Name: _____

Write all of your responses on these exam pages. If you need extra space please use the backs of the pages.

1 Short Answer (10 Points Each)

1. Write a function that will take as input a pointer to an integer array and the size of the array and return a pointer to a duplicate of the input array where the entries of the original array are reversed. The input array is not to be altered by the function. In this function, all array access is to be done with pointer references and dereferences, no bracket notation for any array access.

2. Write a recursive function to calculate the number of ways to select k elements out of a set of n elements (n choose k). Recall that the definition of n choose k is

$$\begin{pmatrix} n \\ k \end{pmatrix} = \begin{pmatrix} n-1 \\ k \end{pmatrix} + \begin{pmatrix} n-1 \\ k-1 \end{pmatrix}$$

   If n or k is 0 then the value is 1, if $n = k$ the value is 1, and we will define anything outside these ranges the value is 0. That is, if n or k are negative or if $k > n$ we will have the program return 0.

3. Write a templated function that will sort an array of the templated type using either the bubble sort, insertion sort, or selection sort. State all of the operators that must be overloaded for the function to work.

4. Write a function that will take one integer input parameter n and allocate an array of doubles of size n. If the system runs out of memory have the function throw a string exception with an out of memory error. The function should return a pointer to the array if created.

5. Explain the concept of *backtracking* and how it is used in the solution to the 8-Queens problem.

6. Give three instances of when a copy constructor is called?

7. Write a function that takes in an integer array, assumed to be sorted in ascending order, and an integer target value. The function is to use the binary search to search the array for the target value. If the target is found the function should return the position of the value in the array and if the element is not in the array the function should return -1.

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

9. Fill out the time complexity table below.

Algorithm	**Best**	**Average**	**Worst**
Bubble Sort			
Insertion Sort			
Selection Sort			
Quick Sort			
Merge Sort			
Tree Sort			
Linear Search			
Binary Search			

2 Coding (20 Points Each)

1. Given the following declaration of the complex number class and the definitions of the arithmetic operations that follow. Write the implementations for the copy constructor and all four arithmetic operations.

```
class Complex
{
  private:
    double real;  //  Real Part
    double imag;  //  Imaginary Part

  public:
    // Constructors and Destructor
    Complex(double a = 0, double b = 0);
    Complex(const Complex &obj);

    // Overloaded operator functions
    Complex operator + (const Complex &);
    Complex operator - (const Complex &);
    Complex operator * (const Complex &);
    Complex operator / (const Complex &);
};
```

If we have two complex numbers $a + bi$ and $c + di$ then the four arithmetic functions are defined as,

$$\begin{aligned}
(a+bi)+(c+di) &= (a+c)+(b+d)i \\
(a+bi)-(c+di) &= (a-c)+(b-d)i \\
(a+bi)\cdot(c+di) &= (ac-bd)+(bc+ad)i \\
(a+bi)/(c+di) &= \frac{ac+bd}{c^2+d^2}+\frac{bc-ad}{c^2+d^2}i
\end{aligned}$$

2. Given the declaration of the integer Linked List class below write the implementation of the insertOrdred and removeElement functions.

   void insertOrdred(int) Inserts the parameter item into the list so that smaller values are before the inserted item.

   void removeElement(int) Searches for the first occurrence of the input parameter in the list. If one is found the item is deleted from the list and if it is not found the list is unaltered.

```
class ListCollection
{
  private:

    class ListNode
    {
      public:
        int value;    // The value in this node
        ListNode *next;  // To point to the next node

        ListNode(int nodeValue = 0)
        {
            value = nodeValue;
            next = nullptr;
        }
    };

    ListNode *head;

  public:
    ListCollection();
    ~ListCollection();

    void insertOrdred(int);
    void removeElement(int);
};
```

3. Given the declaration of the integer Linked List class below write the implementation of the push, pop, enqueue, and dequeue functions so that this class could be used as either a stack or queue.

```
class IntStackQueue
{
  private:

    class ListNode
    {
      public:
        int value;    // The value in this node
        ListNode *next;  // To point to the next node

        ListNode(int nodeValue = 0)
        {
            value = nodeValue;
            next = nullptr;
        }
    };

    ListNode *head;

  public:
    ListCollection();
    ~ListCollection();

    void push(int);
    int pop();
    void enqueue(int);
    int dequeue();
};
```

4. Given the following definition of the GradedActivity class do the following.

   (a) Write the syntax to change the `getLetterGrade` function into a purely virtual function.

   (b) Create a Homework class derived off of the GradedActivity class that has one constructor that will take in the score and work as a default constructor setting the score to 0. The getLetterGrade override should return S (for satisfactory) if the score is 20 or greater and U (for unsatisfactory) otherwise.

   (c) Create an ScoredExam class derived off of the GradedActivity class that has one constructor that will take in the score and work as a default constructor setting the score to 0. The getLetterGrade override should return an A, B, C, D, F letter grade on the 90-80-70-60 scale.

   (d) Create a PassFailExam class derived off of the GradedActivity class that has one constructor that will take in the score and work as a default constructor setting the score to 0. The getLetterGrade override should return P if the score is greater than or equal to 70 and F otherwise.

   (e) Write a main program that uses a vector of GradedActivity pointers to store the grades of the student. The main should create a graded activity of each type, Homework, ScoredExam, and PassFailExam. Set the score for each assignment to an appropriate value and load the assignment into the vector. Then the program should run through the vector calling the getLetterGrade for each so that polymorphism will call the correct version of the function.

```
class GradedActivity
{
  protected:
    double score;

  public:
    // Default constructor
    GradedActivity()
    {
        score = 0.0;
    }

    // Accessor functions
    double getScore() const
    {
        return score;
    }

    void setScore(double s)
    {
        score = s;
    }

    char getLetterGrade() const;
};
```

---

5. Write a templated array merge function for two arrays into one. The declaration is below.

```
void merge(T A[], T B[], int sizeA, int sizeB, T M[])
```

The arrays A and B are assumed to be sorted in ascending order and the array M is to be the merge of A and B so that after M is initially populated with the values from A and B it is already sorted. So if A is `3 9 17 25 32` and B is `1 9 10 15 25 27 41` then the array M will be `1 3 9 9 10 15 17 25 25 27 32 41` after the merge. This is similar to the merge function from the merge sort algorithm. You may assume that the array M is already declared and of the correct size.

6. 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;

    void nodeCountRec(TreeNode *node, int &count);
    void leafCountRec(TreeNode *node, int &count);
    void heightRec(TreeNode *node,int &max,int h);

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);

    int nodeCount();
    int leafNodeCount();
    int treeHeight();
};
```

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

- `void nodeCountRec(TreeNode *node, int &count)` — Which will recursively count the number of nodes in the tree. The number of nodes will be stored in the variable count.
- `void leafCountRec(TreeNode *node, int &count)` — Which will recursively count the number of leaf nodes in the tree. The number of leaf nodes will be stored in the variable count.