Study questions

Note: These questions will form the basis for in-class and out-of-class problem solving, quizzes, and exams. To them are to be added new questions, including selected questions from Johnsonbaugh and Schaeffer.

Multiple topics

1. 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.)

2. 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.

Introduction

1. Use Euclid’s algorithm to find gcd(840, 180), showing your work.
2. Name common features of a set, a dictionary, and a list. Name features that distinguish the three.
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.
4. What are the domain and the range of the square-root function?

Multiple choice

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. 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
4. A language is a (a) string; (b) number; (c) set of numbers; (d) sequence of strings; (e) set of strings
5. $\subseteq$ denotes (a) set membership; (b) union; (c) conjunction; (d) a relation between sets; (e) negation
6. $\land$ 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) negation
8. $\cup$ denotes (a) set membership; (b) union; (c) AND; (d) a set; (e) negation
9. $\emptyset$ 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
Topic 1 (Problem classes)

1. Use the principle of optimality to describe finding the best batting average among players on teams in the major baseball leagues.
2. List two problems involving each of the following: arrays, graphs, sets of points on a coordinate graph.
3. Which of the following problems use solutions to which others, and in what way? String match, string search, pattern match
4. Distinguish a solution tree from a binary search tree.
5. Closest-pair and convex-hull are problems of what type?
6. Why is graph coloring an optimization problem?
7. What are the components of a graph that determine paths?
8. What kind of graph does the shortest-path problem apply to?
9. What is a subgraphs that is connected, acyclic, and provides a minimum-cost subway system design?
10. What kind of path passes through all vertices of a graph exactly once and returns to the start?
11. What problem finds the minimum-cost graph through all vertices of a graph?
12. What problem seeks the maximum-valued subset of weighted items totaling less than a given weight?
13. What problem involves grouping items so as to fit them in a set of containers of fixed size?
14. Satisfiability is a property of what sort of expression?
15. What is an ongoing series of responses to inputs, queries, or requests?
16. What sort of problem requires servicing multiple input streams, possibly under time constraints?
17. In what sense can one problem be said to be reducible to another?

Multiple choice

1. Minimum spanning tree is an attribute of (a) arrays; (b) weighted graphs; (c) unweighted graphs; (d) sets of points; (e) none of these
2. Convex hull is an attribute of (a) arrays; (b) weighted graphs; (c) unweighted graphs; (d) sets of points; (e) none of these
3. Closest pair is an attribute of (a) arrays; (b) weighted graphs; (c) unweighted graphs; (d) sets of points; (e) none of these
4. Which of these is an optimization problem? (a) reachability; (b) traversal; (c) sort; (d) string search; (e) knapsack
5. Web pages in general (a) compute functions; (b) do sorts; (c) provide services; (d) compute traversals; (e) none of these
6. An example of an order statistic is (a) average; (b) mode; (c) standard deviation; (d) median; (e) none of these
7. A topological sort is applied to (a) graph vertices; (b) an array; (c) a tree; (d) a logical formula; (e) none of these
8. (a) ; (b) ; (c) ; (d) ; (e)
9. (a) ; (b) ; (c) ; (d) ; (e)
10. (a) ; (b) ; (c) ; (d) ; (e)
Topic 2 (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?

2. What are three classes of comments that help establish that the spec of a procedure is satisfied? For each, state where the comment should appear in the code or pseudocode.

3. What is the name of a particular mathematical technique or notation for defining a function, that converts straightforwardly into code or pseudocode?

4. By use of preconditions, postconditions, and loop invariants, prove or argue persuasively that the pseudocode below will correctly count and return the number of spaces in a string, s. Note that a loop invariant is an assertion about values of data items, not a narrative of what happens.

   Count-spaces(s)
   \( n \leftarrow 0 \)
   \( i \leftarrow 1 \)
   while \( i \leq \text{length}(s) \) do
     if \( s[i] = ' ' \) then
       \( n \leftarrow n + 1 \)
     \( i \leftarrow i + 1 \)
   return \( n \)

Relate the above algorithm to the Hoare triple \( \langle \phi \rangle P(\psi) \) and explain how you would go about proving the correctness of Count-spaces.

5. Consider the pseudocode below:

   Product (x, y)
   \( \text{result} \leftarrow 0 \)
   for \( i \leftarrow 1 \) to \( x \)
     \( \text{result} \leftarrow \text{result} + y \)
   return \( \text{result} \)

   For this pseudocode, (a) what are an appropriate precondition, postcondition, and loop invariant? (b) How does your loop invariant ensure that the postcondition is valid?

