Note:

For some questions, answers are provided. These answers need to be checked to see that they match the questions, number for number.

Some other checking and editing is needed, including deletion of questions that are trivial or based on an expectation of too-detailed knowledge.

The objective of this set of questions is to help students focus on the main facts, terms, and ideas that underly the course material.

Study questions

Study questions on topic 1:
Data types: Numeric, array, and class

1. Collections and arrays

A. Multiple-choice, T/F

1. An abstract data type is (a) a compound type and its associated operations; (b) a simple type and associated operations; (c) a compound type independent of operations; (d) a simple type independent of operations; (d) none of these
2. What kind of data structure would you use to store information about a set of customers and to look up a customer's address knowing only the last name?
   (a) structure; (b) array of structures; (c) two-dimensional array; (d) string; (e) structure containing a string for each customer
3. A collection typically consists of (a) many items of different types; (b) just one item; (c) many structures or objects of the same type; (d) an array of characters
4. An abstract data type is defined by (a) loops; (b) data attributes; (c) operations; (d) loops and attributes;
   (e) attributes and operations.
5. If a sales transaction has an item ID, a customer ID, and a date, then what kind of data structure would you use to store information about a set of sales items? (a) structure;
   (b) array of structures or objects; (c) two-dimensional array; (d) string; (e) structure containing a string for each customer
6. (a) ; (b) ; (c) ; (d) ; (e) none of these

2. Design: divide and conquer

Multiple-choice, T/F

1. Where a problem can be broken down into two problems, one of which is simple to solve and the other is a smaller instance of the original problem, one way to solve the problem is (a) greedy; (b) object-oriented; (c) structured; (d) recursive
2. Divide-and-conquer refers to the use of (a) intractability; (b) recursion; (c) dominant expressions; (d) branching;
   (e) arithmetic operators
3. A recursive method (a) always calls itself when it runs;
   (b) calls itself subject to certain conditions; (c) calls itself not more than once; (d) never calls itself only once;
   (e) uses *while* to loop
4. (T-F) In a recursive method, when the base case applies, the method calls itself.
5. (T-F) A recurrence defines a data type by self reference.
6. A recurrence defines a function (a) selectively;
   (b) iteratively; (c) exponentially; (d) algorithmically;
   (e) redundantly
7. A *disadvantage* of recursion relates to (a) long-windedness; (b) greater difficulty of analysis; (c) danger of stack overflow; (d) slower memory access time; (e) the use of heap memory

3. Algorithm verification

Multiple-choice, T/F

1. A program's correctness (a) may be proven through testing; (b) may be proven mathematically; (c) may be proven through a combination of testing and persuasive language; (d) can never be proven; (e) none of the above
2. An *assert* statement is used to (a) perform a search; (b) help detect syntax errors; (c) help detect logic errors; (d) help detect user-input errors; (e) document program code
3. In a correct algorithm, a loop invariant is true (a) always; (b) never; (c) at the start of a loop body; (d) throughout execution of loop body; (e) sometimes
4. (T-F) It is easy to prove correctness of a program that works.
5. In an inductive proof, showing that $P(0)$ is true is
   (a) the base step; (b) the inductive step; (c) unnecessary; (d) sufficient to prove $P(x)$ implies $P(x + 1)$; (e) sufficient to prove $P(x)$ for all x
6. To prove that an algorithm terminates, we usually rely on
   (a) recursion; (b) *break* statements; (c) convergence;
   (d) total correctness; (e) partial correctness
7. Total correctness is proven by showing (a) partial correctness; (b) termination; (c) good test results; (d) partial correctness and termination; (e) termination and good test results
8. An inductive proof argues in part that (a) if a certain assertion is true for n then it is also true for $(n+1)$;
   (b) a certain assertion is not true for any value n;
   (c) a certain assertion leads to a contradiction; (d) if one assertion is not true, then a second assertion must be true
9. A valid postcondition is (a) output; (b) input; (c) true throughout the execution of a loop; (d) true after a series of statements execute; (e) true under certain conditions dependent on input.

4. Searching and sorting

Multiple-choice, T/F

1. (T-F) The binary search is a way to search any array
2. (T-F) To sort an array normally requires swapping data items that are in order.
3. (T-F) To sort an array normally requires swapping data items that are out of order.

4. The *swap* method (a) returns a value; (b) uses value parameters; (c) uses reference parameters; (d) performs multiple swapping operations.
5. Which algorithm is most similar to the way many people would sort a hand of cards? (a) insertion sort; (b) bubble sort; (c) selection sort; (d) Shell sort; (e) Quicksort
6. Which algorithm finds adjacent elements out of order and swaps them, until none are found out of order? (a) insertion sort; (b) bubble sort; (c) selection sort; (d) insertion sort; (e) Quicksort
7. In Quicksort, the pivot is (a) a subscript; (b) an element value; (c) a turning point in the execution of the algorithm; (d) used to leverage performance; (e) none of these
8. A merge algorithm (a) must be recursive; (b) performs a search; (c) requires two or more sorted arrays; (d) all of these; (e) none of these
9. (T-F) The binary search is a way to search any array
10. (T-F) To sort an array normally requires swapping data items that are in order.
11. (T-F) To sort an array normally requires swapping data items that are out of order.
12. The *swap* method (a) returns a value; (b) uses value parameters; (c) uses reference parameters; (d) performs multiple swapping operations.
13. Which algorithm is most similar to the way many people would sort a hand of cards? (a) insertion sort; (b) bubble sort; (c) selection sort; (d) Shell sort; (e) Quicksort
14. Which algorithm finds adjacent elements out of order and swaps them, until none are found out of order? (a) insertion sort; (b) bubble sort; (c) selection sort; (d) insertion sort; (e) Quicksort
15. In Quicksort, the pivot is (a) a subscript; (b) an element value; (c) a turning point in the execution of the algorithm; (d) used to leverage performance; (e) none of these
16. A merge algorithm (a) must be recursive; (b) performs a search; (c) requires two or more sorted arrays; (d) all of these; (e) none of these Short-answer

5. Algorithm complexity

Multiple-choice, T/F

1. (T-F) The Bubble sort algorithm makes on the order of *n* passes through an *n*-element array.
2. What is the fastest sure way to search for a value in an unsorted array of numbers? (a) calculate hash value; (b) scan from beginning to end until found; (c) sort and perform binary search; (d) choose random elements until the number is found; (e) no fastest way exists.
3. To efficiently locate the number 9 in an array of random integers, we would use a (a) linear search; (b) binary search; (c) Bubble sort; (d) Quick sort; (e) none of these
4. To efficiently locate the number 9 in a sorted array of random integers, we would use a (a) linear search; (b) binary search; (c) Bubble sort; (d) Quick sort; (e) none of these
5. The analysis of algorithms is most concerned with (a) testing solutions; (b) the documentation of programs; (c) object orientation; (d) the time complexity of programs; (e) all of these
6. For a problem of size *n*, a solution of the *worst* time complexity would be (a) 1 (constant time); (b) *n* (linear); (c) *n* squared (quadratic); (d) 2 to the *n* power (exponential); (e) none is worst
7. Each step of the binary-search algorithm (a) reduces the size of the sub-array to be searched by about half; (b) finds the search key; (c) reports failure; (d) moves one array element; (e) compares two array elements
8. The time complexity of the binary search is on the order of (a) 1; (b) *n*; (c) $\log_2(n)$; (d) *n* squared; (e) 2 to the *n*th power
9. (T-F) The binary search is an efficient way to search any large array
10. (T-F) A linear search must inspect, on average, about half the elements in an array.
11. (T-F) To sort an array normally requires swapping data items that are in order.
12. (T-F) To sort an array normally requires swapping data items that are out of order.
13. $\log_2(100)$ is closest to (a) 1; (b) 2; (c) 6; (d) 20; (e) 100
14. $\log_2(1000)$ is closest to (a) 1; (b) 2; (c) 5; (d) 10; (e) 100
15. (T-F) A binary search should be faster than a linear one.
16. The *swap* method (a) returns a value; (b) uses value parameters; (c) uses reference parameters; (d) performs multiple swapping operations.
17. Which algorithm is most similar to the way many people would sort a hand of cards? (a) insertion sort; (b) bubble sort; (c) selection sort; (d) Shell sort; (e) Quicksort
18. Which algorithm finds adjacent elements out of order and swaps them, until none are found out of order? (a) insertion sort; (b) bubble sort; (c) selection sort; (d) insertion sort; (e) Quicksort
19. Which sort has an execution time proportional to $n \times \log_2(n)$? (a) insertion sort; (b) bubble sort; (c) selection sort; (d) insertion sort; (e) Quicksort
20. In Quicksort, the pivot is (a) a subscript; (b) an element value; (c) a turning point in the execution of the algorithm; (d) used to leverage performance; (e) none of these
21. Where *f* is a function, O(*f* (*n*)) means _____ *f* (*n*). (a) exactly; (b) at most; (c) roughly proportional to; (d) at least; (e) following a specification
22. A merge algorithm (a) takes longer than Quicksort; (b) performs a search; (c) requires two or more sorted arrays; (d) all of these; (e) none of these
23. $\log_2(10)$ is closest to (a) 1; (b) 3; (c) 6; (d) 20; (e) 100
24. $\log_2(1000)$ is closest to (a) 3; (b) 10; (c) 50; (d) 100; (e) 1000
25. A linked list is traversed in time (a) O(log*n*); (b) O(1); (c) O(*n*); (d) O(*n*log*n*); (e) O(n^2)
26. Big-O notation sets (a) a precise time complexity; (b) a lower bound; (c) an upper bound; (d) an upper and lower bound; (e) an indefinite estimate

27. Theta notation sets (a) a precise time complexity; (b) a lower bound; (c) an upper bound; (d) an upper and lower bound; (e) an indefinite estimate
28. Big-Omega notation sets (a) a precise time complexity; (b) a lower bound; (c) an upper bound; (d) an upper and lower bound; (e) an indefinite estimate
29. (T-F) O(log n) problems are considered intractable.
30. (T-F) O(2^n) problems are intractable.
31. (T-F) An algorithm with an O(log n) running time is considered fast.
32. (T-F) You might get a bonus for finding an algorithm with an O(2^n) running time.
33. Asymptotic analysis relates most closely to the (a) growth of processor speeds; (b) decline in unprocessed data; (c) rate of growth of functions; (d) complexity of source files; (e) constant factors that affect speed
34. (T-F) Algorithm analysis provides us a single formula by which to determine the best data structure to use

Short-answer

ST 1

1. What is data abstraction?
2. What is the process of defining new data types called?
3. What is a technical term for a structure or object that contains an array of structures or objects?
4. Name one way to implement a collection in Java.

