Name: _____

Write all of your responses on these exam pages. If you need extra space please use the backs of the pages. The Short Answer questions are worth 10 points each. The Coding exercise is worth 50 points.

1 Short Answer

1. State the precise mathematical definitions of Big-O, Big-Ω, and Big-Θ. Also give the common meaning of each, specifically, what bound does it indicate?

2. Fill out the time complexity table below.

Algorithm	**Best**	**Average**	**Worst**
Bubble Sort			
Insertion Sort			
Selection Sort			
Quick Sort			
Merge Sort			
Tree Sort with BST			
Linear Search on Array			
Binary Search on Sorted Array			

3. Write a recursive function that will compute the double factorial. The double factorial is defined as

$$n!! = n \cdot (n-2) \cdot (n-4) \cdots 1$$

and $0!! = 1$. For example, $3!! = 3$, $4!! = 8$, $5!! = 15$, $6!! = 48$, $7!! = 105$, ...

4. Write a templated recursive binary search function for an array, assume the array is already sorted.

5. Write four functions to be added to the (singularly linked) LinkedList class, the specifications to these are below. Your implementation should be written as functions that are outside the specification.

```
void displayListRec();
void displayListRecRev();
void displayListRec(ListNode<T> *t);
void displayListRecRev(ListNode<T> *t);
```

- The functions `displayListRec()` and `displayListRec(ListNode<T> *t)` work together to print out the list to the console in order.
- The functions `displayListRecRev()` and `displayListRecRev(ListNode<T> *t)` work together to print out the list to the console in reverse order.
- `displayListRec()` is non-recursive, public, and does not print anything directly to the console. It simply does the appropriate call to `displayListRec(ListNode<T> *t)`.
- `displayListRec(ListNode<T> *t)` is recursive, private, and prints the data to the console.
- `displayListRecRev()` is non-recursive, public, and does not print anything directly to the console. It simply does the appropriate call to `displayListRecRev(ListNode<T> *t)`.
- `displayListRecRev(ListNode<T> *t)` is recursive, private, and prints.

With these added to the LinkedList class the following program will produce the following output. The data of the ListNode is stored in field named `value`.

```
int main() {
    LinkedList<int> list;
    list.appendNode(7);
    list.appendNode(2);
    list.appendNode(4);
    list.appendNode(1);
    list.appendNode(9);
    list.appendNode(8);
    list.displayListRec();
    cout << endl;
    list.displayListRecRev();
    cout << endl;
    return 0;
}
```

Output:

```
7 2 4 1 9 8
8 9 1 4 2 7
```

2 Coding Exercise

This exercise is to code portions of a general templated binary search tree. The specification for the class is below.

```
template <class T> class BinaryTree {
  private:
    class TreeNode {
      public:
        T value;
        TreeNode *left;
        TreeNode *right;

        TreeNode(T nodeValue) {
            value = nodeValue;
            left = nullptr;
            right = nullptr;
        }
    };

    TreeNode *root;

    void insert(TreeNode *&, TreeNode *&);
    void destroySubTree(TreeNode *);
    void deleteNode(T, TreeNode *&);
    void makeDeletion(TreeNode *&);
    void displayInOrder(TreeNode *) const;
    int numberOfNodesRec(TreeNode *);

  public:
    BinaryTree();
    ~BinaryTree();
    BinaryTree(const BinaryTree &obj);
    const BinaryTree operator=(const BinaryTree &right);

    void displayInOrder() const;
    void insertNode(T);
    bool searchNode(T);
    void remove(T);
    int numberOfNodes();
};
```

- Constructor, destructor, copy constructor, and overloaded assignment do their usual jobs.
- `destroySubTree` removes the subtree starting at the input node.
- `displayInOrder` and its recursive counterpart prints the tree contents to the console using an in-order traversal of the tree.
- `insertNode` and its recursive counterpart inserts the item in the correct place in the binary search tree.
- `searchNode` returns true if the item is in the tree and false if not.
- `remove` invokes the `deleteNode` and `makeDeletion` functions to remove the item from the tree. `deleteNode` recursively finds the node to delete and `makeDeletion` does the actual deletion of the node.
- `numberOfNodes` and its recursive counterpart counts the total number of nodes in the tree.
- There are, of course, to be no memory leaks.
- If you find the need to add in another function, feel free to do so but you must, of course, write the implementation of the functions you add.
- No inline code for these implementations.

Your code for the constructor, destructor, and `destroySubTree`.

Your code for the copy constructor and overloaded assignment operator.

Your code for the `displayInOrder` and its recursive counterpart.

Your code for the `insertNode` and its recursive counterpart.

Your code for the `searchNode` function.

Your code for the `remove` function and for its support functions `deleteNode` and `makeDeletion`.

Your code for the `numberOfNodes` and its recursive counterpart.