6. Write an algorithm to compute the function \( y = a^x \) for inputs \( a, b \). For this algorithm, answer (a), (b) of Question 21.

7. Write an algorithm to compute the sum of the elements of an input array, \( A \). For this algorithm, answer (a), (b) of Question 21.

10. Write an algorithm to compute the function \( y = x! \) (x factorial), for input \( x \). For this algorithm, answer (a), (b) of Question 21.

12. 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.

13. Show that the following code reverses an array \( A \), where \( \text{first} \) and \( \text{last} \) are subscripts:

   Reverse \( (A, \text{first}, \text{last}) \)
   If \( \text{first} < \text{last} \) then
     swap \( (A[\text{first}], A[\text{last}]) \)
     Reverse \( (A, \text{first}+1, \text{last}-1) \)

14. Write an algorithm that tells whether an array of integers is in ascending order; prove correctness.

15. Write preconditions, postconditions, and loop invariants to argue that Which-sort, below, works correctly. This requires showing that Largest-to-right moves the largest element of \( A[1..n] \) into position \( n \) of \( A \).

   Which-sort (A)
   for \( i \leftarrow \text{size}(A) \) down to 2 do
     Largest-to-right \( (A, i) \)
   Largest-to-right \( (A, n) \)
   largest \( \leftarrow A[1] \)
   for \( i \leftarrow 2 \) to \( n \) do
     if \( A[i] > A[\text{largest}] \) then
       largest \( \leftarrow i \)
     swap \( A[\text{largest}] \) with \( A[n] \)

16. In Computation Tree Logic (CTL), express the assertion that no path exists in which \( \phi \) will never hold. (Practical example: In an operating system, no process is permanently blocked from access to a requested resource.)

17. What verification method is commonly used for reactive systems?

18. What sort of logic is used in model checking?

19. Name the two quantifiers used in predicate logic and the corresponding ones used in CTL.

20. [See problems from Hamburger-Richards]
Multiple choice

1. Which are 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. 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

3. Valid assertions (a) can help establish that code is to spec; (b) guarantee correctness;
   (c) guarantee that a program halts; (d) are always coded; (e) all the above

4. A precondition (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

5. 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

6. 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

7. 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

8. Loop invariants are used in correctness proofs that are (a) by contradiction; (b) inductive;
   (c) diagonal; (d) constructive; (e) none of these

9. 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

10. 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

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

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

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

14. 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

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

16. Which of these is not used to prove correctness of reactive systems? (a) Hoare triples; (b) CTL;
   (c) model checking; (d) Kripke structures; (e) all are used
Topic 3 (Recurrences)

1. In mathematical notation, list some of the most widely discussed classes of algorithmic complexity.
2. 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?
3. What is a lower bound? A tight bound? Distinguish big-O, \( \Omega \), and \( \theta \), and relate to algorithm analysis.
4. For function \( f \), \( \mathcal{O}(f) \), \( \Omega(f) \), and \( \theta(f) \) are sets of functions. What are the set containment relationships among the following:
   (a) \( \mathcal{O}(n^2), \mathcal{O}(\log n) \)
   (b) \( \Omega(n^2), \Omega(\log n) \)
   (c) \( \Theta(n^2), \Theta(\log n) \)
5. Express in big-O notation \( T(n) = 500n + n^5 / 10 + 80 \log n \)
6. Give an example of a lower bound on the function \( f(x) = x^3 \)
7. Explain why \( \theta(f) \) called a tight bound on \( f \).
8. What is the main notation for expressing the complexity of algorithms as lower bounds, and what is the intuitive meaning of this notation?
9. What is the main notation for expressing the complexity of algorithms as upper bounds, and what is the intuitive meaning of this notation?
10. Give the names of five asymptotic efficiency classes, in ascending order of time complexity, and group the following time functions by efficiency class:
    (a) \( t(n) = 6 \log n \)
    (b) \( t(n) = 5 n^3 \)
    (c) \( t(n) = n / 4 \)
    (d) \( t(n) = 2^n \)
    (e) \( t(n) = 10 n \log n + 100 \)
    (f) \( t(n) = 80n + 1000 \)
11. Use two simple formulas to show the relation among \( \mathcal{O}, \Theta, \) and \( \Omega \) notations
    (Hint: if \( f(n) \in \Omega(g(n)) \), what follows?)
12. What is the name of a particular mathematical technique or notation for defining a function, that converts straightforwardly into code or pseudocode?
13. What is the complexity of the problem of finding the best available move in Minesweeper? Show your work. The rules of the game specify that the player loses if player clicks on a mine; numbers in cells tell how many mines are in any of the 8 cells around the cell. Best move is either to find a mine and flag it by right-clicking, or to left-click a cell known to be a non-mine. Left-clicking a non-mine reveals a number.
14. (a) Write the strlen subprogram in iterative pseudocode or a programming language. \( \text{strlen} \) takes an address as a parameter and returns the length of the string (distance from a null character).
   (b) Use verification tools to prove or show its correctness.
   (c) Express the algorithm as a recurrence.
   (d) Based on (c), write a recurrence to express the time function of the algorithm.
   (e) Solve the recurrence.
15. What is worst-case time for binary search, and why?
16. (a) Express an algorithm iteratively for the factorial function (example: \( \text{fact}(4) = 4 \times 3 \times 2 = 24 \) )
(b) By use of comments, show that it terminates;
(c) Write a recurrence to define the factorial function (you may wish to write a recursive version in pseudocode first)
(d) Based on (c), write a recurrence to define the time function that describes the complexity of the algorithm;
(e) Solve the recurrence.