ST2

1. In Big-O notation, what is the complexity of the binary search?
2. In Big-O notation, what is the complexity of the Bubble sort?
3. In Big-O notation, what is the complexity of the selection sort?
4. In Big-O notation, what is the complexity of the Quicksort?
5. In Big-O notation, what is the complexity of the standard solution to the Towers of Hanoi problem?
6. Give complexities of the following operations.
   a) Insert a node at the beginning of a linked list.
   b) Insert a node at the end of a linked list.
   c) Insert a node in order of value.
7. For a linked list of 100 nodes, what is:
   a) the worst-case search time, in node accesses?
   b) The average search time?
   c) The best-case time?
8. What is the running time of linked-list *prepend*?
9. What is the running time of linked-list *insert*, given the address of the node after which the new node is to be inserted?
10. What is the running time of the operation appending a node to a singly-linked list? Doubly linked?
11. What is the running time of the linked-list operation of inserting a node at the end, given the address of the first node?
12. What is the running time of linked-list operation of deleting all nodes?
13. What is the running time to search a linked list of n items organized in ascending order?
14. True/false:
    (a) $O(n^2) = O(2n^2)$
    (b) $O(n^2) = O(100n)$
    (c) $O(50n) = O(n / 2)$
    (d) $O(n) = O(\lg n)$
    (e) $O(\lg n) = O(\lg n + 1000)$
15. Name a sorting algorithm that is of $O(n^2)$ time complexity
16. Name a sorting algorithm that is $O(n \lg n)$ average-case
17. In Big-O notation, what are the worst and average-case complexities of
    (a) the binary search?
    (b) the Bubble sort?
    (c) Quicksort?
    (d) the solution to the Towers of Hanoi problem?
18. If the Bubble sort were improved so that it would be twice as fast, would its complexity category necessarily change, in Big-O notation? Why or why not?
19. Consider this sorting algorithm:
    for $i \leftarrow n$ to 1
      for $j \leftarrow 1$ to n
        if $A[i] > A[j]$
          swap $A[i]$ with $A[j]$
    (a) Is this slower or faster than the selection sort algorithm presented on the slide?
    (b) How much slower or faster?
    (c) Discuss its running time in Big-O terms.
20. Find the solution, in big-O notation, for $T(n)$ =

(a) $\begin{cases} 1 & \text{if } n = 0 \\ 2 + T(n-1) & \text{otherwise} \end{cases}$

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

(c) $\begin{cases} 1 & \text{if } n \le 1 \\ n + T(n-1) + 3 & \text{otherwise} \end{cases}$

(d) $\begin{cases} 1 & \text{if } n \le 2 \\ 2 + T((n-1) / 2) & \text{otherwise} \end{cases}$

(e) $\begin{cases} 1 & \text{if } n = 0 \\ n + ((n-1) / 2) & \text{otherwise} \end{cases}$

21. Simplify:
    (a) $O(3n^2 + 2n + 100)$
    (b) $O(n / 2 + 2 \lg n)$
    (c) $O((n + 2)(n - 2))$
    (d) $O(60 \lg n + 500)$
22. What is the best performance obtainable from an algorithm that uses comparisons in solving the problem of sorting an n-element array?
23. Name two sorting algorithms whose time complexity in Big-O notation is not the same.

Short-answer

1. What loop invariant can be used to prove that the algorithm below returns *true* exactly when parameter *test* is smaller than any element of array *A*?

```
Is-lower (A, test)
For k ← 1 to Size(A)
   if A[k] < test
      return false
return true
```

2. The fact that a value converges during program execution is evidence of the _____ of an algorithm
3. What kind of comment inside a *while* statement would help establish that the statement does its job?
4. What two kinds of comment would help establish that a certain sequence of Java statements bring the program from a certain state of affairs to the desired one?
5. Loop invariants, preconditions, and postconditions help document the _____ of an algorithm.
6. What are the components of a proof of correctness of an algorithm?
7. What special algorithmic technique is used in the solution to the Towers of Hanoi problem? How many times in the pseudocode?

8. The classic Fibonacci function is implemented with what programming feature?

ST 3

1. Fill in the appropriate term for each definition below.
   (a) A true/false proposition that should be true at a certain point in an algorithm's execution, if the algorithm is correct.
   (b) An assertion that should be true before a certain sequence of steps in an algorithm
   (c) An assertion that is true at the start of each iteration of a loop
   (d) An assertion that should be true at the end of a certain sequence of steps
   (e) A verification method that can at best show an algorithm's *incorrectness*
   (f) A verification method based on formal techniques used in mathematics
   (g) A mathematical technique used to prove a general assertion by showing that it is true for one case and that, if it is true for some case, then it is true for the next case after that

Longer-answer questions

1. Write a paragraph or two discussing the problem of searching and sorting arrays. Include considerations of time complexity and use big-O notation to compare algorithms.
2. Write pseudocode for the sorting algorithm of your choice and discuss its complexity.
3. Describe the best, expected, and worst case complexities of various algorithms, using big-O notation. Why is it useful to express an algorithm's running time as a function rather than as a number?
4. Write pseudocode or Java code for a method that will efficiently merge its two parameters, sorted arrays *A* and *B* of sizes *da* and *db*, into the third parameter array, *C*, leaving *C* sorted in ascending order. Discuss its complexity.
5. Write pseudocode to insert a value into a linked list of integers that is in ascending order. What is the complexity of the algorithm?
6. Compare the expected and worst-case running times for searching unsorted arrays, sorted arrays, and linked lists.
7. Write pseudocode or Java code for a method that accepts an integer and a sorted integer array as parameters, and that inserts the single integer into the array, in ascending order. What is its complexity?
8. Write an algorithm that will merge two sorted arrays, *A* and *B*, of sizes *da* and *db*, into the third sorted array, *C*. Discuss its complexity and argue for its correctness.
9. Write pseudocode for a search of a linked list for a given value. State the pre- and postconditions of the algorithm and write comments that argue persuasively for the correctness of the algorithm. What is the complexity of your algorithm?
10. Write pseudocode or Java code for a method that accepts an array, sorted ascending, as a parameter and removes all duplicates, packing the array to fill spaces left by deletions.
11. Write pseudocode or Java code for a method that accepts an array, not necessarily sorted, as a parameter, and removes all duplicates, packing the array to fill spaces left by deletions.
12. Write an algorithm in pseudocode to determine whether a singly-linked list of integers stores any duplicate values. In Big-O notation, what is the worst-case running time?
13. Write a method that accepts an array of integers as a parameter and returns the length of the longest series of consecutive array elements that have the value 2. What is its complexity?
14. Write a method that accepts an array of integers as a parameter and returns the length of the longest series of consecutive elements that have the same value. What is its complexity?
15. Write a method that takes two C-style strings, *s1* and *s2*, as parameters and returns the location of the first match between *s2* and a substring of *s1;* that is, for *find("concat", "cat")*, the return value would be 3. What is the complexity of the method?
16. Modify the *partition* step in Quicksort to select as a pivot the *median* (middle) value of five elements of the array taken at random. Compare sort times for some large set of test data, such as the words in the Web files used in your project. Discuss the results.
17. Design or code an implementation of a linked list of *float*s. Include necessary class declarations and method definitions to implement insertion of one value, and display of all values.
18. Write pseudocode for a search of a linked list for a given value. State the pre- and postconditions of the algorithm and write comments that argue persuasively for the correctness of the algorithm.
19. Write a method that accepts as parameters a sorted array of integers, its occupancy (current size), and a value to be inserted. The method should insert the value at its appropriate location. Write preconditions, postconditions, and loop invariants to argue that the method is correct.
20. Write pseudocode or Java code for an algorithm that finds the sum of the elements of an array of integers, given the array and a value denoting its size. Use preconditions, postconditions, and loop invariants to argue that it is correct. Express its time complexity in big-O notation.
21. Write an algorithm in pseudocode, or code it in Java, to display all the integers from 100 down to 1. Use a loop invariant to show that the algorithm is correct.
22. Write an algorithm in pseudocode, or code it in Java, to input a series of real numbers and find the smallest value among them. Use a loop invariant to show that the algorithm is correct.
23. Using pseudocode, C, or Java, write an algorithm that accepts an array of integers and returns the value os the largest and smallest elements. Show its correctness using preconditions, postconditions, and loop invariants
24. Write an algorithm that tells whether a linked list of integers contains *two or more* nodes that store a given value. Argue for its correctness and discuss its complexity

25. Write an algorithm that takes an array of integers as a parameter and returns the sum of these integers. Argue for correctness and discuss complexity

26. Write a *recursive* algorithm (in C, Java, or pseudocode) that accepts a number, *n*, and finds the sum of all the numbers from 1 to *n*.

Answers to study questions on Topic 1

1. Collections and arrays

Multiple-choice, T/F

1. a. A class is used to declare an abstract data type in Java.
2. b. Each customer could be represented by a structure consisting of the name, street, address, town, etc. The array would make possible a search for the structure with a given name member.
3. c. A collection is a set of items of a single type.
4. e. An abstract data type is characterized by its data properties and a set of behaviors or operations on the data.
5. b. A set of data items of the same class is a collection.

Short-answer

1. The creation of new data types
2. data abstraction
3. collection
4. Define a class that has an array of structures as a member.

2. Design: divide and conquer

1. d. Recursion involves a simple-to-solve base case and a recursive case in which the algorithm repeats for a simpler problem.
2. b. Recursive solutions split a problem into two parts, a simple one and one that is at least simpler than the original problem.
3. b. If the base case does not apply, the recursive case does and the method calls itself.
4. f. When the *recursive* case applies, a method calls itself.
5. f. A recurrence defines a method algorithmically by invoking itself.
6. d. In a recurrence, a method is defined with a branch, in two alternative ways, one of which loops.
7. c. A recursive method, unlike a *while* loop, uses stack space. (Answers (a), (b), and (d) refer to disadvantages of iteration compared with recursion.)
8. b. Testing cannot establish a guarantee of correct results. Such a guarantee is highly desirable but difficult to obtain.

3. Algorithm verification

1. c. The assertion will trigger an error message if a condition is false that the programmer believes always to be true during program execution.
2. c. The loop invariant's validity at the start of the loop body helps establish that a sequence of statements containing the loop accomplish their task.
3. f. Mathematical proof of correctness is difficult and is usually practical only for certain critical parts of a software project.
4. a. The base step is concerned with showing one fact about one item of data.
5. c. If a value steadily progresses toward a certain final value, i.e., converges on it, we know that the loop that contains it terminates.
6. d. Partial correctness means provably correct output for all valid input. Good test results are encouraging but cannot contribute to proof of correctness.
7. a. An inductive proof has a base step, which establishes a fact for a base value *n*, and an inductive step, which establishes that *P*(n) leads to P(*n* + 1), for a certain proposition *P* and all values *n*.
8. d. A postcondition should assert that a state of affairs exists indicating that an algorithm was successful.

4. Searching and sorting

Multiple-choice, T/F

1. f. A binary search cannot be used on an unsorted array.
2. f. Those that are out of order are normally swapped.
3. t. Those that are our of order are normally swapped.
4. c. The values of the reference parameters are swapped and returned to the calling method.
5. a. The algorithm proceeds by placing successive elements in the unsorted part of the array into their ordered location in the sorted part.
6. b. Thus with Bubble, high-valued elements trickle toward the end of the array.
7. b. The pivot value is used in the partition step to arrange all values less than the pivot to its left and all values greater to its right.
8. c. The merge repeatedly takes the lower of two values at the front of sorted input arrays and appends it to an output array.

