

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: (2 pts each) – 18 points total

1. It is possible to implement a doubly linked list with only a tail pointer and no head pointer
2. A single linked list based stack can perform push and pop operations in $O(1)$ without a tail pointer
3. The following code is a correct implementation of "enqueue" for a singly linked list based queue. Assume the queue has three data members: head, tail, and length.

```
void Queue::Enqueue(int val) {  
    Node* newNode = new Node;  
    newNode->val = val;  
    newNode->next = head;  
    head = newNode;  
    length ++;  
}
```

4. A stack is a FIFO structure.
5. If x and y are real numbers such that $0 < x < y$ then n^x is $O(n^y)$ but n^y is not $O(n^x)$
6. 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.
7. Public member variables are a good example of encapsulation
8. An abstract data type is a data type that performs linked-list like operations
9. Object oriented programming involves user-defined objects only.

Multiple choice questions. Please mark the answer on the answer sheet provided:

(3 pts each) – 30 points total

10. A circular singly linked list

- A) has a fixed size
- B) has no special cases for insert and remove
- C) can be traversed in either direction
- D) A and B
- E) All of the above
- F) None of the above

11. Which of the following is NOT an example of a theoretical runtime analysis?

- A) Big-Omega
- B) Little-Omega
- C) Little-Oh
- D) Big-Theta
- E) Little-Theta
- F) All of the above are examples

12. A good example of polymorphism is/are

- A) Inheritance
- B) Operator overloading
- C) Function overloading
- D) All of the above
- E) None of the above

13. An inorder traversal of a binary search tree will visit

- A) $\lg N$ of the nodes
- B) N of the node in sorted order
- C) N of the nodes in an unknown order
- D) The minimum, maximum and root nodes only
- E) None of the above

14. An uncaught exception will:

- A) Continue execution as if nothing happened
- B) Terminate the program
- C) Print an error message
- D) Produce a segmentation fault
- E) Throw another exception
- F) Any of the above are possible. The behavior is undefined

15. A function Q has input requirements and defined side effects (outputs). I want to replace Q with another function P that has a faster run time. What must be true about P?

- A) P's input requirements can be more restrictive than Q's input requirements
- B) P's side effects can be more restrictive than Q's side effects
- C) P's input requirements and side effects must be identical to Q's
- D) P's input requirements can be less restrictive than Q's input requirements
- E) You can't replace the function

Use the following answer key for the questions in this section

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

16. What is the Big-Oh running time of the following snippet of code

```
void f( int n ) {  
    for ( int i = 0; i < n; i ++ ) {  
        for ( int j = n; j > 1; j /= 2 ) {  
            // Something O(1)  
        }  
    }  
}
```

17. What is the Big-Oh running time of the following snippet of code

```
void f( int n ) {  
    for ( int i = 0; i < n; i ++ ) {  
        for ( int j = 0; j < 10; j ++ ) {  
            // Something in O(1)  
        }  
    }  
}
```

18. What is the average case running time for insert into a binary search tree?

A self-adjusting list is like a regular list, except that all insertions are performed at the head, and when an element is accessed through Search, it is moved to the head of the list without changing the relative order of the other items. The search function should find and move all occurrences of the item.

19. What is the Big-Oh runtime for search in an array implemented self-adjusting list?

20. What is the Big-Oh runtime for search in a linked implemented self-adjusting list?

21. (11 pts) Given the following snippet of code:

```
int* x = new int[5];
int y = *x;
int* z = &y;
y = 1;
int a = *z;
(*x) ++;
(*z) --;
int* b = 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 << &z << endl;`

b) `cout << *(x+1) << endl;`

c) `cout << y << endl;`

d) `cout << &y << endl;`

e) `cout << z << endl;`

f) `cout << *(&z) << endl;`

g) `cout << &(*z) << endl;`

h) `cout << *b << endl;`

i) `cout << x << endl;`

j) `cout << *z << endl;`

k) `cout << a << endl;`

22. (18 pts) Write a function to calculate and return the number of nodes in a binary search tree. You may assume the following classes exist:

```
class Tree {  
private:  
    Node* root;  
  
public:  
    int size ( ) { return size(root);}  
};  
  
class Node {  
friend class Tree:  
private:  
    Node* parent;  
    Node* left;  
    Node* right;  
};
```

Hint: You may want to overload the size function like you are doing in your binary search tree assignment.

23. (25 pts) Given a doubly linked list with a head pointer, write a function to find and swap the nodes with the smallest and the largest values. You must physically move the nodes via pointer operations and not simply copy the value from one node to the next. You may **NOT** declare any node objects but you may use node pointers. **What is the big-oh runtime of your function?** You may assume the following classes exist and that the $<$, $<=$, $==$, $>$, and $>=$ operators have been overloaded for *itemtype*. You may write as many auxiliary functions as you like. The list may contain duplicate values.

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

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