17. a. Express an algorithm iteratively that takes as a parameter a bit vector and returns the numeric value of the sequence
b. Write a loop invariant that will help prove its correctness.
c. Write a recurrence to define the same function (you may wish to write a recursive version in pseudocode first);
d. Based on (c), write a recurrence to define the time function that describes the complexity of the algorithm.
e. Solve the recurrence, to yield the time analysis of the algorithm in big-O notation, justifying your answer.

18. a. Express an algorithm \textit{iteratively} that takes as a parameter a sequence of integers and returns the smallest value in the sequence
b. Write a recurrence to define the same function (you may wish to write a recursive version in pseudocode first);
c. Based on (b), write a recurrence to define the time function that describes the complexity of the algorithm.
d. Solve the recurrence, to yield the time analysis of the algorithm in big-O notation, justifying your answer.

19. Discuss the complexity of this search algorithm, write it as a recurrence, and argue for its correctness, using preconditions, postconditions, and a loop invariant:
\begin{verbatim}
Some-search (A, first, last, key)
while first \leq last do
    middle \leftarrow (first + last) / 2
    if key < A[middle]
        last \leftarrow middle - 1
    else if key > A[middle]
        first \leftarrow middle + 1
    else return true
return false
\end{verbatim}

e. Solve the recurrence.

20. State complexity of each algorithm below and justify your answer.
\begin{verbatim}
Which-sort (A)
for i \leftarrow \text{size}(A) \text{ down to } 2 \text{ do }
    Largest-to-right (A, i)
\end{verbatim}
\begin{verbatim}
Largest-to-right (A, n)
largest \leftarrow A[1]
for i \leftarrow 2 \text{ to } n \text{ do }
    if A[ i ] > A[\text{largest}]
        largest \leftarrow i
    swap A[\text{largest}] with A[n]
\end{verbatim}

21. (a) Write an iterative version of an algorithm to find the number of occurrences of a string, \( s \), in a text file, \( t \).
(b) Use verification methods discussed in class to prove correctness.
(c) Express the algorithm as a recurrence
(d) Express the time complexity of the algorithm as a recurrence
(e) Solve the recurrence, stating the complexity of the algorithm.

22. Consider the problem of finding the largest element of a subsequence in an array. Signature: \( \text{Max}(A, \text{first}, \text{last}) \), where \( A \) is an array, \( \text{first} \) and \( \text{last} \) are subscripts. Algorithm should return the largest element of the sequence from \( A[\text{first}] \) to \( A[\text{last}] \), inclusive.
   a. Express an algorithm \textit{iteratively} (in pseudocode or a programming language) to solve this problem. Use verification methods to show correctness.
b. Write a \textit{recursive} version of this algorithm.
c. Write a recurrence that defines the function computed.
d. Write a recurrence, based on (c), that defines the time function, i.e., the function on the array
sequence size \( n = \text{last} - \text{first} + 1 \) that returns the running time.
e. Solve the recurrence, giving a one-sentence explanation.
23. Levitin, pp. 59-60, Ex. 1, 2(b, c), 3 (b, c, d), 5, 8(a). Show your work.
24. p. 67, Ex. 1 (g) (summations; show work), 2 (d) (summation; show work), 4 (mystery algorithm)
25. p. 76, Ex. 1 (b, c) (solve recurrences), 26. p. 76, Ex. 3 (solve recurrences)