Short-answer

1. termination
2. a loop invariant
3. precondition, postcondition
4. correctness
5. (a) termination and (b) partial correctness, i.e., correct output for each valid input
6. Recursion. Twice.
7. Recursion
8. (a) assertion
   (b) precondition
   (c) Loop invariant
   (d) postcondition
   (e) testing
   (f) proof
   (g) induction

5. Algorithm complexity

Multiple-choice, T/F

1. f . On the order of one pass may be necessary to put each element in place.
2. b. The linear search cannot be beaten for unordered arrays. Sorting, then searching with the binary search will take longer than scanning, by any sort algorithm.
3. a. A search is necessary to locate a value. The binary search is available only for sorted arrays.
4. b. The binary search efficiently searches an array sorted ascending or descending.
5. d. Algorithm analysis most often estimates the time of execution as a function of size of input.
6. d. The case where time is greatest is the worst (slowest) solution.
7. a. Each step inspects the middle element of a sub-array and eliminates either all elements to its left or all those to its right.
8. c. If *n* is 10, the number of steps is about 4; if 1000, about 10; if 1,000,000, about 20.
9. f. A binary search cannot be used on an unsorted array.
10. t. The best case is one access, the worst is the whole array. The average will be half the array.
11. f. Those that are out of order are normally swapped.
12. t. Those that are our of order are normally swapped.
13. c. Since $2^1 = 2$, $2^2 = 4$, $2^6 = 64$, $2^{10} = 1024$, the closest value is 6. A logarithm is an exponent.
14. d. Since $2^1 = 2$, $2^2 = 4$, $2^5 = 32$, $2^{10} = 1024$, the closest value is 10. A logarithm is an exponent.
15. t. The linear search must inspect every array element; the binary search homes in on its target more quickly.
16. c. The values of the reference parameters are swapped and returned to the calling method.
17. a. The algorithm proceeds by placing successive elements in the unsorted part of the array into their ordered location in the sorted part.
18. b. Thus high-valued elements trickle toward the end of the array.
19. e. Quicksort's running time is $O(n \log n)$
20. b. The pivot value is used in the partition step to arrange all values less than the pivot to its left and all values greater to its right.
21. c. Big-O is a set of functions, all within a constant factor of each other, so an approximation is involved.
22. c. The merge repeatedly takes the lower of two values at the front of sorted input arrays and appends it to an output array.
23. b. Since $2^3 = 8$, the value closest to the base-2 log of 10 is 3.
24. b. Since $2^{10} = 1024$, the value closest to the base-2 log of 1000 is 10.
25. c. Each of *n* nodes may have to be visited.
26. c. Big-O notation estimates the highest (worst-case) running time
27. d. Theta notation a tight bound on running time
28. b. Big-Omega notation estimates best-case running time
29. f. O(log *n*) problems have log*n*-time solutions, which are quite efficient.
30. t. $O(2^n)$ problems take worst-case time that is exponential in the size of the data set.
31. t. An algorithm with log *n* running time is quite efficient.
32. f. An algorithm with $O(2^n)$ running time can be extremely time consuming.
33. c. Asymptotic analysis relates to the rate of growth of functions.
34. f. Algorithm analysis provides us with tools to help determine which data structure to use for particular applications.

Short-answer

1. Binary search is O(log*n*)
2. Bubble sort is $O(n^2)$
3. Selection sort is $O(n^2)$
4. Quicksort is O(*n*log*n*)
5. Hanoi is $O(2^n)$
6. a) O(1)
   b) O(n)
   c) O(n)
7. a) 100
   b) 50.5
   c) 1
8. The running time of linked-list *prepend* is O(1)
9. The running time of linked-list *insert* is O(1).
10. The running time of the operation appending a node to a linked-list is O(*n*).
11. The running time of the linked-list operation of inserting a node at the end, given the address of the first node is O(*n*).
12. The linked-list operation of deleting all nodes is O(*n*).
13. Linear search applies to linked lists regardless of ordering: O(*n*)
14. (a) T (b) F (c) T (d) F (e) T
15. bubble sort, selection sort, insertion sort are all $O(n^2)$
16. Quicksort
17. (a) O(log *n*), O(log *n*)
    (b) $O(n^2)$, $O(n^2)$
    (c) $O(n^2)$, O(*n* lg *n*)
    (d) $O(2^n)$, $O(2^n)$
18. No -- still $O(n^2)$ because nested loop of up to *n* iterations are still used
19. (a) slower
    (b) half as fast
    (c) $O(n^2)$ -- takes precisely n^2 steps. Wastes much time.
20. (a) O(*n*) (b) O(*n*) (c) $O(n^2)$
    (d) O(lg *n*) (e) O(*n* lg *n*)
21. (a) $O(n^2)$ (b) O(*n*)
    (c) $O(n^2)$ (d) O(lg *n*)
22. O(*n* log *n*)
23. (Selection, insertion, bubble) vs. Quick

Longer-answer questions

1. Design or code an implementation of a linked list of *float*s. Include necessary class declarations and method definitions to implement insertion of one value, and display of all values.
2. 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.
3. Through preconditions, postconditions, loop invariants, or other techniques, argue that the selection-sort algorithm below puts the array *A* into ascending order. Give better names to *m* and *s*.

```
void selection(int A[],int s)
{
    for (int i=0; i < s-1; ++i)
    {
        int m = i,j;
        for (j=i+1; j < s; ++j)
        if (A[j] < A[m])
        m = j;
        swap(A[i],A[m]);
    }
}
```

4. Write an algorithm in pseudocode, or code it in Java, to input a series of real numbers and find the smallest value among them. Use a loop invariant to show that the algorithm is correct.
5. Write a *recursive* algorithm (in C, Java, or pseudocode) that accepts a number, *n*, and finds the sum of all the numbers from 1 to *n*.
6. By use of preconditions, postconditions, and loop invariants, prove or argue persuasively that the code below will terminate rather than hanging the computer.

```
int i = 1;
while (i < 100)
{
   cout << i;
   if (i % 2 == 0)
      i *= 2;
   else
      i++;
}
```

7. By use of preconditions, postconditions, and loop invariants, prove or argue persuasively that the pseudocode below will correctly count the number of spaces in a string, *s*.

```
count-spaces(s)
n ← 0
I ← 0
while i < length(s) do
   if s[I] = ' '
      n ← n + 1
   i ← i + 1
}
```

8. Write a method that accepts as parameters a sorted array of integers, its occupancy (current size), and a value to be inserted. The method should insert the value at its appropriate location. Write preconditions, postconditions, and loop invariants to argue that the method is correct.
9. Write pseudocode for a search of a linked list for a given value. State the pre- and postconditions of the algorithm and write comments that argue persuasively for the correctness of the algorithm.
10. Write pseudocode or Java code for an algorithm that finds the sum of the elements of an array of integers, given the array and a value denoting its size. Use preconditions, postconditions, and loop invariants to argue that it is correct. Express its time complexity in big-O notation.
11. Write an algorithm in pseudocode, or code it in Java, to display all the integers from 100 down to 1. Use a loop invariant to show that the algorithm is correct.
12. Using pseudocode, C, or Java, write an algorithm that accepts an array of integers and returns the value os the largest and smallest elements. Show its correctness using preconditions, postconditions, and loop invariants.
13. Write an algorithm that tells whether a linked list of integers contains *two or more* nodes that store a given value. Argue for its correctness and discuss its complexity.
14. Write an algorithm that takes an array of integers as a parameter and returns the sum of these integers. Argue for correctness and discuss complexity.

Study questions on topic 2: Linked structures

1. Dynamic allocation

1. (T-F) It is possible to declare a class of which one attribute is a reference to another instance of the class.
2. (T-F) Dynamic variables are referenced not by identifiers but by references.
3. A reference variable stores (a) an address; (b) a set; (c) input data; (d) a function; (e) nothing
4. To allocate memory for a dynamic variable, we use (a) a loop; (b) a declaration; (c) *new*; (d) *free* or *delete*; (e) none of these
5. A dynamic variable is allocated where? (a) disk; (b) stack; (c) global variable memory; (d) heap; (e) embedded in machine code
6. (a) ; (b) ; (c) ; (d) ; (e) none of these
7. (a) ; (b) ; (c) ; (d) ; (e) none of these

2. Linked lists

1. (T-F) A linked list is of a length specified in its declaration.
2. (T-F) A linked-list node type is a simple type.
3. (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.
4. The data type of a node of a linked list is (a) array; (b) simple; (c) class; (d) collection; (e) reference
5. How many reference members, minimum, must a linked-list node have? (a) 0; (b) 1; (c) 2; (d) 3; (e) 4
6. A linked-list node object must have how many members? (a) 0; (b) 1; (c) 2; (d) 3; (e) more than 3
7. One member of a linked-list node must be a(n) (a) character; (b) integer; (c) node; (d) reference; (e) none of these
8. (a) ; (b) ; (c) ; (d) ; (e) none of these
9. (a) ; (b) ; (c) ; (d) ; (e) none of these

3. Traversal and search of lists

1. Deleting a node from a linked list normally involves (a) changing the data value in the node; (b) changing the value of the node's *next* reference; (c) changing the link in another node so as not to point to the deleted node; (d) changing the link in another node to point to the deleted node
2. 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
3. (T-F) Insertion of a new value into an ordered linked list takes more steps than insertion into an ordered array.
4. To delete a node a singly linked list, using the best-known algorithm, the address of which node must be known? (a) none; (b) the first; (c) the predecessor of the node to delete; (d) the node to delete; (e) the successor of the node to delete
5. 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

4. Conclusion

1. An advantage of linked lists is (a) low access speed; (b) optimal search speed; (c) zero overhead; (d) expandability; (e) none of these
2. A disadvantage of linked lists is (a) high access time; (b) zero expandability; (c) time consuming insertion and deletion processes; (d) all of these; (e) none of these
3. (T-F) If we know the address of a node in a doubly linked list, then to access its predecessor we must start at the first node in the list.
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

Answers to study questions on topic 2

1. Dynamic allocation

1. t. The link is a reference to the next node.
2. b. Passing an array identifier to a function means copying the address, not each element of the array.
3. c. The dereferencing operator (*) converts a pointer to the value it points to.
4. t. A dynamically allocated variable is anonymous and can be reached only through a pointer to it.
5. d. Dynamic variables are allocated by the heap manager in heap memory.

2. Linked lists

6. f. A linked list is of arbitrary length.
7. f. A node is of a class.
8. f. The physical order after an insertion may be different from the physical order in memory.
9. c. A node is a structure or object containing a data value and a link reference.
10. b. A singly-linked list node has one reference, to the node's successor.

3. Traversal and search of lists

1. c. The predecessor node's link reference is changed to point to the successor of the deleted node.
2. c. A list node is accessed starting at -the first list node.
3. f. List insertion typically takes fewer steps, especially if the insertion point is known.
4. c. The predecessor's link must be altered.

4. Conclusion

1. f. A doubly-linked-list node has a predecessor link.

Short-answer

1. Declare a data type for a linked-list node to store a *float*.
2. Write Java code or pseudocode to display all data values in a linked list whose first node is pointed to by reference variable *p* and whose data value in each node is named *data*.
3. What are three operations appropriate to a linked list?
4. How might you append one linked list, *L2*, to another, *L1*?
5. How might you insert one linked list (*L2*) into another (*L1*) at some desired location whose predecessor node is stored in reference *pn*?

Longer-answer questions

1. In one or two paragraphs, state the relationship among these concepts:
   - Abstract data type
   - C class or Java class
   - Data structure
   - Algorithm
2. Define a collection of library-book data records. Each book has a title, author, and catalog number. In Java, define appropriate data types and define methods to search the collection for a particular catalog number. Discuss two ways to implement the collection other than the one you chose; what are their advantages and disadvantages?
3. Design or code an implementation of a linked list of *float*s. Include necessary class declarations and method definitions to implement insertion of one value, in ascending numeric order, and display of all values.
4. Write declarations for abstract data types or classes needed to implement a collection of butterflies. A butterfly has a wing span, a color, and a name. Include some appropriate method declarations for operations on the butterflies and the collection.
5. Write a method that takes two parameters that are references to list nodes, and that links the two nodes by making the second node the successor of the first.

Short-answer answers

1.
```
struct Nodes
{
   float data; nodes* next;
};
```
2.
```
while (p_node != NULL)
{
   printf("%d",p_node->data);
   p_node = p_node->next;
}
```
3. Some operations appropriate to a linked list are prepend node, append node, insert node, delete node, search list, delete all nodes.
4. Step to end of *L1*, assign the address of the first node of *L2* as the *next* of the last node of *L1*.
5. Save *pt* as a temporary variable with value *pn->next*; assign address of *L2*'s first node as *pt->next*; assign *pt* as *next* of last node of *L2*.

