes (a) set membership; (b) union; (c) AND; (d) a set; (e) negation
9. $\varnothing$ denotes (a) set membership; (b) union; (c) AND; (d) a set; (e) negation
10. $\in$ denotes (a) set membership; (b) union; (c) AND; (d) a relation between sets; (e) negation
11. A string is a (a) collection; (b) set; (c) tree; (d) sequence; (e) list
12. Where B and C are sets, $(B \times C)$ is (a) a pair of sets; (b) a set of pairs; (c) an arithmetic product; (d) a sequence; (e) a concatenation
13. $\vee$ denotes (a) set membership; (b) union; (c) AND; (d) OR; (e) implication
14. $\Rightarrow$ denotes (a) set membership; (b) union; (c) AND; (d) OR; (e) implication
15. Induction is a(n) (a) algorithm; (b) program; (c) proof; (d) proof method; (e) definition
16. Contradiction is a(n) (a) algorithm; (b) program; (c) proof; (d) proof method; (e) definition
17. Construction is a(n) (a) algorithm; (b) program; (c) proof; (d) proof method; (e) definition
18. The induction principle makes assertions about (a) infinite sets; (b) large finite sets; (c) small finite sets; (d) logical formulas; (e) programs
19. 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
20. 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
21. A proof in which we show that a tree with n vertices has a property, and 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
22. $\{1,2,3\} \cup \{2,4,5\} =$ (a) $\{\}$; (b) $\{1,2\}$; (c) 2; (d) $\{2\}$; (e) $\{1,2,3,4,5\}$
23. 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) none of these
24. A function f: $\{1,2,3\} \to \{0,1\}$ is a set of (a) integers; (b) ordered pairs; (c) sets; (d) relations; (e) none of these
25. Propositional logic is associated with (a) boolean variables and logical operators; (b) quantifiers; (c) predicates; (d) numbers and relational operators; (e) none of these

Longer answer questions

1. Write pseudocode for an algorithm that adds the elements of an array of integers. (B1)
2. Write pseudocode for an algorithm that finds the largest element of an array of integers. (B1)
3. Write pseudocode for an algorithm that returns *true* iff all elements of an array of integers are the same. (B1)
4. Write pseudocode for an algorithm that tells whether an array of integers contains an element with the value 5. (B1)
5. Describe the amount of time an algorithm labeled "B1" above takes, in relation to the size of the array, showing your reasoning. (B2)
6. Write an algorithm that, given input of x, displays all the prime numbers from 2 to x (B3)
7. Write pseudocode for a sorting algorithm. (B3)
8. Write pseudocode for an algorithm that returns $y = true$ iff any two elements of an array have the same value. (B3)
9. Why is a binary search tree an efficient way to store and retrieve data? (B4)
10. Why is a heap structure an efficient way to store and retrieve data? (B4)
11. What is a *graph* and how is it searched? (B5)
12. Describe a kind of tree, with efficiency properties, that can be derived from a weighted graph? (B5)
13. Describe an implementation of the *graph* data structure and a way to add an edge (B5)
14. Describe the inductive proof technique. (B6)
15. What is the principle of mathematical induction? (B6)

16. Prove *inductively* that the sum of all whole numbers from 1 to n is $(n^2 + n) \div 2$ (B6)
17. What are the domain and the range of the square-root function?\
18. Name common features of a *set*, a *dictionary*, and a *list*. Name features that distinguish the three.
19. In pseudocode, write an algorithm that computes $y = 2^x$ for input x
20. **Algorithm design**
    Write pseudocode or program code to compute or determine, with respect to an array of integers:
    0. The average
    1. Subscript of the first element that is same as its successor
    2. Number of elements, starting with first, that are all the same
    3. Smallest element
    4. Length of longest ascending sequence starting with first element
    5. Range (maximum minus minimum)
    6. Number of zeroes
    7. Subscript of largest element
    8. The median
    9. Tell whether all values are the same
    10. Sum of squares
    11. Tell whether values are in ascending order

    (a) Describe the time necessary to compute this, in relation to the size of the data file.
21. **Proof in propositional logic**
    Consider these axioms and theorems:
    a. for any p and q, $(p \rightarrow q)$ iff $(\neg p \vee q)$ (definition of $\rightarrow$)
    b. for any q $false \rightarrow q$ (follows from (a))
    c. for any p and q, $((p \rightarrow q) \wedge p) \rightarrow q$ (modus ponens)
    d. for any p and q, $((p \rightarrow q) \wedge \neg q) \rightarrow \neg p$ (modus tollens)
    e. for any p and q, $(p \rightarrow q) \rightarrow (\neg q \rightarrow \neg p)$ (contrapositive)
    f. a is true
    g. f is false
    h. $a \rightarrow b$
    i. $b \rightarrow \neg d$
    j. $b \rightarrow c$
    k. $\neg a \rightarrow e$
    l. $\neg f \rightarrow \neg d$
    m. $e \rightarrow \neg f$
    n. $g \rightarrow f$
    o. $h \rightarrow \neg c$

    Prove by inference, referring at each step to the axiom(s) used:
    0. c
    1. $b \vee \neg e$
    2. $c \vee d$
    3. $c \wedge \neg d$
    4. $e \rightarrow \neg d$
    5. $\neg b \vee \neg f$
    6. $\neg d$
    7. $a \vee e$
    8. $\neg g$
    9. $\neg h$
    10. $f \rightarrow \neg a$
22. **Induction**
    Solve the problem whose number is your classroom ID, mod 3, plus 1.
    1. Show by induction that every complete binary tree has $(2^k - 1)$ vertices, where k is the depth of the tree.
    2. Show by induction that every number whose binaryrepresentation ends in 1 is odd.
    3. Using the definition of a tree and the notion of induction, explain in your own words why every tree $G = (V, E)$ has exactly one more vertex than its number of edges.

Study questions on Topic 1 (Problem classes)

1. Vector, matrix, graph problems

1. An example of an order statistic is (a) average; (b) mode; (c) standard deviation; (d) median; (e) none of these
2. A topological sort is applied to (a) graph vertices; (b) an array; (c) a tree; (d) a logical formula; (e) none of these
3. The order statistic problem finds the ____ of an array (a) average element value; (b) median value; (c) *k*th largest element; (d) *k*th element; (e) sorted version
4. 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
5. 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
6. The prerequesite 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
7. String matching is a(n) _____ problem (a) vector; (b) optimization; (c) state-space search; (d) behavior-of-program; (e) interactive computation
8. 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
9. Matrix multiplication is a(n) _____ problem (a) non-optimization; (b) optimization; (c) state-space search; (d) behavior-of-program; (e) interactive computation
10. Path search is a(n) _____ problem (a) graph; (b) optimization; (c) state-space search; (d) behavior-of-program; (e) interactive computation

2. Constraint and optimization problems

1. 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
2. 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
3. Minimum spanning tree is an attribute of (a) arrays; (b) weighted graphs; (c) unweighted graphs; (d) sets of points; (e) none of these
4. Convex hull is an attribute of (a) arrays; (b) weighted graphs; (c) unweighted graphs; (d) sets of points; (e) none of these
5. Closest pair is an attribute of (a) arrays; (b) weighted graphs; (c) unweighted graphs; (d) sets of points; (e) none of these
6. Which of these is an optimization problem? (a) reachability; (b) traversal; (c) sort; (d) string search; (e) knapsack
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. *Knapsack* is a(n) _____ problem (a) non-optimization vector, matrix or graph; (b) optimization; (c) state-space search; (d) behavior-of-program; (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) interactive computation
16. Hamiltonian path is a(n) _____ problem (a) vector; (b) optimization; (c) state-space search; (d) behavior-of-program; (e) interactive computation
17. Topological sort is a(n) _____ problem (a) matrix; (b) optimization; (c) state-space search; (d) behavior-of-program; (e) interactive computation
18. 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

3. Logical 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. 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

1. Temporal logic is used to verify (a) algorithms; (b) specifications; (c) batch programs; (d) reactive systems; (e) formulas in predicate logic
2. Model checking uses (a) propositional logic; (b) predicate logic; (c) Hoare triples; (d) temporal logic; (e) unit testing
3. A Kripke structure diagrams (a) an algorithm; (b) a reactive system; (c) a module hierarchy; (d) a web interaction; (e) a multi-agent system
4. Safety asserts that (a) that desired states will obtain; (b) that undesired states will never occur; (c) termination; (d) total correctness; (e) efficiency
5. Liveness asserts (a) that desired states will obtain; (b) that undesired states will never occur; (c) termination; (d) total correctness; (e) efficiency
6. 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
7. Model checking is a method to verify (a) sorting; (b) algorithms; (c) poofs; (d) reactive systems; (e) efficiency
8. Reactive systems may be verified formally by (a) quality assurance; (b) unit testing; (c) model checking; (d) Hoare triples; (e) predicate logic
9. A computation tree is induced by (a) pseudocode; (b) program code; (c) machine code; (d) a Kripke structure; (e) a proof
10. Verification of reactive systems may use (a) propositional logic; (b) predicate logic; (c) Computation Tree Logic; (d) fuzzy logic; (e) pseudocode

Short and longer-answer questions on topic 1

1a. Problem specification

For problems 1-16:

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. (1a)

For most problems, you will need to use the universal or existential quantifiers (∀, ∃). Note that A, B are arrays, not sets, so that for example, to claim that all elements of B are ones, we would write, $(\forall i \leq |B|)\ B[i] = 1$.