**Multiple choice**

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. Linear search is (a) \( O(1) \); (b) \( O(\lg n) \); (c) \( O(n) \);
   (d) \( O(n^2) \); (e) \( O(2^n) \)
5. Bubble sort is (a) \( O(1) \); (b) \( O(\lg n) \); (c) \( O(n) \);
   (d) \( O(n^2) \); (e) \( O(2^n) \)
6. Insertion sort is (a) \( O(1) \); (b) \( O(\lg n) \); (c) \( O(n) \);
   (d) \( O(n^2) \); (e) \( O(2^n) \)
7. Quicksort is (a) \( O(1) \); (b) \( O(\lg n) \); (c) \( O(n) \);
   (d) \( O(n^2) \); (e) \( O(2^n) \)
8. Function \( g \) is an upper bound on function \( f \) iff for all \( x \),
   (a) \( g(x) \leq f(x) \); (b) \( g(x) \geq f(x) \);
   (c) \( g = O(f) \); (d) \( f = \Omega(g) \); (e) none of these
9. Function \( g \) is a lower bound on function \( f \) iff for all \( x \),
   (a) \( g(x) \leq f(x) \); (b) \( g(x) \geq f(x) \);
   (c) \( f = O(g) \); (d) \( g = \Omega(f) \); (e) none of these
10. Quadratic time is faster than (a) \( O(1) \);
    (b) \( O(\lg n) \); (c) \( O(n^2) \); (d) \( O(n^3) \); (e) none of these
11. Big-Omega notation expresses (a) tight bounds;
    (b) upper bounds; (c) lower bounds; (d) worst cases; (e) none of these
12. Big-O notation expresses (a) tight bounds;
    (b) upper bounds; (c) lower bounds; (d) best cases; (e) none of these
13. Theta notation expresses (a) tight bounds;
    (b) upper bounds; (c) lower bounds; (d) worst cases; (e) none of these
14. Empirical analysis of algorithms can be done using (a) theorems only;
    (b) system clock; (c) loop invariants; (d) reductions; (e) none of these
15. Empirical analysis of algorithms can be done using (a) theorems only;
    (b) counter variables; (c) loop invariants; (d) reductions; (e) none of these
16. 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
17. When 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) \( O(n^2) \)
18. 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) \( O(n^2) \)
19. When base case is \( O(n) \) 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) \( O(n^2) \)
20. 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) \( O(n^2) \)
21. (a) ; (b) ; (c) ; (d) ; (e)
22. (a) ; (b) ; (c) ; (d) ; (e)
Topic 4 (Brute force)

1. Write the brute-force algorithm for string-matching (string searching) 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.
   (b) Use verification tools to prove or show its correctness.
   (c) Express the algorithm as a recurrence or set of recurrences.
   (d) Based on (c), write a recurrence to express the time function of the algorithm.
   (e) Solve the recurrence.
2. Name a brute-force algorithm for sorting an array, and one for searching an array, and state why they may be considered brute-force.
3. Write a brute-force algorithm to solve the closest-pair problem. Analyze.
4. Write a brute-force algorithm to solve the convex-hull problem. Analyze.
5. Write a brute-force algorithm to find the minimum spanning tree of a graph. Analyze.
6. Write a brute-force algorithm to solve the satisfiability problem. Analyze.
7. Write a brute-force algorithm to find a Hamiltonian path. Analyze.
8. Write a brute-force algorithm to solve the Traveling Salesperson problem. Analyze.
9. Write a brute-force algorithm to solve the knapsack problem. Analyze.
10. Write a brute-force algorithm to tell whether a given destination vertex in a graph is reachable from a given source vertex. Analyze.
11. Write a brute-force algorithm to solve the independent-set problem. Analyze.
12. Write a brute-force algorithm to determine whether a given set of natural numbers may be partitioned into two subsets of equal sum. Analyze.

Multiple choice

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. The insertion sort is what kind of algorithm?
   (a) divide and conquer; (b) greedy;
   (c) brute force; (d) dynamic programming;
   (e) probabilistic
3. The linear search is what kind of algorithm?
   (a) divide and conquer; (b) greedy;
   (c) brute force; (d) dynamic programming;
   (e) probabilistic
4. The Bubble sort is what kind of algorithm?
   (a) divide and conquer; (b) greedy;
   (c) brute force; (d) dynamic programming;
   (e) probabilistic
5. The simpler version of the closest-pair problem discussed in this course is (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic
6. Exhaustive search is what kind of algorithm?
   (a) divide and conquer; (b) greedy;
   (c) brute force; (d) dynamic programming;
   (e) probabilistic
7. (a) ; (b) ; (c) ; (d) ; (e)
1. In divide and conquer, what gets divided?