Study questions on topic 3: Stacks and queues

1. Stack operations

1. (T-F) A stack is a last-in, first out structure.
2. To add to a stack, we carry out a(n) _____ operation. (a) dequeue; (b) enqueue; (c) pop; (d) push; (e) traversal
3. The *push* operation is carried out on (a) lists; (b) trees; (c) arrays; (d) stacks, (e) queues
4. It is impossible to push data onto a(n) _____ stack. (a) full; (b) empty; (c) initialized; (d) array-implemented; (e) list-implemented
5. Access to the contents of a stack is (a) O(log*n*); (b) restricted; (c) random; (d) O(n^2); (e) none of these
6. _____ on a stack is accessible. (a) the top item; (b) the root; (c) the bottom item; (d) the highest-valued item (e) the lowest-valued item.
7. The stack is a _____-in, first-out structure. (a) lowest; (b) highest; (c) first; (d) last; (e) none of these
8. To look at a value in a stack requires (a) random access; (b) dequeueing; (c) enqueueing; (d) *push*; (e) *pop*
9. (T-F) The concept of a *stack* specifies data attributes but not operations.
10. (T-F) A stack is a kind of collection.
11. To delete from a stack, we carry out a(n) _____ operation. (a) dequeue; (b) enqueue; (c) pop; (d) push; (e) traversal
12. The *push* operation triggers an error message on (a) full stack; (b) full queue; (c) empty stack; (d) empty queue; (e) null reference
13. The *pop* operation is an error in case of (a) full stack; (b) full queue; (c) empty stack; (d) empty queue; (e) null reference
14. (a) ; (b) ; (c) ; (d) ; (e) none of these
15. (a) ; (b) ; (c) ; (d) ; (e) none of these

2. Stack examples

1. An appropriate data structure for a problem involving the checking of matching parentheses would be a (a) list; (b) array; (c) stack; (d) queue; (e) tree
2. A postfix way of writing arithmetic expressions is (a) recursive; (b) Big-O notation; (c) quadratic; (d) reverse Polish notation; (e) none of these
3. To evaluate an expression in reverse Polish notation, we use _____ for each operator encountered. (a) one *push*; (b) two *pushes*; (c) one *pop*; (d) two *pops*; (e) none of these
4. To evaluate an expression in reverse Polish notation, we use _____ for each numeric value encountered. (a) one *push*; (b) two *pushes*; (c) one *pop*; (d) two *pops*; (e) none of these
5. (T-F) A method call at runtime triggers a *push* operation.
6. (T-F) A method call at runtime triggers a *pop* operation.

3. Stack implementations

1. (T-F) Stacks and queues may have different implementations in the Java language.
2. The _____ implementation of the stack enables one-step *pop* and *push* (a) class; (b) tree; (c) array; (d) linked-list; (e) none of these
3. Stack can be full in ___ implementation (a) class; (b) tree; (c) array; (d) linked-list; (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

4. Queue operations

Multiple-choice, T/F

1. A queue is a specialized kind of (a) simple type; (b) array; (c) collection; (d) tree; (e) stack
2. To remove data from a queue, we carry out the _____ operation. (a) dequeue; (b) enqueue; (c) pop; (d) push; (e) traversal
3. The first-in, first-out structure is the (a) list; (b) array; (c) queue; (d) stack; (e) tree
4. (T-F) The queue is an abstract data type.
5. The *dequeue* operation triggers an error message on (a) full stack; (b) full queue; (c) empty stack; (d) empty queue; (e) null reference
6. The *enqueue* operation triggers an error message on (a) full stack; (b) full queue; (c) empty stack; (d) empty queue; (e) null reference

5. Applications for queues

1. An appropriate data structure for a problem involving a set of customers waiting in line would be a (a) list; (b) array; (c) stack; (d) queue; (e) tree

6. Queue implementations

Multiple-choice, T/F

1. (T-F) A queue may be implemented as an array.
2. In the list implementation of a queue, a new item would be added to the (a) front; (b) back; (c) second node; (d) middle node; (e) none of these

Answers to study questions on topic 3

Multiple-choice, T/F

[To be allocated among multiple-choice questions]

1. t. A stack is a last-in, first-out structure.
2. c. A queue contains a collection of values, accessible in a particular order.
3. d. The push operation places a new value on top of the stack.
4. a. The dequeue operation removes the first item from the queue.
5. d. The stack operation *push* adds an item to the top of the stack.
6. a. In an array implementation, a stack may be full to capacity.
7. t. In a queue array implementation, the front and back items are named as subscripts.
8. b. The top of the stack is where items are added or removed.
9. a. Access is instantaneous, but only the top item may be retrieved and an item may only be stored in the top position.
10. c. The oldest item in the queue is the next retrieved.
11. d. A stack is last-in, first-out.
12. e. The *pop* operation retrieves the top item.
13. t. A stack or queue may be implemented by an array or a linked list.
14. t. The queue is defined by a collection (array or list) and a set of two operations: dequeue and enqueue.
15. f. The concept involves storage and retrieval operations.
16. t. A stack is an ordered collection that is accessible from only one end.
17. d. A queue is a first-in, first-out structure.
18. c. A left parenthesis could be stored on the stack, then popped when its mate is found.
19. b. It takes *n* steps to insert an item into an *n*-item queue implemented as a linked list.
20. c. *Pop* deletes from the front of a stack; *push* inserts at the front.
21. d. Reverse Polish notation is postfix in that the operator follows the two operands.
22. d. In RPN, the evaluation operation must pop two numeric values off the stack when an operator is encountered.
23. a. In RPN, a numeric value found in the input stream must be pushed on to the stack.
24. t. A method call pushes an activation record onto the runtime stack.
25. f. A method call pushes an activation record onto the runtime stack.
26. d. The *dequeue* operation cannot run on empty queue.
27. b. The *enqueue* operation cannot run on full queue.
28. a. No value can be pushed onto a full stack.
29. c. The *pop* operation cannot execute on an empty stack.

Short-answer

1. queue
2. An array for stack contents, *int* for top subscript; methods *pop*, *push*
3. An array for queue contents, *int* for size of queue; methods *enqueue*, *dequeue*
4. (a) read 5, push 5, read 3, push 3, read +, pop 3 and 5, add 5 + 3, push 8, read 2, push 2, read *, pop 2, pop 8, calculate 8 * 2, push 16.
   (b) read 6, read *, pop 6, get error popping empty stack, terminate
   (c) read 2, push 2, read 1, push 1, read 3, push 3, read –, pop 3, pop 1, compute 1–3, push –2, pop –2 to display answer, display error on stack not empty at end of input stream
   (d) read 1, push 1, read 4, push 4, read +, pop 4 and 1, add 1 + 4, push 5, read 5, push 5, read –, pop 5, pop 5, calculate 5 – 5, push 0.
   (e) read 6, push 6, read 3, push 3, read /, pop 3, pop 6, compute 6 / 3, push 2, read 2, push 2, read +, pop 2, pop 2, calculate 2 + 2, push 4
5. (a) valid
   (b) invalid (– causes pop of empty stack)
   (c) invalid (not enough operators, leaves 1 on stack)
6. (a) queue
   (b) 25
7. (a) O(1)
   (b) O(1)
   (c) O(1)
   (d) O(*size*)
8. (a) O(1)
   (b) O(1)
   (c) O(1)
   (d) O(1)
9. Running time for insertion is improved from O(*n*) to O(1) if link to last node is maintained.

Longer-answer questions

1. Write an implementation of a queue of sales order numbers. The user should have the options of entering an order or removing it from the queue for order fulfillment.
2. Write pseudocode or Java code to implement the main operations on a queue. Discuss their time complexity.
3. Explain what a postfix expression is, with examples, and what data structure is used to work with it.
4. Write program code or pseudocode to implement the operations that store data in a stack and retrieve data from a stack. Discuss time complexity.
5. Assuming a linked-list implementation of a queue, write methods that make use of the list operations to implement the queue operations.
6. Write a program that reads a Java program and outputs a file with all method definitions removed. To accomplish this, you must create a stack

Short-answer

1. Which data structure would be most appropriate to implement a simulation that determines expected time for customers to wait for a teller to serve them in a bank?
2. What data members and what methods are needed to implement a stack using an array?
3. What data members and what methods are needed to implement a queue using an array?
4. Evaluate the following postfix expressions, showing operations on an appropriate data structure and signalling errors if appropriate:
   (a) 5 3 + 2 *
   (b) 6 * 4 +
   (c) 2 1 3 –
   (d) 1 4 + 5 –
   (e) 6 3 / 2 +
5. Which of the following are valid postfix expressions?
   (a) 3 1 – 2 +
   (b) – 1 2 + 3
   (c) 1 4 2 / 3 +
6. Consider that your task is to build a program to simulate the arrival of bank customers at a teller line. Up to 1000 customers may appear in a day, there may be up to four tellers available, and up to 25 customers could possibly line up for service at one time, in the opinion of the client.
   (a) What data structure would you use to represent the customers lined up?
   (b) If it were implemented as an array, how many elements would the array have?
7. In Big-O notation, what are the running times of the following operations on stacks and queues in the implementations that involve one array and one integer, *size*?
   (a) *pop*
   (b) *push*
   (c) *enqueue*
   (d) *dequeue*
8. In Big-O notation, what are the running times for the operations listed in the previous problem in implementations based on linked lists?
9. In the linked-list implementation of a queue, what improvement in running time is made possible by maintaining a reference to the tail of the queue?
10. What is the result?
    push 3
    push 0
    push 1
    pop *y*
    display *y*
11. What is the result?
    push 1
    pop *x*
    pop *y*
    display *y*

12. What is the result?

```
for i ← 1 to 5
    push i
for i ← 1 to 4
    pop x
pop y
display y
```

Study questions on topic 4: Heaps and priority queues

1. The priority-queue problem

1.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

2.The priority queue lends itself to a straightforward ____ algorithm (a) key-search; (b) traversal; (c) sorting; (d) deletion; (e) none of these

3.(a) ; (b) ; (c) ; (d) ; (e) none of these

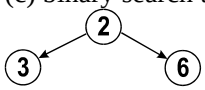
2. Simple array solution

1. In the simple array solution to the priority-queue problem, ____ is/are inefficient (a) *Extract*; (b) *Insert*; (c) either *Insert* or *Extract*; (d) both *Insert* and *Extract*; (e) none of these
2. (a) ; (b) ; (c) ; (d) ; (e) none of these

3. Binary trees and heaps

1.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

2.The figure below is a (a) linked list; (b) minimum heap; (c) binary search tree; (d) hash table; (e) maximum heap



3.The common implementation of a heap is (a) array; (b) linked list; (c) doubly-linked structure; (d) multi-linked structure; (e) none of these

4.The root of a heap implemented as array *A* is (a) the first element of *A*; (b) the last element; (c) a reference; (d) the node pointed to by a certain reference; (e) inaccessible

5.(T-F) In a maximum heap, each node's left child stores a value that is less than that of the parent.

6.(T-F) In a maximum heap, each node's right child stores a value that is greater than that of the parent.

7.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

8.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

9.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

10.If a heap node's subscript is 6, and it is a left child, then the subscript of its parent is (a) 1; (b) 2; (c) 3; (d) 4; (e) 12

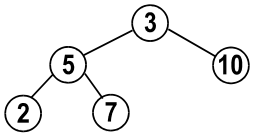
11.(T-F) A heap is implemented by a binary search tree with nodes linked by references.

