

Midterm
100 points possible

For all questions, $\lg N$ or $\log N$ is the log base 2 of N .

For all questions, assume all ADT implementations and algorithms are good implementations.

Do NOT make any stray marks on your answer sheet. You may use pen or pencil, but if you want to change an answer, you MUST erase completely. Marks that are "X"ed out will be counted as filled. Your answer sheet will be scanned and graded automatically.

True/False questions. Please mark A for True and B for False on the answer sheet provided: (1 pt each) – 8 points total

1. $f(N) = N$ has a faster growth rate than $g(N) = \lg N$
2. Operator overloading is a good example of encapsulation
3. $*(anArray+1)$ is equivalent to $anArray[1]$
4. Assume you have a queue implemented with a singly linked list with head and tail pointers. Performing enqueue operations at the head and dequeue operations at the tail is more efficient than performing enqueue operations at the tail and dequeue operations at the head.
5. Late binding is a good example of polymorphism
6. A stack reverses the order in which items occur.
7. If class X publically inherits class Y, the following code is legal:
 $X\ x;$
 $Y\ y;$
 $X^*\ xPtr = \&y;$
8. The size () operation for a singly linked list can never run in $O(1)$ time because every node in the list must be traversed to count how many nodes are in the list.

Multiple choice questions. Please mark the answer on the answer sheet provided:
(2 pts each) – 44 points total

9. A doubly linked list is declared as:

```

class Node {
public:
    itemtype item;
    Node* next;
    Node* prev;
};

class List {
private:
    Node* head;
};
    
```

Which of the following segments of code removes the element pointed to by X from the doubly linked list, if it is assumed that X points to neither the first nor the last element in the list?

- A. X->prev->next = X->next;
X->next->prev = X->prev;
- B. X->prev->next = X->prev;
X->next->prev = X->next;
- C. X->prev->prev = X->next;
X->next->next = X->prev;
- D. X->prev->prev = X->prev;
X->next->next = X->next;
- E. X->prev = X->next;
X->next = X->prev;

10. Which Abstract Data Type (ADT) would be **best** for a word processor's implementation of the Undo button (multiple undos are allowed):

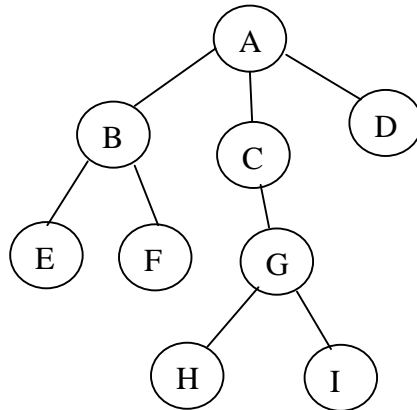
- A. Stack
- B. Queue
- C. List

11. Order the following equations in **increasing** growth rate:

$2n^3 - 6n$, $44n^{1.5} + 33n$, $5n^2 + 16n^2 \log n$, $17n^2 \log n + 31n^3$, $-3n^3 + 23n2^n$

- A. $5n^2 + 16n^2 \log n$, $17n^2 \log n + 31n^3$, $44n^{1.5} + 33n$, $-3n^3 + 23n2^n$, $2n^3 - 6n$,
- B. $44n^{1.5} + 33n$, $17n^2 \log n + 31n^3$, $5n^2 + 16n^2 \log n$, $2n^3 - 6n$, $-3n^3 + 23n2^n$
- C. $44n^{1.5} + 33n$, $5n^2 + 16n^2 \log n$, $17n^2 \log n + 31n^3$, $2n^3 - 6n$, $-3n^3 + 23n2^n$
- D. $44n^{1.5} + 33n$, $5n^2 + 16n^2 \log n$, $2n^3 - 6n$, $-3n^3 + 23n2^n$, $17n^2 \log n + 31n^3$
- E. None of the above

For the next 2 questions, use the following tree:



12. Which node/nodes is/are descendant(s) of C (Mark only one answer)

- | | | | |
|----|------------------|----|-------------------|
| A. | A | B. | E |
| C. | G | D. | D |
| E. | All of the above | F. | None of the above |

13. What is the path length of the path from A to I

- | | | | |
|----|---|----|-------------------|
| A. | 1 | B. | 2 |
| C. | 3 | D. | 4 |
| E. | 5 | F. | None of the above |
-

14. In the template prefix

template <class T>

what kind of variable is the parameter T?

- A. T must be a class
- B. T must not be a class
- C. T can be only types built into the C++ language
- D. T can be any type, whether built into the C++ language or defined by the programmer
- E. Both B and C are true
- F. Both A and D are true

Consider the following class definitions:

```
class Sphere {  
public:  
    virtual double getArea ();  
    void displayStatistics ();  
};
```

```
class Ball: public Sphere {  
public:  
    double getArea ();  
    virtual void displayStatistics();  
};
```

Suppose that the implementation of each version of `displayStatistics` invokes the function `getArea`. Given the statements:

```
Ball myBall;  
Sphere mySphere;  
Ball* ballPtr;  
Sphere* spherePtr;  
Sphere* spherePtr2;  
ballPtr = &myBall;  
spherePtr = &myBall;  
spherePtr2 = &mySphere;
```

15. Which version of `getArea` will the call `spherePtr->getArea()` invoke?

- A) the Sphere class `getArea` member function
- B) the Ball class `getArea` member function

16. Which version of `getArea` will the call `spherePtr2->getArea()` invoke?

- C) the Sphere class `getArea` member function
- D) the Ball class `getArea` member function

17. Suppose that you have a sorted array of 500 integers and suppose also that you want to use binary search to find a specific value. What is the maximum number of array elements that you actually have to look at until you find the value that you are looking for (or until you can tell for sure that the value you're looking for isn't in the array)?