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 consecutive elements of A differ.
7. y = *true* iff any two elements of A are the same.
8. y = the maximum value in A.
9. y = *true* iff the number of 0's in bit vector A is greater than the number of 1's.
10. y = the sum of the elements of A.
11. y = the index of the first occurrence of key value x in A, or 0 if not found.
12. y = *true* iff the first element is the smallest
13. y = *true* iff A has at least one pair of consecutive elements that are the same
14. y = *true* iff A consists of alternating 0s and 1s
15. y = *true* iff A has *no* pair of consecutive elements that are the same
16. y = *true* iff every element in A but the first and second is the sum of the previous two
17. y = *true* iff every element in A is divisible by 3
18. y = *true* iff some element in A is divisible by 3

1b. Optimization

1. Distinguish optimization problems from constraint-satisfaction problems, giving examples.
2. For each of the following, whether the problem is of the constraint type (C) or the optimization type (O). If it is an optimization problem give a corresponding constraint problem; if it is a constraint problem give a corresponding optimization problem.
   a. Finding a path in a graph that visits all vertices
   b. Finding the pair of (x, y) pairs in a two-dimensional plane that is closest together of all the pairs
   c. Finding a set of weighted items in a collection, such that the weights add up to less than x
   b. Searching an array of *location* objects (x, y) for a location in which $y > b$
   c. Finding a path in a graph from vertex a to vertex b
   d. Telling whether a set of (x, y) pairs contains two pairs whose distance apart is less than six units
   e. Telling whether a graph can be colored with fewer than four colors
   f. Finding a subgraph of a graph that is a tree and contains all the vertices in the graph
   g. Finding the shortest path in a graph that visits all vertices
   h. The closest-pair problem
   i. Finding the smallest element of an array
   j. Finding the maximum-valued set of weighted items in a collection, such that the weights add up to less than x
   k. Finding the farthest-apart pair of locations in a collection of ordered pairs
   l. The Traveling Salesperson Problem
   m. Finding the smallest number of colors that can color a graph (1b)
19. What are the components of a graph that determine paths?
20. What does the shortest-path problem apply to?
21. What is a subgraph that is connected and acyclic, and provides a minimum-cost subway system design?
22. What kind of path passes through all vertices of a graph exactly once and returns to the start?
23. Describe the inputs and postcondition of the shortest-path problem.
24. What problem finds the minimum-cost graph through all vertices of a graph?
25. What problem seeks the maximum-valued subset of weighted items totaling less than a given weight?
26. What problem involves grouping items so as to fit them in a set of containers of fixed size?
27. What is an ongoing series of responses to inputs, queries, or requests?
28. What sort of problem requires servicing multiple input streams, possibly under time constraints?
29. Satisfiability is a property of what sort of expression?
30. Distinguish satisfiability, validity, and evaluation of propositional-logic formulas.
31. What is a way to show that computational problem A is at least as hard as problem B?
32. Describe the correspondence between a decision problem and a language.
33. What kind of path passes through all vertices of a graph exactly once and returns to the start?

1c. Verification of reactive systems

1. Describe a verification method commonly used for reactive systems. (1c)
2. Relate the following: Computation Tree Logic; temporal logic; Kripke structures; reactive systems. (1c)
3. What is Computation Tree Logic used for? (1c)
4. What is temporal logic used for and how? (1c)
5. What is a Kripke structure? (1c)

6. 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.) (1c)
7. What sort of logic is used in model checking? (1c)
8. Name the two quantifiers used in predicate logic and the corresponding ones used in CTL. (1c)
9. Write a CTL formula that applies to a given reactive system and asserts, for formulas ϕ and φ, that on this system, (1c)
   a. ϕ will never hold.
   b. there is a way for ϕ to become true
   c. after the next state transition, ϕ will hold.
   d. only after ϕ holds can φ cease to be true.
   e. there is no way to get into the state ϕ.
   f. there is a way to get out of the state ϕ.
   g. there is no way to avoid getting into state ϕ.
   h. after some future state transition, ϕ will hold.
   i. after some future state transition, ϕ will not hold.
   j. ϕ will hold as long as φ is not true.
   k. ϕ or φ will hold in all future states.
10. Translate to English:
    a. $EF\phi$
    b. $AG\neg\phi$
    c. $AX\phi$

Miscellaneous

1. Use the principle of optimality to describe finding the best batting average among players on teams in the major baseball leagues.
2. In what sense can one problem be said to be *reducible* to another?
3. Using the language of predicate logic, give the *postcondition* for an algorithm (but not the algorithm itself) that accepts an array *A* and tells whether any two consecutive elements of *A* are the same.
4. Design 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 constrains 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

Study questions on Topic 2 (Verification)

1. Assertions and correctness

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) none of these
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) none of these
6. Whether a program fits its specifications depends on (a) syntactic correctness; (b) semantic correctness; (c) logical provability; (d) QA's opinion; (e) chance
7. 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

2. Loop invariants and induction

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) none of these
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) none of these
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) none of these

3. Formal proof methods and Hoare triples

1. 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
2. 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
3. $<\phi> P <\psi>$ is a (a) precondition; (b) loop invariant; (c) postcondition; (d) Hoare triple; (e) first-order logic formula
4. In $<\phi> P <\psi>$, ϕ is a (a) precondition; (b) loop invariant; (c) postcondition; (d) Hoare triple; (e) propositional-logic formula
5. In $<\phi> P <\psi>$, φ is a (a) precondition; (b) loop invariant; (c) postcondition; (d) Hoare triple; (e) propositional-logic formula
6. In $<\phi> P <\psi>$, P is a (a) precondition; (b) loop invariant; (c) postcondition; (d) program; (e) propositional-logic formula
7. 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
8. 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

Short and longer-answer questions for topic 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 are Hoare triples used for?
4. What is the likely relationship between a loop invariant and a postcondition?
5. Explain the relationship between loop invariants and induction.
6. Identify the components and meaning of $<\phi>$ P $<\psi>$ as discussed in this class.
7. Describe partial correctness and distinguish from total correctness.
8. Identify and put in plain English:
   (a) $\langle \phi \wedge b \rangle\ P\ \langle \phi \rangle \Rightarrow \langle \phi \rangle$ **while** b $\{P\}\ \langle \phi \wedge \neg b \rangle$
   (b) $\langle \psi [E/x] \rangle\ x \leftarrow E\ \langle \psi \rangle$
   (c) $\langle \phi \rangle\ P_1\ \langle \eta \rangle \wedge \langle \eta \rangle\ P_2\ \langle \psi \rangle \Rightarrow \langle \phi \rangle\ P_1\ P_2\ \langle \psi \rangle$
   (d) $\langle \phi \wedge b \rangle\ P_1\ \langle \psi \rangle \ \wedge\ \langle \phi \wedge \neg b \rangle\ P_2\ \langle \psi \rangle \Rightarrow$
   $\langle \phi \rangle$ **if** b $\{P_1\}$ **else** $\{P_2\}\ \langle \psi \rangle$

2b. Explain loop invariant

Explain how the loop invariant in the algorithm shown on the slide helps to establish the validity of the postcondition.