4. Heap operations

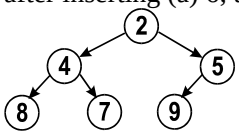
1. The running time of the *Heap-insert* operation is (a) O(1); (b) O(log n); (c) O(n); (d) O(n log n); (e) O(n^2)
2. The running time of the algorithm to delete the minimum value from a heap is (a) O(1); (b) O(log n); (c) O(n); (d) O(n log n); (e) O(n^2)
3. The running time of the algorithm to restore the heap property in a tree, given a node whose two subtrees are both heaps, is (a) O(1); (b) O(log$n$); (c) O(n); (d) O($n$log$n$); (e) O(n^2)
4. (T-F) The heap data structure lets us search for an element with a given value in time O(log n)
5. To find an element of a given value in a heap would take time O(___) (a) 1; (b) log n; (c) n; (d) n log n; (e) n^2
6. The common implementation of a heap is (a) array; (b) linked list; (c) doubly-linked structure; (d) multi-linked structure; (e) none of these
7. To sort an array of size n using a heap will take how many steps, on average? (a) 1; (b) $\log_2 n$; (c) n; (d) n log n; (e) n^2

Short-answer

1. What is the best-known application for heaps?
2. What is the approximate running time of this algorithm, *Build-Heap*, which operates on array *A*?
   Heap-size[A] ← *length[A]*
   For i ← ⌊*length[A]* ÷ *2*⌋ *down to 1*
   Heapify(A, i)
3. Draw a minimum-heap that could be built from these values:
   { 5, 3, 2, 6, 4 }
4. In a maximum heap, what is the relationship between values stored in a parent and its child?
5. What is the name of the operation that restores the heap property to a tree in which both subtrees of a node have the heap property?
6. How many nodes must be looked at to find the second-lowest value stored in a minimum heap?
7. How many nodes must be looked at to find the highest value stored in a maximum heap of size *n*?
8. How many steps does it take to find the lowest value in a maximum heap of size *n*?
9. Name a kind of binary tree in which all leaves are at most one level apart.
10. What is a binary tree in which the bottom-row nodes are filled at the left?
11. Describe the bottom row of a complete binary tree.
12. Name a data structure appropriate for implementing a priority queue in which items may be inserted quickly with arbitrary values and the lowest or highest valued item may be retrieved quickly.
13. Draw a complete binary tree containing the values 4, 2, 5, 9, 8, 6, in that order.
14. Draw a minimum heap containing the values 8, 4, 2, 9, 5, 6
15. What is the heap property for minimum heaps?
16. What is the heap property for maximum heaps?
17. Consider this array:
    `int A[10] = {7,9,3,5,2,4,6,8,1,10};`
    (a) Draw the corresponding complete binary tree;
    (b) Is it a heap? _____
13. Suppose the values 9,3,5,2,4,6,8,10 were in a priority queue of print jobs, with highest priority corresponding to lowest number, i.e., 2 is the highest-priority job. Then if you insert a job with priority 3 and then extract two jobs from the priority queue, what was the priority of the second job you extracted?
18. Consider the tree below.



    (a) Write the corresponding array initialization.
    (b) Is the tree a heap? If not, redraw it as a heap.
19. Draw the trees that correspond to the heap shown below after inserting (a) 6; and (b) 6, followed by 3



20. Draw the tree that corresponds to the heap shown in the figure under the previous problem, after executing the *Extract-min* operation.
21. In a sentence or two, and using Big-O notation, explain what is the running time of Heapsort and why.
22. In Big-O notation, what is the complexity of the operation of extracting all values in a priority queue of size *n*, implemented as
    (a) an unordered arrray;
    (b) a heap
23. What is the depth of a complete binary tree with fifteen nodes?
24. How many nodes are there in a complete binary tree of depth 7?

Longer-answer questions

1. Describe in detail the steps in building a heap from an unordered array and using the heap to produce a sorted array. Discuss the complexity.
2. Implement a minimum-heap data structure in Java, using a class and methods to support insertion and retrieval of minimum value.
3. Write Java code, or pseudocode, for *Heapify*. Include preconditions and postconditions.
4. Explain what the running time for heap insertion and *Extract-min* are, and why.
5. What are two ways to implement a priority queue?
6. Describe the Heapify operation, its preconditions and postconditions, and what it is used for.
7. Evaluate the healp data structure as a possible way to implement a database that will be frequently searched.
8. Write a simple sorting algorithm that uses a heap for internal storage.

Answers to study questions on topic 4

1. The priority-queue problem

2. Simple array solution

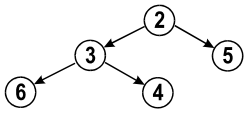
3. Binary trees and heaps

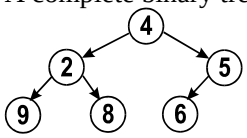
1.c. A heap data structure serves well for a priority queue because its maximum or minimum element is always at the root.
2.b. The figure is a heap because the minimum value is at the root. It violates the binary search tree property and is linked differently from a linked list.
3.a. The binary tree for a heap is stored in an array. Subscripts of parent and child nodes are calculated.
4.a. The root is *A*[0] in a heap implemented in Java with an array *A*.
5.t. Any node below stores a value lower than a node above.
6.f. Any node below stores a value lower, not greater, than a node above such as the parent.
7.b. A complete binary tree with *n* vertices has a depth of log*n*.
8.e. The left child of a node in a complete binary tree implemented as an array will have a subscript twice as large as its parent's.
9.c. The standard heap implementation is an array storing a complete binary tree.
10.c. The parent of a node that is its left child will have a subscript half as large as that of the child.
11.f. A heap is normally implemented as an array in the structure of a complete binary tree; a binary search tree has a very different logical structure.

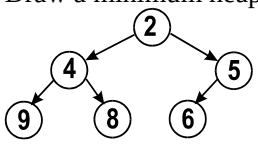
4. Heap operations

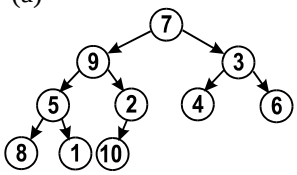
1. b. Inserting an item in a heap entails a step to put the item rightmost in the bottom row of the tree, then filter it up the tree by exchanging it a number of times less than or equal to the depth of the tree, log*n*.
2. b. After determining the minimum value in one step, it is necessary to remove it an restore the heap property, an operation taking up to as many steps as the depth of the tree, log*n*.
3. b. Heapify takes up to as many steps as the depth of the tree.
4. f. To search for an arbitrary value in a heap, it is necessary to extract the minimum value up to *n* times. Extracting the minimum and restoring the heap property would take up to log *n* steps. The search of a heap is worse than a linear search of an array and leaves the heap empty.
5. d. See explanation for previous problem's answer.
6. d. Roughly each array element must be inserted in the heap, which takes roughly log *n* steps for each element.

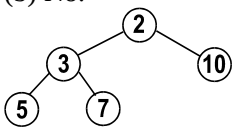
Short-answer

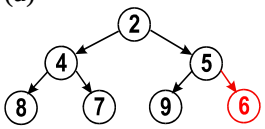
1. Heaps are chiefly an implementation of priority queues.
2. O(*n* log *n*). Building a heap from an unsorted array requires on the order of *n* heapifying steps, each of which is O(log*n*).
3. 

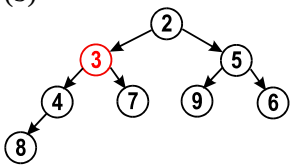
4. The value stored in the parent will be greater than that stored in the child.
5. *Heapify*.
6. Two nodes must be inspected: the first and the second in the priority queue.
7. One node must be looked at to find the highest value in a maximum heap.
8. *n* nodes must be accessed to find the lowest value in a maximum heap.
9. complete binary tree
10. complete binary tree
11. The bottom row of a complete binary tree is filled from the left.
12. heap
13. A complete binary tree containing the values 4, 2, 5, 9, 8, 6:


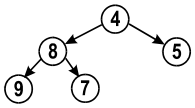
14. Draw a minimum heap containing the values 8, 4, 2, 9, 5, 6


15. The heap property for minimum heaps is: Each non-leaf node stores a value less than or equal to those stored in its child nodes.
16. The heap property for maximum heaps is: Each non-leaf node stores a value greater than or equal to those stored in its child nodes.
17. (a)


(b) no
18. (a) int A[] = { 3, 5, 10, 2, 7 }
(b) No.


19. (a)


(b)


20.


21. *Heap-sort* is O(*n*log*n*), because *n* items must be stored, in log *n* time each; then *n* items must be retrieved in log *n* time each.
22. (a) O(n^2)
(b) O(*n* log *n*)
23. 3
24. 255

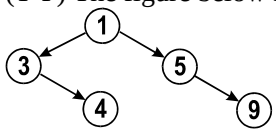
Study questions on topic 5: Trees

1. General trees

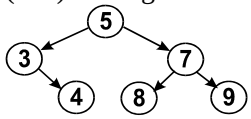
1. In a general tree, each node may have a (a) root; (b) right sibling; (c) leaf; (d) subnode; (e) none of these
2. To model a hierarchy, it is most convenient to use a(n) (a) simple type; (b) array; (c) linked list; (d) binary search tree; (e) general tree
3. (T-F) In a general tree, it is mandatory for each node to have a link to its parent.
4. To model an arithmetic expression, it is most convenient to use a(n) (a) simple type; (b) array; (c) linked list; (d) binary search tree; (e) binary tree
5. (T-F) A tree is a connected, cyclic graph.
6. In an expression tree, an operator is always (a) a leaf; (b) a child; (c) the root; (d) a parent; (e) a subtree
7. (T-F) Removing an edge from a tree produces a non-tree.
8. A node in a tree that is the child of no other node is called the (a) leaf; (b) parent; (c) root; (d) ancestor; (e) none of these
9. A leaf node is one without (a) data; (b) children; (c) a parent; (d) references pointing to it; (e) none of these
10. In an expression tree, the root of a subtree stores (a) an expression; (b) a value; (c) an operator; (d) a position; (e) none of these
11. The maximum path length from the root to a leaf is a tree's (a) degree; (b) connectivity number; (c) depth; (d) edge count; (e) vertex count
12. The main implementation of a general tree shown in this course used (a) parallelism; (b) arrays; (c) nodes with one reference each; (d) nodes with two references each; (e) none of these
13. Adding an edge to a tree without adding a vertex produces (a) a cycle; (b) a deeper tree; (c) a non-tree; (d) answers *a* and *c*; (e) none of these
14. (T-F) All acyclic graphs are trees
15. (T-F) All trees are connected graphs
16. In a tree, any two vertices are connected by ____ distinct path or paths. (a) no; (b) exactly one; (c) one or more; (d) many; (e) exactly two
17. The connectivity number of a tree is (a) 0; (b) 1; (c) at least 1; (d) 2; (e) none of these
18. According to the course material, the result of applying Java grammar rules to parse Java program code would be stored most conveniently in (a) an array; (b) a linked list; (c) a general tree; (d) a binary search tree; (e) a heap
19. The number of edges in a tree is ____ the number of vertices. (a) the same as; (b) one greater than; (c) one less than; (d) one or more greater than
20. A tree has no (a) edges; (b) vertices; (c) paths; (d) cycles; (e) connectivity
21. (a) ; (b) ; (c) ; (d) ; (e) none of these

2. Binary search trees

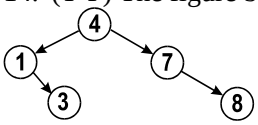
1. (T-F) A binary tree is generally faster to search than a linked list that has the same number of nodes.
2. The height of a binary tree is (a) the number of nodes it contains; (b) the maximum path length between two leaf nodes; (c) the number of leaf nodes; (d) the maximum path length from the root to a leaf node; (e) infinite
3. A binary search tree node has up to two (a) root nodes; (b) children; (c) paths to a given node; (d) parents
4. 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
5. 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
6. A binary search tree is like a linked list in that it (a) is nonlinear; (b) always has two references per node; (c) uses references; (d) has no self-referential members; (e) has leaves
7. A binary search tree node is *unlike* the node of a singly linked list in that it (a) is linear; (b) has two references per node; (c) uses references; (d) has self-referential members; (e) a BST node is not unlike an LL node at all
8. The complexity of a search of a balanced binary search tree of n nodes is O(__) (a) log$n$; (b) n; (c) n log n; (d) n^2
9. The maximum path length from the root to a leaf is the tree's (a) degree; (b) connectivity number; (c) depth; (d) edge count; (e) vertex count
10. To tell whether a certain value is in a binary search tree takes, on *average*, how many steps? (a) O(1); (b) O(log$n$); (c) O(n); (d) O($n$log$n$); (e) O(n^2)
11. To tell whether a certain value is in a binary search tree takes, *worst case*, how many steps? (a) O(1); (b) O(log$n$); (c) O(n); (d) O($n$log$n$); (e) O(n^2)
12. (T-F) The figure below is a binary search tree.



13. (T-F) The figure below is a binary search tree.