2. Discuss faster-than-brute-force approach to the convex-hull problem, including the complexity of the algorithm.

3. Describe the merge sort.

4. Describe the merge algorithm. What are its preconditions and postconditions?

5. Distinguish the breadth-first and depth-first searches. What do they search in and what do they search for?

6. Relate the notions of order statistics to the notion of median.

7. How does the divide-and-conquer algorithm for order statistics use Partition to improve on the efficiency of brute-force order statistic?

8. 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

9. Discuss a faster-than-brute-force approach to the closest-pair problem, including the complexity of the algorithm.

10. What is worst-case time for binary search, and why?

11. Write an algorithm that determines whether a given binary tree has the BST property. Analyze.

12. Write an algorithm that determines whether a given binary tree has the heap property. Analyze.

13. Levitin, p. 128, Ex. 1 (array maximum)

14. p. 139, Ex. 6 (binary search for elements in a range)

15. p. 143, Ex. 2 (number of leaves in a binary tree),

16. p. 143, Ex. 8 (internal path length in binary tree)

17. p. 154, Ex. 3 (closest pair of points on plane)
   – pseudocode is OK

18. p. 170-171, Ex. 1 (depth-first search of graph)

**Multiple choice**

1. 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

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

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

4. The BST search uses which approach to algorithm design? (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 convex hull and closest-pair problems is (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic

6. 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

7. 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

8. The breadth-first search (a) uses a queue; (b) uses a stack; (c) searches an array; (d) searches a tree; (e) none of these

9. The depth-first search (a) uses a queue; (b) uses a stack; (c) searches an array; (d) searches a tree; (e) none of these

10. (a) ; (b) ; (c) ; (d) ; (e)
11. (a); (b); (c); (d); (e)
Topic 6 (Greedy algorithms)

1. What algorithm finds the single-source shortest path in a graph?
2. What two algorithms find minimal spanning trees in graphs?
3. Give a lower time bound for the minimal spanning tree problem.
4. 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?
5. Compare pseudocode in the slides and the textbook for Dijkstra’s algorithm.
6. What is the running time of the Dijkstra algorithm and why?
7. What does Huffman’s algorithm find?
8. Describe a greedy solution to the continuous knapsack problem.
9. Defend or refute: The optimal-substructure property guarantees that some greedy algorithm solves a given problem.
10. Design an algorithm to tell whether a graph is cyclic; write a recurrence to define the function computed; write a time recurrence; state complexity.
12. Show that minimal spanning tree (Prim, Kruskal) is not necessarily the same as tree of single-source shortest paths (Dijkstra).
13. Show how optimal-substructure property applies with each greedy algorithm discussed.
14. Show how optimal-substructure property is used in continuous-knapsack problem solution.
15. Levitin, p. 314, Ex. 7 (Apply Prim’s algorithm to pictured graph)
16. P. 314, Ex. 8, 9, 10, 11
17. p. 322, Ex. 10 (Compare efficiencies of Prim and Kruskal algorithms experimentally)

Multiple choice

1. 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
2. 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
3. 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
4. The optimal substructure property asserts that (a) if S solves an optimization problem, then any component of S is an optimal solution to a subproblem; (b) if some component of S is an optimal solution to a subproblem, then S solves the larger problem; (c) all parts of a problem have suboptimal solutions; (d) the sum of the parts is greater than the whole; (e) none of these
5. Greedy solutions exist for (a) no problems; (b) no optimization problems; (c) some problems; (d) all problems; (e) none of these
6. The single-source shortest path problem has a good 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 optimal is an instance of which property? (a) excluded middle; (b) reduction; (c) optimal substructure; (d) rationality; (e) order statistic
8. Huffman’s algorithm is (a) divide and conquer; (b) greedy; (c) brute force; (d) dynamic programming; (e) probabilistic
9. (a) ; (b) ; (c) ; (d) ; (e)
10. (a) ; (b) ; (c) ; (d) ; (e)
Topic 7 (Dynamic programming)

1. 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.
2. What are the space and time considerations raised by B trees?
3. Compare the time complexities of double-recursive Fibonacci and Fibonacci with dynamic programming.
4. Describe the binomial-coefficient problem.
5. What is the transitive closure of the adjacency relation?
6. How does Warshall’s algorithm compute the reachability matrix of a graph?
7. 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?
8. See slide 10 (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?
9. Describe an algorithm to find the mode (most frequent element) of an array. Analyze.
10. What are AVL trees used for?
11. What are the running times of heapify, extract-min, and insert for heaps? Relate this to the data structure used.
12. In dynamic-programming algorithms, what resource defines the main overhead of using this method? Write algorithm to solve sorting-by-counting problem (see slide).
14. Write algorithm to decide path of descent of B tree.
15. Find complexities of $C(n, k)$ binomial coefficient algorithms that use recursion (brute force) and dynamic programming.
16. Levitin, p. 271, Ex. 7
17. p. 276, Ex. 3
18. p. 283, Ex. 4(a)
19. p. 292, Ex. 1 (See D. Keil C++ code, available in course docs)
20. p. 303, Ex. 2(a)
21. p. 350, Ex. 4(a)
22. p. 362, Ex. 4(a)
23. p. 369, Ex. 1(a)
24. p. 375, Ex. 2
25. p. 375, Ex. 5(a)
26. p. 201, Ex. 2(a) (closest pair of numbers in array)
27. p. 229, Ex. 2 (heap recognition)