0. Sum (slide 8)
1. Largest (#12)
2. Linear search (#13)
3. Strcpy (#14)
4. Array insertion (#15)
5. Insertion sort (#16)
6. Bubble sort (#17)
7. Is-largest (#23)
8. Search-stack (#24)
9. Factorial (#28)
10. Power (#34)

2b, 2c. Proofs of correctness

For the algorithm among the following that matches your classroom ID:

(a) Write a valid postcondition (see topic 1)
(b) Write a valid loop invariant that helps establish that the postcondition holds (2b)
(c) Explain the proof of correctness. (2c)

0. ***Count-spaces*(*s*)**

```
> Returns the number of spaces in a string, s.
y ← 0
i ← 1
while i ≤ length(s) do
    if s[ i ] = ' '
        y ← y + 1
    i ← i + 1
return y
```

1. ***Search-stack* (*S, key*)**

```
> Tells whether stack S contains key
found ← false
while not empty(S)
    test ← Pop(S)
    if test = key
        found ← true
return found
```

2. **Quotient (*a*, *b*)**

```
> Performs integer division
y ← 0
s ← a – b
while s > 0
    s ← s – b
    y ← y + 1
return y
```

3. **All-same (*A*)**

```
y ← true
for i ← 2 to |A|
    If A[ i ] = A[i – 1]
    y ← false
return y
```

4. **Largest-to-right (*A*)**

```
> Returns result of moving the largest element of A[1..n]
> into position n of A,.
largest ← 1
for i ← 2 to |A| do
    if A[ i ] > A [largest]
    largest ← i
A[largest] with A[|A|]
return A
```

5. ***Index-of-largest* (A)**

```
> Returns index of the largest element of A
y ← 1
for i ← 1 to |A| – 1)
    if A[ i ] < A[y]
        y ← i
    i ← i + 1
return y
```

6. **Fact (*x*)**
```
> Computes factorial function:
y ← 1
i ← 1
while i < x
    y ← i × y
    i ← i + 1
return y
```
7. Max (*A*)
```
> Returns largest element of A:
If |A| > 0
    y ← A[1]
i ← 1
while i < |A|
    if y < A[ i ]
        y ← A[ i ]
    i ← i + 1
return y
```
8. **Pow (a, b)**
```
> returns a^b
y ← a
i ← 1
while i < b
    y ← a × y
    i ← i + 1
return y
```
9. **Sum (*A*)**
```
> Computes sum of array elements
y ← 0
i ← 1
while i ≤ |A|
    y ← y + A[ i ]
    i ← i + 1
return y
```
10. **Product (x, y)**
```
>Performs multiplication
result ← 0
For i ← 1 to x
    result ← result + y
Return result
```
11. **Which-sort (A)**
```
for i ← size(A) down to 2 do
    A← Largest-to-right (A[1.. i ])
```
You may assume that *Largest-to-right* (#4 above) is correct.

12. Prove or argue persuasively that the code below will terminate rather than hanging the computer.
```
int i = 1;
while (i < 100)
{
    System.out.print( i );
    if (i % 2 == 0)
        i *= 2;
    else
        i++;
}
```
13. Write an algorithm to compute the function $y = a^b$, for inputs *a, b*. Show correctness.
14. Write an algorithm to compute the sum of the elements of an input array, A. Show correctness.
15. (a) Express an algorithm iteratively for the factorial function;
(b) By use of comments, show that it terminates and show partial correctness.
16. Write 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. Show its correctness.
17. Write an algorithm that tells whether an array of integers is in ascending order; prove correctness.
18. The Insertion Sort algorithm can be written with a subroutine that uses a loop that inserts values successively into a sorted initial sequence until this sequence is the whole array.
(a) Write the insertion and the sorting algorithms in pseudocode.
(b) Show correctness of the insertion algorithm, using the tools we discussed in class and relating your work to the form, $\langle\phi\rangle P\langle\psi\rangle$.
(c) Write iterative pseudocode for Insertion Sort, using the insertion algorithm.
19. Write a series of statements that divide input value *a* by value *b* without the division operator and, by use of preconditions, postconditions, and loop invariants, prove or argue persuasively that your code will not crash the program with a divide-by-zero error.
20. Write pseudocode for a linear search and prove correctness using loop invariant.
21. Write pseudocode for an algorithm that finds the maximum value in an array, and prove correctness.

Study questions on Topic 3 (Algorithm analysis, recurrences, and brute force)

1. Big-O notation and algorithm analysis

1. Function g is an upper bound on function f iff for all x, (a) $g(x) \le f(x)$; (b) $g(x) \ge f(x)$; (c) $g = O(f)$; (d) $f = \Omega(g)$; (e) none of these
2. Function g is a lower bound on function f iff for all x, (a) $g(x) \le f(x)$; (b) $g(x) \ge f(x)$; (c) $f = O(g)$; (d) $g = \Omega(f)$; (e) none of these
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 α computes function f; (b) algorithm α produces a result in time at least $f(n)$ for inputs of size n; (c) algorithm α produces a result in time not greater than $f(n)$ for inputs of size n; (d) algorithm T runs in time α; (e) algorithm f computes function T on data α
7. The theorem, $T_1(n) \in \Theta(g_1(n)) \wedge T_2(n) \in \Theta(g_2(n)) \Rightarrow T_1(n) + T_2(n) \in \Theta(\max\{g_1(n), g_2(n)\})$ 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. $\log_2 n \in O(\mathrm{sqrt}(n))$ means that the logarithm function _____ the square root function (a) grows as fast as; (b) grows no faster than; (c) grows at least 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) $\Theta(\lg n)$; (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 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
2. 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
3. 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
4. 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
5. 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

6. Peano defined **N** (a) by induction; (b) by contradiction; (c) by enumeration; (d) by encryption; (e) as a subset of **R**
7. Any computable function can be defined (a) by induction; (b) by contradiction; (c) by enumeration; (d) by encryption; (e) as a subset of **R**
8. 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
9. A recursive function definition (a) uses a *while* loop; (b) lists all possibilities; (c) contains a call to the function itself; (d) is impossible; (e) is inefficient
10. Recurrences are used in (a) input specification; (b) proofs of correctness; (c) time analysis; (d) type checking; (e) none of these
11. 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
12. 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(lg n); (c) O(n lg n); (d) O(n); (e) O(n^2)
13. 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(lg n); (c) O(n lg n); (d) O(n); (e) O(n^2)
14. 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(lg n); (c) O(n lg n); (d) O(n); (e) O(n^2)
15. 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(lg n); (c) O(n lg n); (d) O(n); (e) O(n^2)

3. Brute force

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(___) (a) 1; (b) lg 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(lg 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 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(___) (a) 1; (b) lg 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(lg n); (c) O(n); (d) O(n lg 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(___) (a) 1; (b) lg 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(lg n); (c) O(n); (d) O(n lg 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(lg n); (c) O(n); (d) O(n lg 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(lg n); (c) O(n); (d) O(n lg 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(lg n); (c) O(n); (d) O(n lg 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(lg n); (c) O(n); (d) O(n lg 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 $f(\lg n)$
21. Computing the factorial function has an efficient _____ algorithm (a) brute-force; (b) divide-and-conquer; (c) greedy; (d) dynamic-programming; (e) probabilistic

4. Sorting

1. Insertion sort is (a) O(1); (b) O(lg n); (c) O(n); (d) O(n^2); (e) O(2^n)
2. The Bubble sort is what kind of algorithm? (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic
3. The insertion sort is what kind of algorithm? (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic
4. Selection sort is O(___) (a) 1; (b) lg n; (c) n; (d) n^2; (e) 2^n
5. Selection sort uses (a) divide and conquer; (b) brute force; (c) the greedy approach; (d) dynamic programming; (e) a probabilistic approach
6. A recursive-case running time of ($n + T(n-1)$) indicates ____ time (a) constant; (b) logarithmic; (c) linear; (d) quadratic; (e) exponential
7. Bubble sort is (a) O(1); (b) O(lg n); (c) O(n); (d) O(n^2); (e) O(2^n)

5. 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. Exhaustive search is (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic
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

Short and longer answer questions on Topic 3

3a. Define and use the big-O, theta, and big-omega notations

Explain why each of the following is true or false:

1. $f(n) = 1 \in O(n)$
2. $f(n) = \lg n \in O(n^2)$
3. $f(n) = n^2 \in O(1)$
4. $f(n) = n^2 \in O(n^3)$
5. $f(n) = n^2 \in O(n)$
6. $f(n) = 4n^2 \in O(n^2)$
7. $f(n) = 3n + 10 \in O(\lg n)$
8. $f(n) = n \lg n \in O(n^2)$
9. $f(n) = n^2/8 \in O(n \lg n)$
10. $f(n) = 100 \in O(n^3)$
11. $f(n) = n \in O(n^2)$
11. In mathematical notation, list some of the most widely discussed categories that classify algorithms by performance.
12. 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?
13. What is a *lower bound* on a function f? A tight bound? Distinguish big-O, Ω, and θ, and relate to algorithm analysis.
14. For function f, $O(f)$, $\Omega(f)$, and $\theta(f)$ are sets of functions. What are the set containment relationships among the following:
    (a) $O(n^2)$, $O(\lg n)$
    (b) $\Omega(n^2)$, $\Omega(\lg n)$
    (c) $\theta(n^2)$, $\theta(\lg n)$
15. Express in big-O notation:
    $T(n) = 500n + n^5 / 10 + 80 \lg n$
16. Give an example of a lower bound on the function $f(x) = x^3$
17. Referring to other notations and their meanings, explain why $\theta(f)$ called a *tight bound* on f.
18. What is the main notation for expressing the complexity of algorithms as *lower bounds*, and what is the intuitive meaning of this notation?
19. What is the main notation for expressing the complexity of algorithms as *upper bounds*, and what is the intuitive meaning of this notation?
20. Give the names of five asymptotic efficiency classes, in ascending order of time complexity.
21. Group the following time functions by efficiency class:
    (a) $t(n) = 6 \lg n$;
    (b) $t(n) = 5 n^3$;
    (c) $t(n) = n / 4$;
    (d) $t(n) = 2^n$;
    (e) $t(n) = 10 n \lg n + 100$;
    (f) $t(n) = 80n + 1000$
22. Simplify (a) $O(2n)$; (b) $O(n - 8)$; (c) $O(4n^2 + 2n)$
23. Define
    a. O notation
    b. Ω notation
    c. Θ notation
24. Use two simple formulas to show the relation among O, Θ, and Ω notations.
    (*Hint:* if $f(n) \in \Omega(g(n))$, what follows?)
25. What is the name of a particular mathematical technique or notation for defining a function, that converts straightforwardly into code or pseudocode?
26. What is the name of a particular mathematical technique or notation for defining a function, that converts straightforwardly into code or pseudocode?
27. What is worst-case time for binary search, and why?
28. Using the definition of big-O notation, explain why "$f(n) = 4n^2 \in O(n^2)$" is true or false
29. Using the definition of Θ, explain why
    (a) $\Theta(n / 2) = \Theta(n)$
    (b) $\Theta(3n) = \Theta(n)$
    (c) $\Theta(2n^2 - 5) = \Theta(n^2)$

3b, c, d. Algorithm analysis using recurrence

1. For each of the following,
   (a) Write a solution in iterative pseudocode or a programming language.
   (b) Write a recursive version. (3b)
   (c) Express the algorithm as a recurrence. (3c)
   (d) Based on (c), write a recurrence to express the time function of the algorithm, and solve the recurrence using big-O or Θ notation. (3d)
   a. the *strlen* subprogram, which takes an address as a parameter and returns the length of the string (distance from a null character).
   b. the factorial function where
      $fact(4) = 4 \times 3 \times 2 = 24$
   c. summation of $1..x$;
   d. the numeric value of a bit vector
   e. the smallest value in a sequence of integers
   f. find the number of occurrences of a string, s, in a text file, t
   g. find the largest element of a subsequence in an array. Signature: $Max(A, first, last)$, where A is an array, *first* and *last* are subscripts. Algorithm should return the largest element of the sequence from $A[first]$ to $A[last]$, inclusive
   h. The Insertion Sort algorithm, written with a subroutine that uses a loop that inserts values successively into a sorted initial sequence until this sequence is the whole array.
2. Discuss the complexity of this search algorithm, write it as a recurrence, and argue for its correctness, using preconditions, postconditions, and a loop invariant:

```
Some-search (A, first, last, key)
while first ≤ last do
    middle ← (first + last) / 2
    if key < A[middle]
        last ← middle – 1
    else if key > A[middle]
        first ← middle + 1
    else return true
return false
```

3. State complexity of each algorithm below and justify your answer.

```
Which-sort (A)
for i ← size(A) down to 2 do
    Largest-to-right (A, i )

Largest-to-right (A, n)
largest ← A[1]
for i ← 2 to n do
    if A[ i ] > A [largest]
        largest ← i
swap A[largest] with A[n]
```

4. (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 Θ notation.

```
num_passes ← 0
Repeat
    swapped ← false
    for i ← 1 to size(A) - 1
        if A[ i ] > A[ i + 1]
            swap(A[ i ], A[ i + 1] )
            swapped ← true
    num-passes ← num-passes + 1
until swapped is false
```

3e. Describing a brute-force design

1. (See topic 1 for descriptions of these problems) Describe, but don't write, a brute-force solution to the following:
    a. The convex-hull problem
    b. The knapsack problem
    c. The satisfiability problem
    d. The closesst-pair problem
    e. Traveling salesperson
    f. Minimal spanning tree
    g. Search for a path in a graph
    h. Minimal spanning tree for a graph
    i. Hamiltonian Path
    j. The independent-set problem
2. What is brute force?
3. What is an approach to algorithm design that addresses problems in a straightforward way?
4. Describe *exhaustive search* and associate it with an algorithm-design approach.
5. Name a brute-force algorithm for sorting an array, and one for searching an array, and describe them, giving their time analysis.
6. Levitin, pp. 59-60, Ex. 1, 2(b, c), 3 (b, c, d), 5, 8(a). *Show your work.*
7. Levitin, p. 67, Ex. 1 (g) (summations; show work), 2 (d) (summation; show work), 4 (mystery algorithm)
8. Levitin, p. 76, Ex. 1 (b, c) (solve recurrences),
9. Levitin, p. 76, Ex. 3 (solve recurrences)
10. Give pros and cons of the *empirical* and *theoretical* approaches to algorithm analysis.

3f, g. Analysis of brute-force algorithms

Write brute-force algorithms for the following problems in iterative pseudocode or a programming language. String-searching takes parameters *A* and *B*, finds first occurrence of *B* inside *A*. It uses a string-matching algorithm that takes two string parameters and tells whether or not they are the same.

(a) Express the algorithm as a recurrence or set of recurrences. (3f)

(b) Based on (a), write and solve a recurrence to express the time function of the algorithm. (3g)

1. Find the intersection of the sets represented by two arrays of integers.
2. The selection sort
3. Tell whether a given destination vertex in a graph is reachable from a given source vertex
4. Find the longest common subsequence of two arrays
5. Find the mode (most frequent value) of an array
6. Find the index of the longest ascending subsequence in an array
7. The insertion sort
8. Tell whether any elements of an array are duplicated
9. Find the median (middle) value in an array
10. Return the subscripts of the start and end of the longest run of array elements in ascending order
11. Find the index of the longest subsequence of identical elements in an array
12. The bubble sort

Extra credit:

13. string-matching (string searching)
14. tell whether a given destination vertex in a graph is reachable from a given source vertex.
15. the independent-set problem.
16. determine whether a given set of natural numbers may be partitioned into two subsets of equal sum.
17. find the disjunction (OR) of two arrays of integers.
18. find the conjunction (AND) of two arrays of integers.

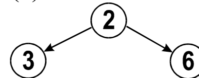
Study questions on Topic 4 (Divide and conquer)

1. Vector algorithms

1. The Quicksort partition step is (a) O(1); (b) O(lg n); (c) O(n); (d) O(n^2); (e) O(2^n)
2. 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)
3. 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)
4. 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
5. 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
6. The binary search uses which approach to algorithm design? (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic
7. Quicksort uses which approach to algorithm design? (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic
8. After each step of the binary search, the quantity of remaining data to be searched is on average (a) 1; (b) lg n; (c) $n \div 2$; (d) n; (e) $2n$
9. When the quantity of remaining data to be processed in an algorithm, at each step, is ($n \div 2$), the complexity is O(___) (a) 1; (b) lg n; (c) n; (d) n^2; (e) 2^n
10. A recursive-case running time of ($1 + T(n \div 2)$) indicates ____ time (a) constant; (b) logarithmic; (c) linear; (d) quadratic; (e) exponential
11. A recursive-case running time of ($n + T(n \div 2)$) indicates ____ time (a) constant; (b) n lg n; (c) linear; (d) quadratic; (e) exponential
12. 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)
13. 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 disposed of; (d) effort can be saved by saving the results of previous effort in a table; (e) none of these

2. Tree 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) $n \div 2$; (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(__) (a) 1; (b) lg n; (c) n; (d) n lg n; (e) n^2
5. 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
6. The figure below is a (a) linked list; (b) minimum heap; (c) binary search tree; (d) hash table; (e) maximum heap



7. The common implementation of a heap is (a) array; (b) linked list; (c) doubly-linked structure; (d) multi-linked structure; (e) none of these
8. 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
9. (T-F) In a maximum heap, each node's left child stores a value that is less than that of the parent.
10. (T-F) In a maximum heap, each node's right child stores a value that is greater than that of the parent.
11. The depth of a heap of size n is close to (a) 1; (b) $\log_2 n$; (c) the square root of n; (d) n / 2; (e) n^2
12. 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
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. (T-F) A heap is implemented by a binary search tree with nodes linked by pointers.
15. The running time for *heap-extract-min* is (a) O(1); (b) O(lg n); (c) O(n lg n); (d) O(n); (e) O(n^2)
16. The running time for the algorithm presented in class as *heapify* is (a) O(1); (b) O(lg n); (c) O(n lg n); (d) O(n); (e) O(n^2)
17. The running time for *heap-sort* is (a) O(1); (b) O(lg n); (c) O(n lg n); (d) O(n); (e) O(n^2)
18. The running time for *build-heap* is (a) O(1); (b) O(lg n); (c) O(n lg n); (d) O(n); (e) O(n^2)

