1 Short Answer (8 Points Each)

1. State the definitions of Big-O, Big-Ω, and Big-Θ.

   Solution:

   - Big-O (Upper Bound): A function $f(x)$ is $O(g(x))$ if and only if there exist a constant C and a constant k such that, for every $x > k$, $|f(x)| \leq C|g(x)|$.
   - Big-Ω (Lower Bound): A function $f(x)$ is $\Omega(g(x))$ if and only if there exist a constant $C > 0$ and a constant k such that, for every $x > k$, $|f(x)| \geq C|g(x)|$.
   - Big-Θ (Tight Bound): A function $f(x)$ is $\Theta(g(x))$ if and only if there exist constants $C_1 > 0$ and C_2 and a constant k such that, for all $x > k$, $C_1|g(x)| \leq |f(x)| \leq C_2|g(x)|$.

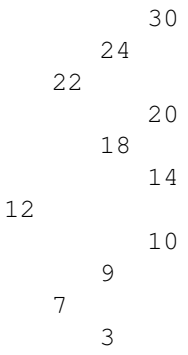
2. Fill out the time complexity table below.

   Solution:

Algorithm	**Best**	**Average**	**Worst**
Bubble Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Insertion Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Selection Sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$
Quick Sort	$\Omega(n\log(n)$	$\Theta(n\log(n))$	$O(n^2)$
Merge Sort	$\Omega(n\log(n)$	$\Theta(n\log(n))$	$O(n\log(n)$

3. Suppose the following values are inserted into a binary tree, in the order given: 12, 7, 9, 10, 22, 24, 30, 18, 3, 14, 20

   (a) Draw a diagram of the resulting binary tree.
   Solution:



   (b) Display an in-order traversal of the tree.
   Solution: 3 7 9 10 12 14 18 20 22 24 30

   (c) Display a pre-order traversal of the tree.
   Solution: 12 7 3 9 10 22 18 14 20 24 30

   (d) Display a post-order traversal of the tree.
   Solution: 3 10 9 7 14 20 18 30 24 22 12

4. Write a recursive function to calculate the factorial of a parameter input. Recall that the factorial is defined as, $0! = 1$ and for $n > 0$ we have $n! = 1 \cdot 2 \cdot 3 \cdots (n-1) \cdot n$

   Solution:

```
int factorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * factorial(n - 1);
}
```

5. Write a recursive function to calculate the n^{th} Fibonacci number. Recall that the Fibonacci sequence is defined as, $F_0 = 1$, $F_1 = 1$, and for $n > 1$ we have $F_n = F_{n-1} + F_{n-2}$.

   Solution:

```
int fib(int n)
{
    if (n <= 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return fib(n - 1) + fib(n - 2);
}
```

6. Write a recursive function to solve the Towers of Hanoi problem. The setup and initial function call are below.

```
const int FROM_PEG = 1;     // Initial "from" peg
const int TO_PEG = 3;       // Initial "to" peg
const int TEMP_PEG = 2;     // Initial "temp" peg
moveDiscs(NUM_DISCS, FROM_PEG, TO_PEG, TEMP_PEG);
```

   Solution:

```
void moveDiscs(int num, int fromPeg, int toPeg, int tempPeg)
{
    if (num > 0)
    {
        moveDiscs(num - 1, fromPeg, tempPeg, toPeg);
        cout << "Move a disc from peg " << fromPeg << " to peg " << toPeg << endl;
        moveDiscs(num - 1, tempPeg, toPeg, fromPeg);
    }
}
```

7. Answer the following questions about inheritance.

   (a) What is the difference between a protected class member and a private class member?
   Solution: Protected members can be used by derived classes bit private members cannot.

   (b) Which constructor is called first, that of the derived class or the base class?
   Solution: Base class.

   (c) When does static binding take place? When does dynamic binding take place?
   Solution: Static binding happens at compile time, this is the case for all non-virtual functions. Dynamic binding happens at run time, this is the case for all virtual functions.

   (d) What is an abstract base class?
   Solution: An abstract base class is created when the base class contains one or more purely virtual functions.

8. Answer the following True and False questions about inheritance. Circle the correct answer.

   (a) **TRUE:** The base class destructor is called after the derived class destructor.

   (b) **FALSE:** Protected members of a public base class become public members of the derived class.

   (c) **FALSE:** It isn't possible for a base class to have more than one constructor.

   (d) **TRUE:** Pointers to a base class may be assigned the address of a derived class object.

9. Find and correct the errors in the following code.

```
class Three : public Two : public One
{
  protected:
    int x;
  public:
    Three() { y = 0 }
    Three(int a, int b, int c), Two(b), Three(c)
      { x == a; }
    ~Two();
};
```

Solution:

```
class Three : public Two, public One
{
  protected:
    int x;
  public:
    Three() { x = 0; }
    Three(int a, int b, int c): Two(b), One(c)
      { x = a; }
    ~Three();
};
```

10. What is a purely virtual function and how is it created?

Solution: A purely virtual function is a function in a base class that must be overridden by all derived classes. To create a purely virtual function the base class will not implement the function and an `= 0` is placed at the end of the function declaration. When one creates a purely virtual function, the base class becomes abstract and you can no longer instantiate a variable to the base class type.

2 Coding (10 Points Each)

Given the following specification for the BinaryTree class and TreeNode class.

```
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;
    void displayPreOrder(TreeNode *) const;
    void displayPostOrder(TreeNode *) const;

public:
  BinaryTree() {
      root = nullptr;
  }

  ~BinaryTree() {
      destroySubTree(root);
  }

  void displayInOrder() const {
      displayInOrder(root);
  }

  void displayPreOrder() const {
      displayPreOrder(root);
  }

  void displayPostOrder() const {
      displayPostOrder(root);
  }

  void clear() {
      destroySubTree(root);
      root = nullptr;
  }

  void insertNode(T);
  bool searchNode(T);
  void remove(T);
};
```

Write the implementations of the following functions as they would be written in the cpp file.

- `void displayInOrder(TreeNode *) const`
- `void insert(TreeNode *&, TreeNode *&)`
- `bool searchNode(T)`

---

1. `void displayInOrder(TreeNode *) const`

   Solution:

   ```
   template <class T>
   void BinaryTree<T>::displayInOrder(TreeNode *nodePtr) const
   {
       if (nodePtr)
       {
           displayInOrder(nodePtr->left);
           cout << nodePtr->value << endl;
           displayInOrder(nodePtr->right);
       }
   }
   ```

2. `void insert(TreeNode *&, TreeNode *&)`

   Solution:

   ```
   template <class T>
   void BinaryTree<T>::insert(TreeNode *&nodePtr, TreeNode *&newNode)
   {
       if (nodePtr == nullptr)
           nodePtr = newNode;                  // Insert the node.
       else if (newNode->value < nodePtr->value)
           insert(nodePtr->left, newNode);     // Search the left branch
       else
           insert(nodePtr->right, newNode);    // Search the right branch
   }
   ```

3. `bool searchNode(T)`

   Solution:

```
template <class T>
bool BinaryTree<T>::searchNode(T item)
{
    TreeNode *nodePtr = root;

    while (nodePtr)
    {
        if (nodePtr->value == item)
            return true;
        else if (item < nodePtr->value)
            nodePtr = nodePtr->left;
        else
            nodePtr = nodePtr->right;
    }
    return false;
}
```