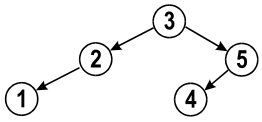
14. (T-F) The figure below is a binary search tree.



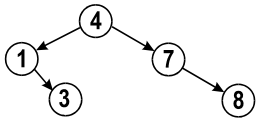
3. Tree traversal

1. Traversal of a binary search tree is normally (a) linear; (b) impossible; (c) recursive; (d) risky; (e) circular
2. A binary search tree is traversed (a) depth first; (b) breadth first; (c) in the order in which nodes were inserted; (d) at random; (e) none of these

15. (T-F) The binary search tree shown below could be generated by the insertion of values 3, 5, 2, 4, 1 in that order.



16. (T-F) The binary search tree shown below could be generated by the insertion of values 1, 4, 3, 7, 8 in that order.



4. BST insert and delete

1. 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
2. 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
3. (T-F) Insertion into or deletion from a binary search tree takes an amount of time proportional to the size of the tree.
4. (T-F) Deletion of a node from a binary search tree may include moving data in one node to a distant node.
5. To delete a leaf node from a binary tree, (a) nullify the reference to it in the node's parent; (b) find the leftmost node of the node's right child; (c) link the node's parent to the node's child; (d) link the root to the node; (e) make the node the child of its own child
6. To delete the parent of a single node (only child) from a binary search tree, (a) nullify the reference to it in the node's parent; (b) find the leftmost node of the node's right child; (c) link the node's parent to the node's child; (d) link the root to the node; (e) make the node the child of its own child
7. If node *b* in a binary search tree has a value smaller than node *a*, and *b* is inserted after node *a*, then *b* is located where in relation to node *a*? (a) above and left of; (b) below and left of; (c) below and right of; (d) above and right of; (e) none of these

5. BST implementation

1. A BST node implemented in Java contains ____ reference(s) (a) 0; (b) 1; (c) 2; (d) 3; (e) more than 3
2. How many non-null references does a BST node have? (a) 0; (b) 1; (c) 2; (d) 0 or 1; (e) 0, 1, or 2
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. Performance issues

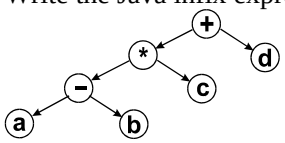
Multiple-choice, T/F

1. (T-F) The level of two leaf nodes in a balanced tree differs at most by one.
2. A degenerate tree has (a) balanced branches; (b) no nodes; (c) nodes with only 1 reference; (d) almost all branches going uniformly left or right at all levels; (e) none of these
3. Search time on a degenerate tree is (a) constant; (b) logarithmic-time; (c) linear time; (d) quadratic time; (e) none of these

Short-answer

ST1

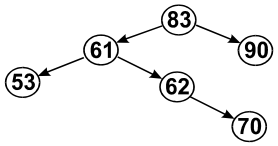
1. Which data structure would you use to implement an outline processor?
2. Write a structure-type definition for a general-tree node.
3. Name or describe the member data items required to define a general-tree node.
4. In the logical structure of a general tree, one node may have how many siblings?
5. Write the Java infix expression that corresponds to this tree:



6. Draw the expression tree that corresponds to the Java expression *a / b – c * d*.

ST2

1. Draw the binary search tree, of characters, that would be formed by inserting the following values in order: ‘S’, ‘E’, ‘L’, ‘I’, ‘T’, ‘B’, ‘M’
2. What is the average time complexity of the standard algorithm to insert one new value into a binary search tree of *n* nodes?
3. What is the rough expected running time of a standard algorithm to build a binary search tree from a random series of distinct values, and then to traverse the tree displaying value in ascending order?
4. What is the binary-search-tree property?
5. What is the time complexity of inserting one new value into a *degenerate* binary search tree of *n* nodes?
6. What is the time complexity of inserting one new value into a *balanced* binary search tree of *n* nodes?
7. Sketch the binary search tree created by inserting values in the following order: ‘K’, ‘M’, ‘A’, ‘D’, ‘H’, ‘B’, ‘R’, ‘Q’.
8. List a possible ordering of values that could produce this BST when inserted one by one:



9. Draw the BST that would be formed by inserting, in order, 4, 2, 5, 9, 8, 6.
10. In big-O notation, for balanced binary search trees, what are the complexities of
    (a) node insertion?
    (b) node deletion, for each of the three cases?
    (c) tree sorting, from scratch?
    (d) displaying contents of tree using inorder traversal?
11. Write the expression tree for “*a* + 2*b* – *c* + 4”.
12. Draw a binary tree with five nodes and three leaves.
13. A binary tree consists of eight nodes, of which four are leaves. If new nodes are added in such a way that one leaf node becomes a parent with two children, the tree will have how many nodes, including how many leaves?
14. Using nodes with only two links each, draw a tree to represent a family whose senior member (Ada) has three children, Betty, Carl, Don, and Emily. Betty has one child, Carl has none, Don has three, and Emily has two.
15. Write a class or structure-type declaration, listing data members of the nodes for the family tree described in the previous problem. A person’s name is sufficient to describe the person.
16. Draw a binary search tree of characters for the values S, I, T, M, E, L, B, inserted in that order.
17. What does this code do?

```
int weight(nodes* p)
{
   if (p == NULL)
      return 0;
   else
      return 1 + weight(p->left) +
          weight(p->right);
}
```

18. Name three ways to traverse a tree.

Longer-answer questions

1. What are the advantages and disadvantages of organizing a local area network as a tree? As a ring?
2. Describe three tree-like structures we have encountered in this course and briefly name their implementations.
3. What are some of the applications of general trees?
4. What are some advantages of binary search tree structures over linked lists and arrays?
5. Describe in detail the data members of a general-tree structure and give pseudocode for two or three operations; or code in Java.
6. Write an algorithm to find the total number of nodes in a general tree, given its root.
7. Write pseudocode or Java code for
   (a) a BST search or
   (b) traversal.
   (c) deletion
   (d) insertion
   (e) emptying tree
8. Write pseudocode or Java code to delete a node from a binary search tree, given a reference to it and given a reference to the root of the tree.
9. Discuss the relative complexities of the storage, retrieval, and search operations on (a) a stack; (b) a linked list sorted in ascending order; (c) a binary search tree.
10. Compare and contrast the heap property and the BST property and discuss implications for operations on these structures.
11. Compare the complexities of various standard operations, including searching, on arrays, linked lists, and trees, using Big-O notation. Option: use pseudocode or code to illustrate your points.

12. Write pseudocode or Java code for *one* of the following and discuss its complexity:
    (a) delete a node from a binary search tree, given a reference to it and given a reference to the root of the tree;
    (b) insert a value into a binary search tree whose root is pointed to by *p*;
    (c) display all values in a binary search tree in alphabetical order by key.
13. Write a recursive method that accepts a reference to a BST node as a parameter and returns 0 if the reference is null, otherwise returns one plus the combined weights of the node's left and right subtrees. That is, your method should return the number of nodes in a BST subtree whose root is pointed to by the parameter. Prove the correctness and discuss the complexity of the method.
14. Compare and contrast heaps and binary search trees.
15. Describe how a binary search tree of integers could be implemented using an array.
16. What are some of the applications of trees? What are some advantages of binary search tree structures over linked lists and arrays?
17. Write pseudocode or Java code for a BST search or traversal. Argue for its correctness and characterize its complexity.
18. Declare classs or classes for an operating system's disk-directory tree and its contents. Each item, or entry, in the directory should have a name and a reference to a subtree. An entry that is a file should have a null subtree; a subdirectory's subtree should store its contents. You need not write methods or test your code. Write a sentence or two describing this data structure and draw a diagram of it.
19. In Java, declare a type for a BST node storing a floating-point number, and write a method to initialize such a node.
20. Write the pseudocode or program code for BST search and comment it, arguing that it (a) terminates and (b) returns correct result when search key is not in tree.
21. Modify a tree-manipulating file in subdirectory *bst* (*treesort.c*, *treesort.cpp*, or *intsort.cpp* and the library it uses, *bst.h*), or write your own BST code from scratch, to prompt for integer values to delete from a binary search tree and to delete them. (See slides.) Display contents of resulting tree. Using data file *intsort.dat*, test for deletion of 50 (not present), 18, 15, 61 and 83. Submit your new code and the test results.
22. Implement a word-counting program that reads a text file and stores each white-space-delimited string in a node of a binary search tree. The node should also store a count of the number of times the word has been found so far. Use your word counter with the text of a C program.
    (*Challenge:* modify the specifications to define a word, or *lexeme*, as being a white-space-delimited single *punctuator* (';', ',', ':', '{', '}', '(', ')', '[', ']', '+', '=', '-', '.', '/', '<', '>', '*') or a series of characters following or preceding a punctuator.) Display in ASCII/alphabetical order the lexemes found and their counts.
23. Write a program that inserts random numeric values into a binary search tree. Estimate the average path length in trees of different sizes, by having your program count the number of steps necessary to search for a value. Compare this to the calculated estimates for best, worst, and expected running times.

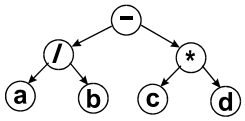
Answers to study questions on topic 5

1. General trees

A. Multiple-choice, T/F

1. c. Successor nodes are particular to a linked list; leaf nodes are particular to a tree.
2. e. A general tree may model a hierarchy, in which each item may have one, two or more items immediately below it.
3. f. Only a doubly-linked general tree will give each node a link to its parent.
4. e. A binary tree models an arithmetic expression; each parent node is an operator with two operands represented by subtrees.
5. f. A tree is defined mathematically as a connected, *acyclic* graph.
6. d. An operator is the parent of its operands, which are represented as subtrees.
7. t. Removing an edge disconnects the tree; a tree must by definition be connected.
8. c. The root has no parent.
9. b. A leaf may have a parent, but no children.
10. c. A subtree is a sub-expression, with the binary operator as the root of the subtree above the two operands.
11. c. The depth of a tree is the greatest distance, in edge, from the root to a leaf.
12. d. The general tree is represented as nodes, each with a child link and a right-sibling link.
13. d. Adding an edge to a tree without adding a vertex must create a path from some node back to itself, i.e., a cycle, and a graph with a cycle cannot be a tree.
14. f. Unconnected acyclic graphs are not trees.
15. t. All vertices in a tree must be connected to each other by some path.
16. b. Only one path exists between any two vertices in a tree, and any two vertices are connected by some path.
17. b. All vertices in a tree are connected.
18. c. A parse tree could be implemented as a general tree, with each nonterminal program element at the root of a subtree.
19. c. $|V| = |E| + 1$, where $|V|$ is the size of the set of vertices and E is the set of edges. Example: tree with two vertices and one edge connecting them.
20. d. A tree is an acyclic graph.

B. Short-answer

1. A general tree implements an outline.
2.
```
struct nodes
{
  void* data;
  nodes* child,*r_sibling;
};
```
3. *data, child, r_sibling;*
   (optional:) *l_sibling*, *parent*
4. A node may have any number of siblings, though only one right sibling directly connected to it.
5. *(a – b) * c + d*
6. 

2. Binary search trees

A. Multiple-choice, T/F

1. t. A tree's depth determines its search time; the depth is usually smaller than the length of a list with the same number of nodes.
2. d. A tree node is accessed starting at the root.
3. b. A node has a left and right child.
4. b. Each node is the root of a subtree.
5. c. The right child of the root will contain a higher value than the root.
6. c. Each node has references to a left and a right child.
7. b. A linked list node may have one link reference.
8. a. Each step in the search reduces the remaining work by half.
9. c. The depth of a tree is the maximum number of edges in a path from root to a leaf.
10. b. The depth of the tree will normally be roughly $\log_2$ of *n* edges.
11. c. The worst case is a degenerate binary tree, whose form is similar to that of a linked list. It is *n* nodes long.
12. f. The 3 and 4 should not be left of the 1, because they have a higher value.
13. f. 8 should not be left of 7, since its value is greater.
14. t. Possible order of insertion: 4, 1, 3, 7,8.