3. Solutions to geometric problems

1. 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. The solution to convex-hull that recursively finds triangles defined by fartherst points is an instance of (a) brute force; (b) divide and conquer; (c) greedy algorithms; (d) dynamic programming; (e) probabilistic algorithms
3. 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

4. Graph algorithms

1. The breadth-first search (a) uses a queue; (b) uses a stack; (c) searches an array; (d) searches a tree; (e) none of these
2. The depth-first search (a) uses a queue; (b) uses a stack; (c) searches an array; (d) searches a tree; (e) none of these
3. 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
4. Breadth-first search is O(___) (a) 1; (b) lg n; (c) n; (d) n^2; (e) 2^n
5. Depth-first search is O(___) (a) 1; (b) lg n; (c) n; (d) n^2; (e) 2^n

5. Decrease by constant factor

1. 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) $2T(n-1) + O(1)$; (e) none of these
2. $T(n) = T(n / b) + f(n)$ is consistent with (a) decrease by constant; (b) decrease by one; (c) decrease by constant factor; (d) $O(n^2)$; (e) $O(n)$
3. 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) $O(n)$
4. Quicksort is of the algorithm type (a) decrease by constant; (b) decrease by one; (c) decrease by constant factor; (d) $O(n^2)$; (e) $O(n)$
5. 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
6. 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

Short and longer-answer questions for topic 4

1. In divide and conquer, what gets divided?
2. Distinguish the breadth- and depth-first searches. What do they search in and what do they search for?
3. What is it about divide-and-conquer that sometimes offers fast running time?
4. Relate the notions of *order statistics* to the notion of *median*.
5. What is worst-case time for binary search, and why?
6. Give the complexity of the *Heapify* algorithm and justify your answer.
7. Give the complexity of the *Heap-insert* algorithm and justify your answer.
8. Give the complexity of the *Heap-sort* algorithm and justify your answer.
9. Write a recurrence for the function computed by the Heapify algorithm. (4b)
10. Based on a solution to the previous question, write a recurrence for the running-time function of the Heapify algorithm. (4b)
11. Compare BST search and binary search.
12. Give the complexity of the algorithm below and justify your answer.

```
BST-search (root, 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))
```

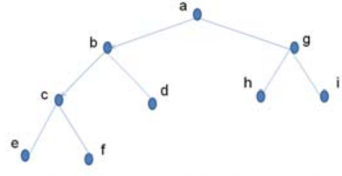
12. Give the complexity of the algorithm below and justify your answer.

```
Heapify()
L ← Left( i )
R ←  Right( i )
if L ≤ Heap-size(A) and A[L] < A[ i ]
    smallest ←  L
else
    smallest ←  i
if R ≤ Heap-size(A) and A[R] < A[smallest]
    smallest ←  R
if smallest ≠ i
    exchange A[ i ] with A[smallest]
    Heapify(A, smallest)
```

13. Give the complexity of the algorithm below and justify your answer.

```
OS(k, A) =
    A[pivloc(A)]            if k = pivloc(A)
    OS(pivloc(A) – k, A[1..pivloc(A)-1]
                            if k < pivloc(A)
    OS(k – pivloc(A), A[pivloc(A)+1, |A|)
                            otherwise
```

14. Explain:
    $T_{uick}(n) =$
    1 if $n \leq 1$
    $O(n) + 2T_{Quick}(\lceil (n - 1) / 2 \rceil)$ otherwise
15. 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.

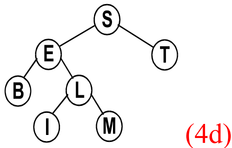


\

16. Describe these divide-and-conquer algorithms and explain how they use the divide-and-conquer strategy to achieve efficiency:
    a. Binary search
    b. Quicksort
    c. Merge
    d. BST insertion
    e. BST search
    f. *Heapify*
    g. *Extract-min* for heaps
    h. *Heap-insert*
    i. *Heapsort*
17. a. Write a *divide-and-conquer* algorithm to solve the following problems. (4a)
    b.Write a recurrence that defines the function computed. (4b)
    c. Write a time recurrence; solve it to give an analysis. (4c)

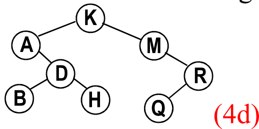
    i. inserting a value in a data structure that has the BST property, in such a way that the BST property is maintained after the insertion process.
    ii. Quicksort
    iii. Rearrange a complete binary tree, stored in an array, so that it has the heap property, given the array, a node, and the precondition that its two children have the heap property
    iv. Return the index of an element of a sorted array that matches a search key, given the array and the search key; return 0 if no match exists
    v. Given bit vector A, arranged so that all 1's are after any of the zeroes, return the number of 0's.
    vi. 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. Find k.
18. Discuss an approach to the convex-hull problem that is faster than brute force, including discussion of the complexity of the algorithm.
19. Explain inorder tree traversal and give the design approach and running time. (4d)
20. Name and explain two tree-traversal methods. (4d)

21. 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? (4e)
22. Contrast the depth-first and breadth-first search algorithms. Explain their running times. (4e)
23. Contrast the preorder, inorder, and postorder tree traversals. Explain their running times. (4e)
24. Describe and contrast two efficient graph search algorithms. (4e)
25. Describe depth-first search and breadth-first search.
26. Give the nodes visited in the following tree, in the order visited, in a *preorder* (postorder, inorder) traversal. Give the algorithm's complexity.



(4d)

27. Give the nodes visited in the following tree, in the order visited, in a *postorder* (postorder, inorder) traversal. Give the algorithm's complexity.



(4d)

28. How does the BST data structure support a divide-and-conquer algorithm design strategy? (4f)
29. Describe the merge sort, including complexity analysis.
30. Descibe the merge algorithm, including preconditions, postconditions, and complexity analysis.
31. How does a certain divide-and-conquer algorithm for order statistics use *Partition* to improve on the efficiency of brute-force order statistic?
32. Discuss an approach to the closest-pair problem that is faster than brute force, including discussion of the complexity of the algorithm.
33. Write an algorithm that determines whether a given binary tree has the BST property. Analyze.
34. Write an algorithm that determines whether a given binary tree has the heap property. Analyze.
35. What are the running times of *heapify, extract-min,* and *insert* for heaps? Relate this to the data structure used.
36. Code and test divide-and-conquer algorithms for:
    a. Binary search
    b. Quicksort
    c. Breadth-first search
    d. Depth-first search
    e. BST search
    f. BST insertion
    g. Order statistic
    h. Merge sort
    i. Convex hull
    j. Closest pair
37. Levitin, p. 128, Ex. 1 (array maximum); p. 139, Ex. 6 (binary search for elements in a range); p. 143, Ex. 2 (number of leaves in a binary tree), p. 143, Ex. 8 (internal path length in binary tree); p. 154, Ex. 3; (closest pair of points on plane) – pseudocode is OK; p. 170-171, Ex. 1 (depth-first search of graph)

Study questions on Topic 5 (Greedy and other efficient optimization algorithms)

1. Optimal-substructure property

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 when it works, but does not always work, 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. The optimal substructure property asserts that (a) if *S* solves an optimization problem, then any component of a *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
9. Hill climbing is a(n) ____ approach (a) divide-and-conquer; (b) brute-force; (c) greedy; (d) dynamic-programming; (e) probabilistic
10. 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
11. All problems with greedy solutions have (a) recursive structures; (b) many local optima; (c) th e optimal-substructure property; (d) tree structures; (e) constraint satisfaction as the goal
12. 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
13. 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
14. Greedy solutions exist for (a) no problems; (b) no optimization problems; (c) some problems; (d) all problems; (e) none of these
15. 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
16. Greedy algorithms tend to be (a) O(lg n); (b) O(n); (c) O(n^3); (d) O(2^n); (e) none of these

2. Greedy graph algorithms

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 Kruskal, Prim, and Dijkstra algorithms have what approach to algorithm design in common? (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic
6. The single-source shortest path problem has an efficient well-known solution of the type (a) brute force; (b) greedy; (c) depth-first search; (d) divide-and-conquer; (e) none of these
7. 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

8. The Dijkstra algorithm finds (a) minimal spanning tree; (b) single-source shortest path; (c) a traversal; (d) maximal path; (e) a compressison encoding tree
9. Adding elements to a set according to weight and value solves the _____ problem (a) shortest-path; (b) minimal-spanning-tree; (c) knapsack; (d) convex-hull; (e) sorting

3. Compression and packing

1. Huffman's algorithm is (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic
2. 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
3. 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
4. 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

4. 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

5. 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

Short and longer answer questions for topic 5

5a. Greedy approach

1. What algorithm finds the single-source shortest path in a graph and how?
2. What two algorithms find minimal spanning trees in graphs and how?
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.