Multiple choice

1. 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
2. Dynamic programming solutions are associated with (a) space-time tradeoffs; (b) brute force; (c) intractability; (d) probabilistic algorithms; (e) approximation
3. Hashing solutions is associated with (a) space-time tradeoffs; (b) brute force; (c) intractability; (d) probabilistic algorithms; (e) approximation
4. Warshall’s algorithm for finding reachability matrix is associated with (a) dynamic programming; (b) brute force; (c) intractability; (d) probabilistic algorithms; (e) approximation
5. 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
6. The running time for heap-extract-min is (a) $O(1)$; (b) $O(lg n)$; (c) $O(n lg n)$; (d) $O(n)$; (e) $O(n^2)$
7. The running time for heapify is (a) $O(1)$; (b) $O(lg n)$; (c) $O(n lg n)$; (d) $O(n)$; (e) $O(n^2)$
8. The running time for heap-sort is (a) $O(1)$; (b) $O(lg n)$; (c) $O(n lg n)$; (d) $O(n)$; (e) $O(n^2)$
9. The running time for \textit{build-heap} is (a) \(O(1)\); (b) \(O(lg\ n)\); (c) \(O(n\ lg\ n)\); (d) \(O(n)\); (e) \(O(n^2)\)

10. The AVL tree is an instance of (a) dynamic programming; (b) brute force; (c) intractability; (d) probabilistic algorithms; (e) transform-and-conquer


**Topic 8 (Complexity)**

1. 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”.
2. What bound on what resource is given by the minimum height of a decision tree representing execution of an algorithm?
3. What common-sense evidence is there that the Hamiltonian Cycle problem is very hard?
4. Define the two main complexity classes that distinguish tractable and intractable problems.
5. What sorts of running times is intractability associated with?
6. If you think that the satisfiability problem for formulas in propositional logic is intractable, give a reason why.
7. Concerning the following formula in propositional logic
   \[(p \lor q \lor r) \land (p \lor \neg q \lor r) \land (\neg p \lor q \lor \neg r)\]
   (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.
8. What are the complexities of these problems w.r.t. formulas \( \phi \) in propositional logic?
   a. \( \phi \) is a tautology
   b. \( \phi \) is satisfiable
   c. \( \phi \) is a contradiction
   d. \( \phi \) holds for a given set of variable assignments
9. Write program code to generate a truth table that verifies the assertion that
   \[p \lor q \lor r \lor s = (r \lor q \lor x) \land (r \lor \neg x \lor y) \land (s \lor \neg y \lor \neg x)\]
   (Note: This will help establish that any CNF formula can be rewritten as a 3-CNF one.)
10. 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 \( B \)?
11. Describe the correspondence between a decision problem and a language.
12. What is DTIME(\( T(n) \))
13. What is EXPTIME?
14. What is \( P \)?
15. What is \( NP \)?
16. What is \( NPC \)?
17. 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?
18. 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?
   (b) What implications would it have for other problems to which \( TSP \) is reducible in polynomial time?
19. Give reasons why it makes sense to associate \( P \) with tractability, and give reasons why this association has limited value.
20. Levitin, p. 385, Ex. 3(a), 3(c) (Lower bound classes for maximum and subset problems)
21. p. 393, Ex. 9(a)-(b) (Tournament tree, number of games and rounds)
22. p. 402, Ex. 7(a) (State the decision version of knapsack problem)
23. p. 423, Ex. 3(a) (Backtracking for n-queens problem; pseudocode plus statement of rules is OK)
24. p. 432, Ex. 1 (Data structure to keep track of live nodes in best-first branch-and-bound algorithm)