3. Tree traversal

1. c. The standard tree traversal algorithm is recursive.
2. a. Traversal requires descending a tree to a leaf repeatedly.
3. t. The tree shown contains all values listed, has the binary-search-tree property, and each child node follows its parent node in the chronological order of insertion.
4. f. The tree shown has the binary-search-tree property, but 4 is inserted before 3, so 3 should be below 4.

4. BST insert and delete

1. b. A new node is the child of a node that was previously a leaf.
2. e. A binary search tree has no particular maximum size.
3. f. Insertion or deletion takes time proportional to the depth of the tree.

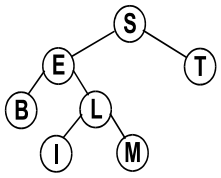
4. t. The deletion of a parent of two nodes may require replacing the node's value with the value stored in the node that is leftmost in the right subtree of the node.
5. a. Since a leaf has no children, it may simply be unlinked to delete it.
6. c. Linking the parent of a node to the node's child in effect removes the node from the tree without removing its child.
7. b. A node inserted later is below; a node with a smaller value is inserted to the left.

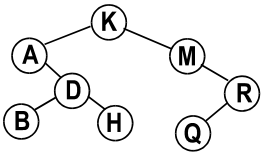
5. BST implementation

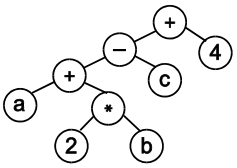
6. Performance issues

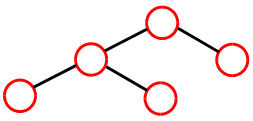
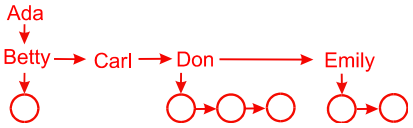
1. t. A balanced tree is defined as one in which all nodes are at levels differing by at most one.

Short-answer

1. 

2. O(log*n*). An insertion is at a leaf location; a leaf is on average at about the depth of the tree.
3. O(*n*log*n*). Each step in building a tree takes O(log*n*) steps, because the depth of the tree reaches that value. There are *n* building steps. The traversal takes *n* steps. The traversal's running time is dominated by the building time.
4. The binary search tree property states that a node's left child stores a value less than or equal to that of the node, and a node's right child stores a value greater than or equal to that of its parent.
5. O(*n*), because a degenerate tree is in the form of a linear list, since each node inserted stores a greater value than its predecessor.
6. O(lg *n*), because a balanced tree branches out evenly, with no path longer than the base-2 logarithm of *n*, for *n* nodes.
7. 

Traversal: A, B, D, H, K, M, Q
8. 83, 61, 53, 62, 70, 90, or
8, 90, 61, 53, 62, 70, etc.
9.
10. (a) O(lg *n*)
(b) leaf: O(lg *n*)
parent of 1 node: O(lg *n*)
parent of 2 nodes: O(lg *n*)
(c) O(*n* lg *n*)
(d) O(*n*)
11. 

12. One Solution:

13. The tree will have ten nodes and five leaves.
14. 

15.
```
struct tree_node
{
   char name[40];
   tree_node *child,
        *sibling;
};
```
16. Counts nodes in a binary search tree.
17. Preorder, postorder, inorder

Study questions on topic 6: Hashing

1. Hash tables

1. (T-F) Hashing arranges items by comparing them with each other.
2. 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
3. Address calculation in source code is associated with (a) hashing; (b) dynamic allocation; (c) all array accesses; (d) recursion; (e) searching
4. A well-respected hash function is (a) the modulo-2 operation; (b) dividing by 2; (c) the factorial function; (d) recursive; (e) mid squares
5. (T-F) Time for insertion into a sparsely filled hash table of size n is $\Theta(\log n)$
6. (T-F) With a hash table, lookup and insertion are of the same complexity.
7. (a) ; (b) ; (c) ; (d) ; (e) none of these

2. Simple hashing

1. (T-F) A bit vector (bit array) can store information about whether a value belongs to a set in a location that depends on the value.
2. Simple hashing involves storing items in locations whose array subscripts are (a) hashed; (b) computed; (c) the values stored; (d) chosen at random; (e) none of these
3. The simple-hashing way to store a set is (a) as a linked list of set elements; (b) as an array of set elements; (c) as an array of Booleans such that the elements whose subscripts are in the set are assigned *True*; (d) as a string that describes the set; (e) none of these
4. If two keys map to the same slot in a hash table, (a) one must be discarded; (b) processing is the same as if only one key mapped to that slot; (c) the collision situation must be resolved; (d) there is only one way to respond; (e) none of these
5. (a) ; (b) ; (c) ; (d) ; (e) none of these
6. (a) ; (b) ; (c) ; (d) ; (e) none of these

3. Linear probe implementation

1. 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
2. (T-F) For a hash table of size n, the lookup operation can be worse than O(log n).
3. (T-F) For a hash table of size n, implemented for linear probe, lookup can be worse than O(n).
4. (T-F) For a hash table of size n, the lookup operation can be faster than O(log n).
5. Double hashing is a variant of (a) the mid-squares hash function; (b) linear probing; (c) chaining; (d) Quicksort; (e) the linked list
6. (T-F) An efficient hash table can be implemented with an array.

4. Bucketing (chaining)

1. A bucket is used in (a) binary search trees; (b) hashing; (c) linked lists; (d) stacks; (e) queues
2. (T-F) For a hash table of size n, implemented with chaining, lookup can be worse than O(n).
3. (T-F) Chaining is a way to avoid collisions in hash tables
4. Clustering occurs in (a) only hash tables populated by open addressing; (b) only hash tables with chaining; (c) hash tables with open addressing or chaining; (d) perfect hash functions; (e) the Bubble sort
5. Clustering occurs in (a) a linked list; (b) a binary search tree; (c) a hash table; (d) a stack; (e) a queue
6. (T-F) An efficient hash table can be implemented with one linked list.
7. (T-F) An efficient hash table can be implemented with multiple linked lists.
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

Short-answer

1. Collisions are a problem in the use of which data structure?
2. What kind of function is used to uniformly scatter data items, by key, across an array so that the location of a data item can be easily obtained from its value?
3. What is the name of the occurrence when many data values hash to a similar vicinity of the table?
4. What hashing technique uses a hash function to generate a location and an offset?
5. What is the name of the technique used to resolve collitions in a hash table that is stored entirely in an array?
6. The use of linked lists in hash tables is called _____.
7. Supply the terms:
   (a) A problem that arises when we attempt to store data in a location whose address is calculated from a key value
   (b) A method for storing data in a location whose address is calculated from a key value
   (c) A solution to the problem of two keys mapping to the same address
   (d) A subprogram that scatters data in a seemingly random and uniform way
   (e) A measure of the density of a hash table
8. Under what circumstances will any hashing scheme base only on arrays break down?
9. In Big-O notation, what is the
   (a) worst-case running time to store one data item in a hash table of size *m* storing *n* items?
   (b) best-case running time?

Longer-answer questions

1. Describe some problems solved by hashing, some problems it raises for software implementers, and solutions to these problems.
2. Describe two ways to implement a hash table and the chief way in each to resolve the collision problem. Discuss complexity of each.
3. Give arguments for and against the idea that theory is of critical importance in working with computers.

Answers to study questions on topic 6

1. Hash tables

1. f. Hashing determines the location of an item by performing a computation on the value of the item alone.
2. d. The hash function maps from a key value to an address. If a hash table is an array of buckets, then the hash function returns the subscript of a bucket.
3. d. The hash function maps from a key value to an address. If a hash table is an array of buckets, then the hash function returns the subscript of a bucket.
4. b. A bucket is the structure, perhaps a linked list, that stores one or more keys. A hash function determines which bucket a key goes to.
5. e. Modulo-2 maps to only two values; divide-by-2 yields too many different values; factorial produces a huge result, too big for a hash table. Mid-squares scrambles data well.
6. f. If there are no or few collisions, an item can be inserted into a hash table in constant time.

2. Simple hashing

1. a. Hash functions in effect calculate the location of a key from its value.

3. Linear probe implementation

1. f. Hashing determines the location of an item by performing a computation on the value of the item alone.
2. t. Like hashing, the bit-vector implementation of sets stores information about a value at a location that may be calculated from the value.
3. b. Load factor rises as the table has more slots filled.
4. t. A densely populated hash table can require up to *n* steps to search.
5. f. The array-based hash table's size puts an upper bound on the running time.
6. t. A single chain has no limit on how many items it may store.
7. t. The open addressing method uses an array.

4. Bucketing (chaining)

1. c. Two solutions to the collision problem are: implement the buckets as linked lists; store an extra key in the next bucket after the one whose location was returned by the hash function.
2. t. Insertion requires lookup and only a constant amount of steps beyond that.
3. t. Hash-table lookup can be done in constant time if there have been few collisions on insertion.
4. b. Calling a hash function twice makes sense only when a linear probe is the result of a collision.
5. f. Chaining is a way to *resolve* collisions; there is no way to *avoid* them.
6. c. Clustering occurs whenever many collisions occur in the same vicinity of a hash table.
7. c. When a hash function maps many values to the same vicinity of a hash table, we have clustering, a problem.
8. f. A linked list is very slow to search, so it would not be used for an entire hash table, only for part of it.
9. t. The chaining strategy for hash tables uses an array of linked lists.

Short-answer

1. a hash table
2. hash function
3. collision
4. double hashing
5. linear probing (open addressing)
6. chaining
7. Supply the terms:
   (a) collision
   (b) hashing
   (c) linear probing
   (d) hash function
   (e) load factor
8. when the number of values stored exceeds the array size
9. (a) O(*n*)
   (b) O(1)

Study questions on topic 7: Graphs

1. Relations and directed graphs

1. A graph is (a) a set of integers; (b) a set of vertices; (c) a set of vertices and a set of edges; (d) a set of edges; (e) a set of paths
2. Which might be used in task scheduling? (a) functions; (b) vertices; (c) undirected graphs; (d) directed graphs; (e) none of these
3. (T-F) In a connected graph an edge exists between each pair of vertices.
4. The connectivity number of a graph is (a) the number of vertices; (b) the number of edges; (c) the number of paths; (d) the number of distinct connected subgraphs; (e) the number of other adjacent vertices
5. The degree of a vertex in a graph is (a) the number of vertices in its graph; (b) the number of edges in its graph; (c) the number of paths; (d) the number of distinct connected subgraphs; (e) the number of other vertices adjacent to it
6. 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
7. 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.
8. (a) ; (b) ; (c) ; (d) ; (e) none of these

2. Paths

1. A series of edges that connect two vertices is called (a) a path; (b) a cycle; (c) a connection; (d) a tree; (e) a collection
2. A series of edges that form a path from a vertex to itself is (a) a spanning path; (b) a cycle; (c) a connection; (d) a tree; (e) an edge
3. To design a communications network that joins all nodes without excessive lines, we must find a (a) path; (b) connectivity number; (c) minimal spanning three; (d) expression tree; (e) search tree
4. To find a path from one vertex to another, we may use (a) depth-first search; (b) connectivity number; (c) minimal spanning tree; (d) expression tree; (e) search tree
5. A graph in which exactly one path joins any pair of vertices is (a) subgraph; (b) tree; (c) connected graph; (d) cyclic graph; (e) unconnected graph
6. (a) ; (b) ; (c) ; (d) ; (e) none of these