5b. 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. How does the optimal substructure property apply to minimal spanning trees?

5c. Greedy graph algorithms

1. Describe an efficient way to solve the minimal-spanning-tree problem, giving an algorithm and its running time.
2. Briefly describe and analyze the running time of
   a. Prim's algorithm
   b. Dijkstra's algorithm
   c. Kruskal's algorithm
3. Design a greedy algorithm to tell whether a graph is cyclic; write a recurrence to define the function computed; write a time recurrence; state complexity.
4. Show that minimal spanning tree (Prim, Kruskal) is not necessarily the same as tree of single-source shortest paths (Dijkstra).
5. Show how the optimal-substructure property applies with
   a. Minimal spanning tree (Kruskal)
   b. Minimal spanning tree (Prim)
   c. Single-source shortest path (Dijkstra)
   d. Data compression (Huffman)
   e. Knapsack packing
   f. the continuous-knapsack problem
6. Give and explain solutions for the previous problems, with time analysis.
7. Describe an algorithm to find the shortest path from a certain vertex in a graph to a certain other vertex. Name the *two* design approaches used.
8. Name a property of the knapsack packing problem that enables an efficient solution; describe that solution.
9. Describe two solutions to the minimal-spanning-tree problem. What is the design approach used by these algorithms and why?
10. Show that the optimal substructure property applies to searches for minimal paths in weighted graphs.
11. Does the Prim algorithm produce a *unique* optimal solution? Explain.
12. Does the Kruskal algorithm produce a *unique* optimal solution? Explain.
13. Show how the optimal-substructure property applies to the minimal-spanning-tree problem.
14. Show that the minimal spanning tree (Prim, Kruskal) is not necessarily the same as the tree of single-source shortest paths (Dijkstra).
15. Show how the optimal-substructure property applies to the single-source shortest path problem.
16. Show how the optimal-substructure property applies to the continuous-knapsack problem.
17. Show how the optimal-substructure property applies to the problem solved by Huffman's code-defining algorithm.\
18. Compare pseudocode in the slides and the textbook for Dijkstra's algorithm. What is the running time of the Dijkstra algorithm and why?
19. Defend or refute: The optimal-substructure property guarantees that for any problem, a greedy algorithm exists that finds an optimal solution.
20. Compare pseudocode for Prim and Kruskal's algorithms in the instructor's slides with the pseudocode in Johnsonbaugh and Schaefer. Are they of the same complexity? Do they have the same depth of loop nesting? Which is clearer and why?
21. Compare the time complexities of the search algorithms for hash tables stored with open addressing (linear probe algorithm) and buckets (lists). State the range of possible load-factor values for each implementation and explain.

5d. Dynamic programming

1. What is dynamic programming about? Describe an algorithm of this kind.
2. Consider the following recurrence:
   Binomial (n, k) =
   (1 if $n = k$ *or* $k = 0$
   (Binomial ($i - 1, j - 1$) +
   (Binomial ($i - 1, j$) otherwise
   (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. Express 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] ← 1, F [1] ← 1
For i ← 2 to x do
    F [ i ] ← F [ i – 1] + F [ i – 2]
Return F [ x ]
```

What is the design approach, and what resource trade-offs are involved?

2. Analyze:

```
Closure (M [n, n])
for i ← 1 to n do
    for j ← 1 to n do \
        for k ← 1 to n do
            if M [i, j ] ∧ M [ j, k ]
                M [i, k ] ← true;
Return M
```

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

5e. 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 sorting 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. In dynamic-programming algorithms, what resource defines the main overhead of using this method?
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 both be categorized as greedy, or both 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 algorithm to decide path of descent of a B tree.
17. Find complexities of algorithms that compute binomial coefficent, C(n, k), that use recursion (brute force) and dynamic programming.
18. Levitin, p. 271, Ex. 7
19. p. 276, Ex. 3
20. p. 283, Ex. 4(a)
21. p. 292, Ex. 1 (See D. Keil C++ code, available in course docs)
22. p. 303, Ex. 2(a)
23. p. 350, Ex. 4(a)
24. p. 362, Ex. 4(a)
25. p. 369, Ex. 1(a)
26. p. 375, Ex. 2
27. p. 375, Ex. 5(a)
28. p. 201, Ex. 2(a) (closest pair of numbers in array)
29. p. 229, Ex. 2 (heap recognition)

Study questions on Topic 6 (Intractibility)

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) $O(n \lg n)$; (c) $O(n^2)$; (d) $\Omega(n \lg n)$; (e) $\Omega(n^2)$
5. Decision problems can also be considered as (a) formulas in propositional logic; (b) assertions; (c) array manipulations; (d) languages; (e) none of these
6. Exponential time is closely associated with (a) tractability; (b) combinatorial explosion; (c) constraint problems; (d) sorting problem; (e) interaction
7. 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
8. ϕ_P is (a) a problem; (b) an algorithm; (c) the function computed by program *P*; (d) the time function of program *P*; (e) none of these
9. $f(x)\downarrow$ means (a) $f(x)$ is descending as x rises; (b) $f(x)$ is unknown; (c) f is defined for parameter x; (d) f is undefined for parameter x; (e) none of these
10. $f(x)\uparrow$ means (a) $f(x)$ is descending as x rises; (b) $f(x)$ is unknown; (c) f is defined for parameter x; (d) f is undefined for parameter x; (e) none of these
11. A lower bound may (a) give the maximum running time for an algorithm; (b) give the average running time for an algorithm; (c) give the minimum running time for an algorithm; (d) give the minimum value output by an algorithm; (e) give a shorter version of an algorithm
12. The Hamiltonian-cycle problem is considered (a) intractable; (b) undecidable; (c) tractable; (d) polynomal-time; (e) polymorphic
13. 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
14. Deciding whether a formula in propositional logic is satisfiable is considered (a) intractable; (b) undecidable; (c) tractable; (d) decidable; (e) polymorphic
15. 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
16. 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
17. The Hamilton-cycle problem is (a) undecidable; (b) decidable in $O(n^2)$ time; (c) a vector problem; (d) a tree problem; (e) intractable
18. The set-partition problem is (a) undecidable; (b) decidable in $O(n^2)$ time; (c) a vector problem; (d) a tree problem; (e) intractable
19. The Traveling Salesperson problem is (a) undecidable; (b) decidable in $O(n^2)$ time; (c) a vector problem; (d) a tree problem; (e) intractable
20. The independent-set problem is (a) undecidable; (b) decidable in $O(n^2)$ time; (c) a vector problem; (d) a tree problem; (e) intractable
21. The 3SAT problem is (a) undecidable; (b) decidable in $O(n^2)$ time; (c) a vector problem; (d) a tree problem; (e) intractable
22. The channel-assignment 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. 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) none of these
2. *SPACE*($f(n)$) is a set of (a) functions on natural numbers; (b) time functions; (c) problems; (d) algorithms; (e) none of these
3. *TIME*($f(n)$) is a set of (a) functions on natural numbers; (b) time functions; (c) problems; (d) algorithms; (e) none of these
4. *DTIME* is a complexity class of (a) algorithms; (b) data structures; (c) problems; (d) theorems; (e) none of these
5. *EXPTIME* = *DTIME*(___) (a) 1; (b) n; (c) $\lg n$; (d) n^2; (e) none of these
6. The Cook-Karp Thesis states that the problem of deciding membership of strings in a language is tractable only if the language is decidable in (a) polynomial time; (b) $O(n)$; (c) $O(2^n)$; (d) $O(\lg n)$; (e) none of these
7. P is the set of (a) algorithms that execute in $O(n)$ time; (b) problems decidable in $O(n^k)$ time for some constant k; (c) problems *not* decidable in $O(n^k)$ time; (d) intractable problems; (e) exponential-time problems
8. NP is the set of (a) algorithms that execute in $O(n^k)$ time for some constant k; (b) problems decidable in $O(n^k)$ time; (c) problems for which possible solutions may be checked in $O(n^k)$ time; (d) intractable problems; (e) exponential-time problems
9. Problems for which no polynomial-time solutions are known are called (a) undecidable; (b) intractable; (c) NP; (d) optimization; (e) none of these
10. 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*

11. 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 tLonger answer questions
12. NPC is the set of *all* (a) algorithms that execute in $O(2^n)$ time; (b) problems decidable in $O(n^k)$ 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
13. Problems to which *SAT* or similar problems are *reducible* are called (a) *P*; (b) *NP*; (c) *NP*-complete; (d) *NP*-hard; (e) undecidable
14. *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
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
16. The set of intractable problems is associated with (a) P; (b) divide-and-conquer algorithms; (c) greedy algorithms; (d) NP; (e) NPC and EXPTIME

3. Approximate algorithms and iterative improvement

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
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
1. 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
2. The iterative-improvement solution to the maximal-flow problem uses (a) brute force; (b) divide and conquer; (c) greediness; (d) dynamic proramming; (e) approximation
3. Augmenting paths are used in (a) brute force; (b) divide and conquer; (c) iterative improvement; (d) dynamic programming; (e) probabilistic methods
4. 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
5. Iterative improvement starts with a(n) ____ solution (a) greedy; (b) uncomputable; (c) intractable; (d) feasible; (e) unacceptable
6. Iterative improvement uses (a) brute force; (b) divide and conquer; (c) greediness; (d) dynamic proramming; (e) approximation
7. 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
8. 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) none of these
9. 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) none of these
10. If a solution to the stable-marriage problem involves a series of proposals to preferred partners and acceptabnces if proposal is by a more preferred proposer, then it uses (a) transform and conquer; (b) greediness; (c) iterative improvement; (d) brute force; (e) none of these

4. State-space search

1. Generate-and-test methods explore (a) trees; (b) arrays; (c) graphs; (d) state spaces; (e) none of these
2. 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
3. The knapsack problem searches (a) a tree; (b) a graph; (c) an array; (d) a state space; (e) none of these
4. Iterated local search is (a) hill climbing; (b) $O(\lg n)$; (c) deterministic; (d) exact; (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 breadth-first search (a) uses a queue; (b) uses a stack; (c) searches an array; (d) searches a tree; (e) none of these
5. The depth-first search (a) uses a queue; (b) uses a stack; (c) searches an array; (d) searches a tree; (e) none of these
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. Bounded rationality pursues (a) optimality; (b) a global maximum; (c) satisficing; (d) speed; (e) none of these

9. Minimax is a(n) (a) inference rule; (b) heuristic; (c) form of knowledge representation ; (d) protocol; (e) none of these
10. Hill climbing is a(n) (a) problem; (b) heuristic strategy; (c) best-first search; (d) expression of consciousness; (e) form of representation
11. A drawback of hill climbing is (a) very long running time; (b) undecidability; (c) tendency to become stuck at local maxima; (d) the absence of goals; (e) none of these
12. Minimax is a (a) problem; (b) algorithm; (c) game; (d) neural-network design; (e) form of consciousness
13. The assumption that a game opponent will make the best possible move is made in (a) depth-first search; (b) breadth-first search; (c) all two-player games; (d) the minimax algorithm; (e) none of these
14. 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
15. A POMDP is a (a) stochastic differentiation problem; (b) stochastic decision problem; (c) deterministic process; (d) delayed payoff; (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) none of these
2. Evolutionary computation uses the technique of maximizing (a) fitness; (b) reward; (c) performance; (d) quantity of output; (e) none of these
3. Evolutionary computation (a) is deterministic; (b) seeks optimal solutions; (c) was developed in the 19th century; (d) is probabilistic; (e) none of these
4. Evolutionary computation is modeled on (a) brute force; (b) divide and conquer; (c) greediness; (d) natural selection; (e) fractals
5. Function optimization searches for (a) a function; (b) parameter values; (c) a return value; (d) an algorithm; (e) a time analysis
6. Fitness measures are (a) parameters to functions; (b) functions to be optimized; (c) return values; (d) algorithms; (e) time functions
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
9. 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

Longer-answer problems for topic 6

6a. Complexity classes

1. Define the two main complexity classes that distinguish tractable and intractable problems. Describe associated complexity classes.
2. What are 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. 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.
6. What bound on what resource is given by the minimum height of a decision tree representing execution of an algorithm?
7. Explain and compare complexities of TSP and Hamiltonian path.
8. Explain the *set-partition* problem and describe its complexity.
9. Explain the complexity of the problem of printing all subsets of a set, *A*.
10. Explain the complexity of the problem of printing all permutations of a sequence, *A*.
11. Compare *backtracking* to *depth-first search*.
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. If you think that the satisfiability problem for formulas in propositional logic is intractable, give a reason why.
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 ϕ in propositional logic?
    a. ϕ is a tautology
    b. ϕ is satisfiable
    c. ϕ is a contradiction
    d. ϕ holds for a given set of variable assignments
20. Write program code to generate a truth table that verifies the assertion that
    $p \lor q \lor r \lor s =$
    $(r \lor q \lor s) \land (r \lor \neg x \lor y) \land (s \lor \neg y \lor \neg x)$
    (*Note:* This will help establish that any CNF formula can be rewritten as a 3-CNF one.)
21. 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.
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. Describe the correspondence between a decision problem and a language.
24. What is DTIME($T(n)$)
25. What is EXPTIME?
26. What is P?
27. What is NP?
28. What is NPC?
29. What is the (very short) name of the set of problems that are decidable in time that is a polynomial function of the input size?
30. Give reasons why it makes sense to associate *P* with tractability, and give reasons why this association has limited value.
31. What complexity class (P, NP, NPC, NP-hard, EXP) are the following problems in?
    a. Searching an array
    b. Searching a graph
    c. Tree traversal
    d. Traveling salesperson
    e. Hamiltonian path
    f. SAT
    g. 3SAT
    h. Knapsack
    i. Independent set
    j. Sorting
    k. Minimum spanning tree
    l. Single-source shortest path

6b. Reducibility

1. If the SAT problem is reducible to problem π, then what can be said about problem π?
2. 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?
3. Suppose someone found an $O(n^4)$ 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?

4. Suppose someone found an $O(n^4)$ 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?
5. 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?
6. Consider the problem π of finding a path through a graph that passes through k or fewer vertices. Discuss the possible reducibility relationship of the following with π? Explain one possible case.
   a. Single-source shortest path
   b. Minimal spanning tree
   c. Search for a vertex in a graph
   d. SAT
   e. Convex hull
   f. Sorting
   g. Tree traversal
7. 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)
8. Explain which of the following may or may not be reducible to each other, and why.
   a. (SAT, TSP)
   b. (3SAT, SAT)
   c. (SAT, sorting)
   d. (sorting, reachability)
   e. (knapsack, array search)
   f. (shortest path, minimum spanning tree)
   g. (Hamiltonian path, TSP)
   h. (Hamiltonian path, graph coloring)
   i. (knapsack, independent set of vertices)
   j. (set partition, Hamiltonian path)
   k. (set partition, minimum spanning tree)

6c. Approximation and randomized algorithms

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. 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?
6. For what sorts of problems are approximation algorithms, randomization, and heuristics used? When are heuristics used?
7. Can the accuracy of heuristics be quantified?
8. Discuss the problem of finding a maximum matching in a bipartite graph, and one solution.
9. Why is the algorithm shown for approximately solving the stable-marriage problem $O(n^2)$?
10. What does searching state spaces have to do with solving problems?
11. When is the generate-and-test algorithm unlikely to find an optimal solution?
12. Why does hill climbing often find approximate rather than exact optima?
13. What is the complexity of minimax, relative to the number of moves ahead it looks in a game?
14. 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?
15. What sorts of problems does evolutionary computation address and how?
16. In your own words, relate fitness to function optimization.
17. What sort of algorithm uses *Select, Alter*, and *Evaluate* functions?
18. Describe a solution to the problem of linear-programming with Simplex method.
19. Describe a solution to the maximal-flow problem
20. How does game theory contribute to finding approximate solutions?
21. What is bounded rationality?

Study questions on Topic 7: Concurrency

1. Concurrent computing

1. Serial computing is the same as (a) concurrency; (b) parallelism; (c) distributed computing; (d) single-core; (e) none of these
2. Parallel computing is a kind of (a) concurrency; (b) Von Neumann architecture; (c) serial paradigm; (d) single-core computing; (e) none of these
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. Parallel computing may be inspired by (a) ants; (b) the human brain; (c) molecular processes; (d) the wavelike nature of light; (e) none of these
5. Parallel computing is (a) single core; (b) concurrency; (c) distributed computing; (d) intractable; (e) none of these
6. Distributed computing is (a) single core; (b) concurrency; (c) distributed computing; (d) intractable; (e) none of these

2. Modeling concurrency

1. 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
2. Process algebras assume communication by (a) packet; (b) indirect interaction; (c) bus; (d) message passing; (e) none of these
3. PRAM assumes that memory is (a) all distributed; (b) shared; (c) infinite; (d) magnetic; (e) none of these
4. Concurrent write is as opposed to ___ write (a) serial; (b) automatic; (c) shared; (d) exclusive; (e) none of these
5. The standard model for parallel computing is (a) CRCW; (b) RAM; (c) PRAM; (d) Von Neumann; (e) none of these
6. SIMD assumes that all CPUS execute ___ instructions (a) the same; (b) different; (c) pipelined; (d) cached; (e) none of these
7. SIMD assumes that all CPUs operate on ___ data (a) the same; (b) different; (c) pipelined; (d) cached; (e) none of these

3. Parallel programming and 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. When memory is distributed in parallel computing, global address space (a) does not exist; (b) is limited; (c) is available; (d) may be excessive; (e) none of these
3. 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
4. A parallel algorithm exists to add all the elements of an array in parallel time (a) $\Theta(1)$; (b) $\Theta(\lg n)$; (c) $\Theta(n)$; (d) $\Theta(n^2)$; (e) none of these
5. A parallel algorithm exists to assign the same value to all elements of an array in parallel time (a) $\Theta(1)$; (b) $\Theta(\lg n)$; (c) $\Theta(n)$; (d) $\Theta(n^2)$; (e) none of these
6. A parallel algorithm exists to search an array in parallel time (a) $\Theta(1)$; (b) $\Theta(\lg n)$; (c) $\Theta(n)$; (d) $\Theta(n^2)$; (e) none of these
7. A parallel algorithm exists to apply any associative operator to an array in parallel time (a) $\Theta(1)$; (b) $\Theta(\lg n)$; (c) $\Theta(n)$; (d) $\Theta(n^2)$; (e) none of these
8. Parallel prefix computation operates in parallel time (a) $\Theta(1)$; (b) $\Theta(\lg n)$; (c) $\Theta(n)$; (d) $\Theta(n^2)$; (e) none of these

4. 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

5. Performance of parallel and distributed systems

1. 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
2. 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
3. Problems considered tractable in parallel computing are in class (a) P; (b) NP; (c) NC; (d) NPC; (e) none of these
4. NC problems are ___ on parallel systems (a) log-time; (b) constant-time; (c) linear-time; (d) polylog-time; (e) none of these

5. The serial fraction is the _____ part of an algorithm (a) efficient; (b) parallelizable; (c) unparallelizable; (d) intractable; (e) none of these
6. 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
7. Quality of service must be maintained for (a) scalability; (b) tractability; (c) decidability; (d) correctness; (e) $O(n)$ time performance
8. 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
9. 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
10. Parallel work is ____ parallel time (a) less than; (b) equal to; (c) somewhat greater than; (d) much greater than; (e) none of these
11. If speedup is linear, a parallel system and algorithm are said to be (a) serial; (b) distributed; (c) scalable; (d) efficient; (e) quadrataic

5. 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

6. 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) none of these
6. Self-organization is associated with ____ behavior (a) deductive; (b) reactive; (c) planning; (d) emergent; (e) none of these
7. Emergent behavior is associated with (a) self-organization; (b) planning; (c) reflex agents; (d) stochastic reasoning; (e) none of these
8. Stigmergy makes use of (a) the environment; (b) first-order logic; (c) adaptive learning; (d) Bayesian reasoning; (e) none of these
9. Coordination via the environment is (a) minimax; (b) stigmergy; (c) centralized; (d) a heuristic; (e) none of these
10. Indirect interaction is in contrast to (a) deduction; (b) distributed AI; (c) stigmergy; (d) message passing; (e) none of these
11. Stigmergy uses (a) indirect interaction; (b) language; (c) reasoning; (d) a knowledge base; (e) none of these

12. Self-organization is observed in (a) English grammar; (b) garden design; (c) social insects; (d) proofs; (e) none of these
13. Indirect interaction features (a) anonymity; (b) synchronization; (c) use of predicate logic; (d) fuzzy reasoning; (e) none of these
14. Indirect interaction features (a) space decoupling; (b) synchronization; (c) use of predicate logic; (d) fuzzy reasoning; (e) none of these
15. Ant foraging by use of chemical trails is an example of (a) inference; (b) linguistic processing; (c) stigmergy; (d) evolution; (e) none of these
16. 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
17. 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
18. 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
19. The scalability of the _____ model is questioned (a) algorithmic; (b) sequential-interactive; (c) message-passing; (d) distributed; (e) parallel
20. Time efficiency of systems designs can be expressed as (a) processor speed; (b) parallelism; (c) scalability; (d) intractability; (e) ease of use
21. Intractable problems are (a) solvable with efficient algorithms; (b) scalable; (c) unscalable; (d) unsolvable; (e) none of these
22. Message passing models are considered (a) uncomputable; (b) intractable; (c) unscalable; (d) scalable; (e) none of these
23. If time performance degrades quickly as the number of computing agents rises, the system is (a) worthy; (b) incorrect; (c) unscalable; (d) synchronous; (e) efficient

Short and longer answer questions on topic 7

7a. Design parallel algorithm

1. Using parallel *for,* write a parallel algorithm to find, for an array of n integers:
   a. the sum
   b. the maximum value
   c. the OR, if the integers are all 0 or 1
   d. the AND
   e. the minimum value
   f. the number of 0's
   g. whether the array is sorted
   h. whether the array contains a given search key
2. Describe an efficient algorithm to send the same message to all nodes of a distributed system. (7a)
3. What term incorporates multi-tasking, multi-threading, networked computing, and parallel computing?
4. What are three theoretical models for concurrent computing?
5. (a) Describe a distributed algorithm to send the same message to all nodes of a distributed system organized in (ring, binary-tree) configuration. (7a)
   (b) Give its complexity and justify your answer. (7b)

7b. Parallel complexity

1. W.r.t. the previous questions, give complexity.

Miscellaneous

1. Describe parallel prefix computation
2. Describe the ring broadcast algorithm
3. Describe the leader-election algorithm
4. Define the *parallel fraction* and related terms.
5. What is Amdahl's Law?
6. Distinguish *work* from *parallel time*.
7. What are the two memory architectures for parallel computing?
8. Briefly state the common and distinguishing features of distributed computing, multi-tasking, and parallel processing.
9. Describe two forms of concurrency.
10. Describe two models of parallel computation.
11. What factors other than those considered in serial algorithm analysis must be considered in performance analysis of parallel and distributed systems?
12. Why is the following invalid?