- A. 8
- B. 9
- C. 10
- D. 500

18. The operation `enqueue()` affects

- A. the front of the queue.
- B. the back of the queue.
- C. both the front and the back of the queue.
- D. either the front or the back of the queue, but not both at the same time, depending on the user's needs.

19. The runtime complexity of an algorithm

- A. depends on the programming language being used.
- B. depends on how the data accessed by the algorithm is organized.
- C. depends on the speed of the computer on which the algorithm is being run.
- D. All of the above.
- E. None of the above.

20. Suppose that you are considering running two algorithms under the same conditions and on the same computer, on problems of size n . Furthermore, suppose that Algorithm A has a worst-case runtime complexity proportional to $\log n$ while algorithm B has a worst-case runtime complexity proportional to n . For large enough values of n ,

- A. Algorithm A is always faster than algorithm B, in the worst-case scenario.
- B. Algorithm A is always slower than algorithm B, in the worst-case scenario.
- C. Algorithm A may be faster or slower than algorithm B, in the worst-case scenario, depending on those pesky constants which we typically ignore when we use O-notation.
- D. We cannot conclude anything about this situation when n is large; we should be looking at the case when n is small.

21. Suppose I gave you a homework assignment in which you have to implement a hierarchical file system. Which data structure would be most appropriate for this assignment?

- A. A graph.
- B. A pointer-based list.
- C. An array-based stack.
- D. A pointer-based queue.
- E. A tree.
- F. Any of the above are equally appropriate

22. Lists may be either array-based or pointer-based (linked lists). Which of the following specify an **advantage** of an **array-based** implementation of a list over a pointer-based implementation of a list:

- A. Less space for each item
- B. Faster inserts and removes in the middle of the list
- C. No unused space
- D. A and B
- E. B and C

23. An uncaught exception (one that matches no catch blocks) will

- A. End the function where it was thrown
- B. Terminate the program
- C. Continue execution
- D. Do nothing
- E. None of the above
- F. Not enough information

24. Lists may be either array-based or pointer-based (linked lists). Which of the following specify an **advantage** of a **pointer-based** implementation of a list over an array-based implementation of a list:

- A. Less space for each item
- B. Faster inserts and removes in the middle of the list
- C. No unused space
- D. A and B
- E. B and C

For the next X questions, use the following answer choices:

- | | |
|----------------|----------------------|
| A. $O(1)$ | B. $O(N)$ |
| C. $O(N^2)$ | D. $O(N \log N)$ |
| E. $O(\log N)$ | F. None of the above |

Which of the above best represents the Big-Oh runtime for...

25. Search in an unsorted array?

26. Deleting the last element of a singly linked list with a tail pointer.

27. Linear search on a sorted array.

28. Printing the elements of an unsorted list in sorted order. (Do not sort the elements or use any auxiliary data structure).

29. Removing an item from a sorted array given the index of the item you want to remove (do not leave any empty spaces in the array)

30. replaceAtRank for a linked list implemented sequence?

31. (9 pts) Briefly define the following:

a) Static/early binding

b) Briefly define object oriented programming (OOP).

c) Big-Oh notation

32. (3 pts) Explain why the statement “The running time of algorithm A is at least $O(n^2)$,” is meaningless.

33. (3 pts) Is $2^{n+1} = O(2^n)$? Prove why or why not (I don't want just a yes or no answer).

34. (7 pts) *Explain* how to efficiently implement two stacks using one array of size `ARRAY_SIZE` in such a way that neither stack is full unless the total number of elements in both stacks together is `ARRAY_SIZE`. **Give the big-oh running time of the push and pop operations.** To get full points, you must describe the most time efficient implementation.

35. (8 pts) Whereas a stack allows insertion and deletion of elements at only one end, and a queue allows insertions at one end and deletion at the other end, a deque (double-ended queue) allows insertion and deletion at both ends. Write four $O(1)$ -time procedures to insert elements into and delete elements from both ends of a deque constructed from an array.

36. (6 pts) Given the following snippet of code: (There are no syntax errors in this code)

```
int* x = new int(5);  
int y = *x;  
int* z = x;  
(*x)++;  
int* a = 0;
```

Describe what the following statements will produce (either what will be printed or what will happen). If a memory address is printed, be as specific as you can by telling me what variable/value the memory address is associated with.

- a) `cout << *a << endl;`
- b) `cout << y << endl;`
- c) `cout << *z << endl;`
- d) `cout << *&y << endl;`
- e) `cout << a << endl;`
- f) `cout << z << endl;`

37. (12 pts) Write a function to reverse the nodes in a singly linked list with only a pointer to the head of the list (no last pointer) and no sentinel (dummy) node. Your function should be general in that it would work for any item type. **You must reverse the nodes of the list via pointer manipulations. You may not simply copy the item from one node to the next.** Remember to use good programming style. **Give the Big-Oh runtime of your code.** You may assume the following classes exist:

```
class List {  
private:  
    Node* head;  
public:  
    void reverse();  
};
```

```
class Node {  
private:  
    itemtype item;  
    Node* next;  
friend class List;  
};
```