**Multiple choice**

1. Uncomputable functions correspond to problems that are called (a) undecidable; (b) intractable; (c) P-time; (d) optimization; (e) none of these
2. Problems for which no polynomial-time solutions are known are called (a) undecidable; (b) intractable; (c) NP; (d) optimization; (e) none of these
3. The Halting Problem is (a) undecidable; (b) intractable; (c) NP-complete; (d) optimization; (e) none of these
4. 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
5. The problem of sorting by comparisons is
(a) $O(n)$; (b) $O(n \log n)$; (c) $O(n^2)$; (d) $\Omega(n \log n)$; (e) $\Omega(n^2)$
6. The Hamiltonian-cycle problem is considered
(a) intractable; (b) undecidable; (c) tractable; (d) decidable; (e) polymorphic
7. The problem of evaluating a formula in propositional logic is (a) intractable;
(b) undecidable; (c) tractable; (d) decidable; (e) polymorphic
8. Deciding whether a formula in propositional logic is satisfiable is (a) intractable;
(b) undecidable; (c) tractable; (d) decidable; (e) polymorphic
9. If checking a given candidate solution to a problem is easy enough, then the problem is in
(a) P; (b) NP; (c) NPC; (d) EXP; (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 these
12. 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
13. Decision problems are equivalent to
(a) formulas in propositional logic; (b) assertions; (c) array manipulations;
(d) languages; (e) none of these
14. $SPACE(f(n))$ is a set of (a) functions on natural numbers; (b) time functions; (c) languages;
(d) algorithms; (e) none of these
15. $TIME(f(n))$ is a set of (a) functions on natural numbers; (b) time functions; (c) languages;
(d) algorithms; (e) none of these
16. $DTIME$ is a complexity class of (a) algorithms; (b) data structures; (c) problems; (d) theorems;
(e) none of these
17. $EXPTIME = DTIME(\_\_\_\_\_)$ (a) 1; (b) $n$; (c) $\log n$; (d) $n^2$; (e) none of these
18. 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) $O(\log n)$; (e) none of these
19. $P$ 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
20. $NP$ 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
21. $NPC$ is a set of (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
22. Problems to which SAT or similar problems are reducible are called (a) P; (b) NP; (c) NP-
complete; (d) NP-hard; (e) undecidable
23. NP-complete problems are 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
Topic 9 (Approximation)

1. When are heuristics used?
2. Can the accuracy of heuristics be quantified?
3. For what sorts of problems are approximation algorithms, randomization, and heuristics used?
4. Discuss the problem of finding a maximum matching in a bipartite graph, and one solution.
5. Why is the algorithm shown for approximately solving the stable-marriage problem $O(n^2)$?
6. What does searching state spaces have to do with solving problems?
7. When is the generate-and-test algorithm unlikely to find an optimal solution?
8. When is the exhaustive search for a path through a tree likely to take a huge amount of time?
9. Why does hill climbing often find approximate rather than exact optima?
11. What is the complexity of minimax, relative to the number of moves ahead it looks in a game?
12. 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?
13. What is a reasonable alternative to the goal of optimality?
14. In what environments do POMDPs occur?
15. What sorts of problems does evolutionary computation address?
16. In your own words, relate fitness to function optimization.
17. What sort of algorithm uses Select, Alter, and Evaluate functions?
18. What is a probabilistic method for computing the approximations of functions, inspired by a phenomenon found in nature?
19. Describe how you might write a program that writes programs by evolutionary computation; e.g., your program is given as input a postcondition that specifies program $P$’s output as a function of input, and your program will output program $P$.

Multiple choice

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. 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
5. Finding a feasible solution to an optimization problem under constraints, and refining it by successive steps, is (a) brute force; (b) dynamic programming; (c) iterative improvement; (d) intractable; (e) none of these
6. For ______ problems, sometimes global maxima/minima differ from local ones (a) optimization; (b) $O(n)$; (c) BST search; (d) sorting; (e) none of these
7. Generate-and-test methods explore (a) trees; (b) arrays; (c) graphs; (d) state spaces; (e) none of these
8. 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
9. The knapsack problem searches (a) a tree; (b) a graph; (c) an array; (d) a state space; (e) none of these
10. Iterated local search is (a) hill climbing; (b) $O(lg n)$; (c) deterministic; (d) exact; (e) none of these
11. The minimax algorithm emerged from (a) recursive function theory; (b) calculus; (c) game theory; (d) symbolic logic; (e) none of these
12. 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.

13. Bounded rationality pursues (a) optimality; (b) a global maximum; (c) satisficing; (d) speed; (e) none of these.

14. The Markov property holds for systems whose next state depends on (a) past history; (b) accumulated value; (c) random chance; (d) current state; (e) none of these.

15. Evolutionary computation seeks to maximize (a) fitness; (b) reward; (c) performance; (d) quantity of output; (e) none of these.