3. Weighted graphs

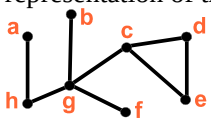
1. A weighted graph has an adjacency matrix that is (a) integers; (b) vertices; (c) real numbers and ∞; (d) booleans; (e) none of these
2. (T-F) The Dijkstra algorithm examines all possible paths from each vertex to each other one.
3. (T-F) The Dijkstra algorithm builds a tree of the minimum-weight paths from a single source vertex to each other vertex.
4. A minimal spanning tree is a subset of a (a) binary search tree; (b) undirected graph; (c) weighted graph; (d) array; (e) none of these
5. (a) ; (b) ; (c) ; (d) ; (e) none of these
6. (a) ; (b) ; (c) ; (d) ; (e) none of these

4. Implementations

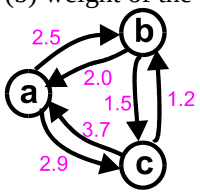
1. A graph may be fully represented by (a) its vertices; (b) its edges; (c) an adjacency matrix; (d) the degrees of its vertices; (e) none of these
2. A graph may be conveniently represented as (a) a vector of characters; (b) a two-dimensional array of Booleans; (c) a single linked list of Booleans
3. In the array implementation of graphs, what is the type of the array elements? (a) Booleans; (b) characters; (c) integers; (d) structures; (e) arrays
4. The two-dimensional array implementation of graphs tells whether or not a pair of vertices are (a) part of the graph; (b) connected; (c) adjacent; (d) the same; (e) none of these
5. A list implementation of a graph might use an array of linked lists, each list containing all the _____ a vertex. (a) paths containing; (b) vertices adjacent to; (c) edges adjacent to; (d) trees containing; (e) none of these
6. (a) ; (b) ; (c) ; (d) ; (e) none of these

Short-answer

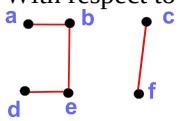
1. A two-dimensional array that represents a graph is a(n) _____ matrix.
2. A graph is defined by a set of _____ and _____.
3. A directed or undirected graph may be implemented in C or Java by _____ or _____.
4. Write an adjacency matrix *and* a diagram of a linked-list representation of the graph below.



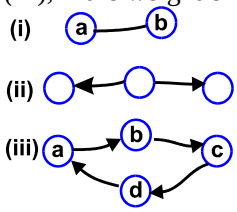
5. What is the minimum number of vertices in an undirected cyclic graph?
6. In the graph below, what are the
   (a) outdegree of vertex *a*?
   (b) weight of the shortest path from vertex *c* to vertex *a*?



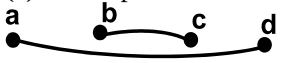
7. With respect to the graph below,



   (a) Is it cyclic?
   (b) Is it connected?
   (c) List, in order, the vertices in the longest path
8. In Big-O notation, what is the running time of the Dijkstra algorithm? (See slide) Why?
9. Referring to the graphs below,
   (a) Which are directed?
   (b) Which are cyclic?
   (c) What is the degree of *a* in (*i*)?
   (d) What is the weight of the shortest path from *a* to *d* in *(iii)*, if the weight of each edge is 1?



10. (a) Is the graph below connected?
    (b) Is it cyclic?
    (c) Find a path from *a* to *d*



Longer-answer questions

1. Contrast depth-first and breadth-first search of a graph.
2. Describe the Dijkstra algorithm for finding paths in graphs.
3. Write pseudocode or Java code for an algorithm to convert a graph represented by a matrix to an array-of-lists representation
4. Write pseudocode or Java code for an algorithm to convert a graph represented by an array-of-lists to a matrix representation
5. Describe the Prim algorithm for finding paths in graphs.
6. Describe a way to find a path from a given vertex in a graph to a second given vertex.

Answers to study questions on topic 7

1. Relations and directed graphs

1. c. The set of edges is a relation on the set of vertices.
2. d. An arrow might run from one task to a task that can be performed after it.
3. b. The set of edges is a relation on the set of vertices.

2. Paths

1. a. A path joins vertices directly.
2. b. A cycle is a path from a vertex to itself without passing through any other vertex twice.
3. c. The minimal spanning tree contains sufficient edges to connect each vertex to each other, with minimal total weight.
4. a. Depth-first and breadth-first are two kinds of search for paths in a graph.
5. b. A tree is connected and acyclic.

3. Weighted graphs

1. f. A path exists between any two vertices, but not necessarily an edge.
2. d. A subgraph is connecte4d if each of its vertices is connected by some path to each other one.
3. e. If a vertex is adjacent to five other vertices, its degree is 5.
4. b. All vertices in a tree connect and there are no cycles.
5. f. The Dijkstra algorithm builds one tree of paths using a greedy algorithm. It does not examine every possible path.
6. t. The tree produced by the Dijkstra algorithm contains optimal paths from the source node to each other one.
7. b. In a weighted graph, the least-weight path may have more edges than some greater-weight path.

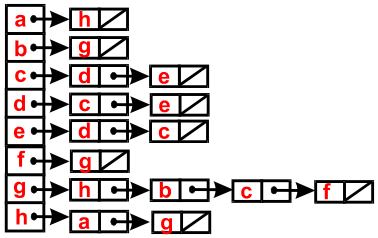
4. Implementations

1. c. The edges are the relation denoted by the matrix.
2. b. A graph may be represented by a matrix (2-dimensional array) or by an array of linked lists.
3. a. Each element in the two-dimensional array tells whether the two vertices specified by its two subscripts are adjacent.
4. c. The array is called an adjacency matrix.
5. b. The adjacency matrix of an *n*-vertex graph may be represented as an *n*-element array of lists of vertices adjacent to a given one.

Short-answer

1. adjacency
2. vertices and edges
3. two-dimensional array of Booleans or an array of lists of integers.
4. 3
5.

	a	b	c	d	e	f	g	h
a								1
b							1	
c				1	1		1	
d			1		1			
e			1	1				
f							1	
g		1	1			1		1
h	1						1	



6. (a) 2
   (b) 3.2
7. (a) no
   (b) no
   (c) a, b, e, d
8. $O(n^2)$; nested loops
9. (a) ii, iii
   (b) iii
   (c) 1
   (d) 4
10. (a) no
    (b) no

Study questions on topic 8: Multithreading

1. Concurrency

1. Multitasking is (a) concurrent activities by multiple computers; (b) a series of activites by one processor; (c) one processor running multiple programs concurrently; (d) the same as multithreading; (e) parallel computing
2. Multithreading is (a) concurrent activities by multiple computers; (b) a series of activites by one processor; (c) one program running multiple processes concurrently; (d) the same as multitasking; (e) parallel computing
3. A thread simulates ownership of (a) a network; (b) a user account; (c) a computer's resources; (d) a processor; (e) none of these
4. Garbage collection in the background is an example of (a) batch processing; (b) non-concurrency; (c) parallel computing; (d) multithreading; (e) distributed computing
5. (a) ; (b) ; (c) ; (d) ; (e) none of these

2. Java threads

1. The Java *new* operator invokes (a) dynamic allocation of memory; (b) memory allocation on the stack; (c) deallocation of memory; (d) allocation and deallocation of secondary storage; (e) none of these
2. Threads of equal priority get processor resources by (a) pre-emption; (b) round-robin time slicing; (c) submitting requests; (d) user intervention; (e) none of these
3. The Java class that enables concurrency within a program is (a) *System*; (b) *String*; (c) *Output*; (d) *Memory*; (e) *Thread*
4. (a) ; (b) ; (c) ; (d) ; (e) none of these

3. Priorities, scheduling, and synchronization

1. The Java scheduler ensures that the ____ thread executes at all times (a) shortest; (b) next-in-line; (c) highest-priority; (d) lowest-priority; (e) none of these
2. Synchronized methods for one object may execute (a) under multiple concurrent threads; (b) without limitation; (c) under only one thread at a time; (d) all of these; (e) none of these
3. (a) ; (b) ; (c) ; (d) ; (e) none of these

4. Producers and consumers

1. Daemon threads (a) execute in the foreground; (b) execute in the background; (c) can execute while waiting for the threads they serve to start; (d) all of these; (e) none of these
2. Java garbage collection is a (a) program; (b) distributed service; (c) web service; (d) daemon thread; (e) none of these
3. Daemon threads are (a) programs; (b) distributed services; (c) utilities performing services for other threads; (d) error threads; (e) none of these
4. (a) ; (b) ; (c) ; (d) ; (e) none of these

Study questions on topic 9: Graphics programming

1. Events, controls, and views

1. In a model/view/controller architecture, program execution is driven by (a) events; (b) views; (c) models; (d) commands; (e) interrupts
2. Three important aspects of an application are (a) method, view, and control; (b) data structure, model, and view; (c) control, design, and method; (d) control, model, and view; (e) analysis, data structure, and method
1. An instance of a view class may be (a) an array; (b) a character; (c) a window; (d) a screen driver; (e) a menu option
2. A GUI or application framework (a) is a data structure; (b) is a single class; (c) is a set of classes that defines a user interface; (d) is a form of documentation
3. In a GUI or application framework, a window is (a) a class; (b) an instance of a model class; (c) an instance of a view class; (d) an instance of a controller class; (e) an event
4. In a model/view/controller architecture, program execution is driven by (a) events; (b) views; (c) models; (d) commands; (e) interrupts
5. The application programmer may use a pre-written user-interface library and write _____ handler methods to respond to different kinds of user input. (a) view; (b) string; (c) loop; (d) event; (e) mouse
6. An application programmer writes an application class (a) as a base class for the library's derived class; (b) as a class derived from the library's base class; (c) as a class that is not part of an inheritance relationship; (d) to implement a model; (e) to implement a view
7. (a) ; (b) ; (c) ; (d) ; (e) none of these

2. Bitmap and vector graphics

1. Drawing an image pixel by pixel (a) takes several seconds; (b) is bitmap rendering; (c) is vector rendering; (d) is impossible; (e) is always preferred
2. Drawing an image stored as instructions (a) takes several seconds; (b) is bitmap rendering; (c) is vector rendering; (d) is impossible; (e) is always preferred
3. Two advantages of vector over bitmap storage are (a) precision and compression; (b) precision and ease of editing; (c) compression and ease of editing; (d) compression and esthetics; (e) none of these
4. (a) ; (b) ; (c) ; (d) ; (e) none of these

3. Polymorphic collections of shapes

1. In polymorphism, the behavior of an object depends on whether (a) it is an instance of a base or derived class; (b) the object is statically or dynamically allocated; (c) it is dynamically allocated; (d) it has a reference member; (e) it has a private member method
2. (T-F) With polymorphism, a derived class may change the behavior of its ancestor class.
3. Polymorphism is implemented through _____ methods. (a) global; (b) base-class; (c) virtual; (d) access; (e) *void*
4. (T-F) In a mixed collection of graphical objects of different types, the same *draw* method would be used to display each shape.
5. The address of a call to a virtual method is resolved at (a) compile time; (b) the time a program is loaded; (c) the time the method actually executes; (d) the time the method terminates; (e) the time the program terminates
6. A virtual method is a member of _____ classes. (a) simple and complex; (b) container and contained; (c) base and derived; (d) application and view; (e) model and view
7. Determination of the call address of a member method call at run time rather than compile time is (a) inheritance; (b) encapsulation; (c) early binding; (d) late binding; (e) forward reference
8. An array of _____ makes possible a collection of shapes of mixed classes with polymorphic behavior. (a) integers; (b) method references; (c) character references; (d) base-class references; (e) none of these
9. (T-F) A data item may be polymorphic.
10. (T-F) Polymorphism is used only in cases where a base class has a single derived class.
11. (T-F) An array of references may sensibly point to instances of a variety of data types.
12. Polymorphism is an alternative to the use of (a) *if…else*; (b) *switch*; (c) *while*; (d) *goto*; (e) method calls
13. Virtual methods are used most often in (a) encapsulation; (b) data hiding; (c) memory management; (d) inheritance; (e) polymorphism
14. An abstract base class (a) has no virtual methods; (b) cannot be instantiated; (c) has a definition that may not be overridden; (d) has a null constructor; (e) inherits from a concrete base class
15. (a) ; (b) ; (c) ; (d) ; (e) none of these