```
y ← 0
for PID = 1 to n in parallel
    y ← y + A[PID]
return y
```

13. What is the standard model for parallel computation?
14. How does work relate to parallel time?
15. Discuss work, parallel time, speedup, and efficiency.
16. Give a definition of scalability in your own words.
17. What parallel algorithms apply associative operators efficiently, and in what parallel time?
18. Briefly state the common and distinguishing features of distributed computing, multi-tasking, and parallel processing.
19. Name two forms of concurrency.
20. Describe two models of parallel computation.
21. Write a parallel algorithm to compute the AND of an array of bits in time O(lg n).
22. 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.
23. 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.
15. Explain why the following pseudocode would fail:

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

16. Explain why the following pseudocode would fail:

```
Count-true (A)
count ← 0
forall processors PID ← 1 to |A|
    if A[PID] = true
        count ← count + 1
return count
```

17. What factors other than length of input affect the performanc eof parallel and distributed systems? (7c)
18. What sort of computation is associated with computing functions, and what sort of computation is associated with providing services? Give another distinction. (7c)
19. What performance issues are associated with types of computation that provide services? (7d)
20. What measures of efficiency are applied in interactive computing? (7d)
21. Distinguish two types of interaction by the number of active agents involved.
22. Distinguish two types of interaction by the use or non-use of the environment in interaction.
23. In multi-agent systems, what scalability issues are raised by the choice between message passing and shared access to storage?
24. Argue that interaction is more powerful than algorithms.
25. Argue that interaction is not more powerful than algorithms.

26. Argue that multi-stream interaction is more powerful than sequential interaction; that it is not more powerful.
27. Summarize Peter Wegner's view of reasoning versus interaction.
28. Discuss the time complexity of sequential interactive processes.
29. 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,
   a. Write an algorithm to solve the problem.
   b. Using a postcondition and a loop invariant, prove that your algorithm is correct.
   c. Use a recurrence to define the function computed by the algorithm.
   d. 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,
    e. Write an algorithm to *efficiently* solve the problem.
    f. Write a recurrence that defines the function computed by the algorithm
    g. 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?