16. Evolutionary computation (a) is deterministic; (b) seeks optimal solutions; (c) was developed in the 19th century; (d) is probabilistic; (e) none of these.

17. (a); (b); (c); (d); (e)
1. What term incorporates multi-tasking, multi-threading, networked computing, and parallel computing?
2. What are three theoretical models for concurrent computing?
3. What is the standard model for parallel computation?
4. How does work relate to parallel time?
5. Discuss work, parallel time, speedup, and efficiency.
6. Give a definition of scalability in your own words.
7. Describe the ring broadcast algorithm.
8. What are the two memory architectures for parallel computing?
9. What are three processor architectures for parallel or distributed computing?
10. Write a parallel algorithm to find, for an array of 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
11. What is the complexity of the algorithm for #12(a)?
12. What parallel algorithms apply associative operators efficiently, and in what parallel time?
13. Briefly state the common and distinguishing features of distributed computing, multi-tasking, and parallel processing.
14. Name two forms of concurrency.
15. Describe two models of parallel computation.
16. Write a parallel algorithm to compute the AND of an array of bits in time $O(\log n)$.
17. 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.
18. 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.
19. What factors other than those considered in serial algorithm analysis must be considered in performance analysis of parallel and distributed systems?

Multiple choice
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. 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
5. Process algebras assume communication by
   (a) packet; (b) indirect interaction; (c) bus; (d) message passing; (e) none of these
6. PRAM assumes that memory is (a) all distributed; (b) shared; (c) infinite; (d) magnetic; (e) none of these
7. Concurrent write is as opposed to ___ write
   (a) serial; (b) automatic; (c) shared; (d) exclusive; (e) none of these
8. The standard model for parallel computing is
   (a) CRCW; (b) RAM; (c) PRAM; (d) Von Neumann; (e) none of these
9. In parallel computing, work is greater than
   (a) serial time; (b) parallel time; (c) amount of data; (d) best-case time; (e) none of these
10. SIMD assumes that all CPUS execute ___ instructions
    (a) the same; (b) different; (c) pipelined; (d) cached; (e) none of these
11. SIMD assumes that all CPUS operate on ___ data
    (a) the same; (b) different; (c) pipelined; (d) cached; (e) none of these
12. 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

Topic 10 (Parallel and distributed algorithms)
13. 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

14. 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

15. A parallel algorithm exists to add an array in parallel time (a) $\Theta(1)$; (b) $\Theta(lg\ n)$; (c) $\Theta(n)$; (d) $\Theta(n^2)$; (e) none of these

16. 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

17. 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

18. 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

19. 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

20. Parallel speedup is the (a) parallel time; (b) parallel work; (c) rate of increase in work done as number of processors rises; (d) efficiency of an algorithm; (e) none of these

21. Problems considered tractable in parallel computing are in class (a) P; (b) NP; (c) NC; (d) NPC; (e) none of these

22. NC problems are (a) log-time; (b) constant-time; (c) linear-time; (d) polylog-time; (e) none of these

23. The serial fraction is the (a) efficient; (b) parallelizable; (c) unparallelizable; (d) intractable; (e) none of these

24. 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

25. 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

26. (a) ; (b) ; (c) ; (d) ; (e)
Topic 11 (Interactive computation)

1. What sort of computation is associated with computing functions, and what sort of computation is associated with providing services?
2. Distinguish two types of interaction by the number of active agents involved.
3. Distinguish two types of interaction by the use or non-use of the environment in interaction.
4. Whereas algorithmic computation operates on strings and natural numbers, what does sequential interaction operate on?
5. What measures of efficiency are applied in interactive computing?
6. In multi-agent systems, what scalability issues are raised by the choice between message passing and shared access to storage?
7. Argue that interaction is more powerful than algorithms.
8. Argue that interaction is not more powerful than algorithms.
9. Argue that multi-stream interaction is more powerful than sequential interaction; that it is not more powerful.

Multiple choice

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) dynamic; (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 active partner in (a) computation (b) 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. 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
16. UML is required to specify (a) functional problems; (b) services; (c) algorithms; (d) algorithmic inputs; (e) none of these
17. Mutual causality characterizes (a) algorithms; (b) only directly-interacting entities; (c) all interacting entities; (d) functions; (e) none of these
18. 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

19. (a); (b); (c); (d); (e)

20. (a); (b); (c); (d); (e)
Homework 1 (Introduction and Problem classes)
Due 9/17

1. (60%) Go to the course Discussion Board at http://framingham.blackboard.com, Topic 1, and briefly describe your background and expectations coming into this course.

2. (40%) Comment on another person’s posting.