Name: ______________________________

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

1. (*15 points*) Pointers & Dynamic Arrays: Write a function that takes as parameters the pointers of two sorted integer arrays and the two array sizes, and returns the pointer of a new integer array that is the sorted merging of the two arrays. For example, if the two parameter arrays are $[3, 7, 10, 16, 22]$ and $[1, 2, 5, 10, 12, 17]$ the resulting array is $[1, 2, 3, 5, 7, 10, 10, 12, 16, 17, 22]$.

2. (*15 points*) Basic Classes: Write the specification and implementation (no inline code) for a Rectangle class that stores a height and width (decimal values possible), constructor that takes the height and width as well as a default constructor that sets height and width to 1 each. The non-default constructor should error check the input so that any negative input is changed to 0. Accessors and mutators, mutators should do the same error checking. A member function for the area and a member function for the perimeter.

3. (*20 points*) Inheritance & Polymorphism: In this exercise we construct class structures, `automobile`, `car`, and `truck`. `automobile` is to be the base class and the other two will be derived classes off of this. For the mutators listed below you do not need to do any error checking on the values.

   - `automobile` is to store the name of the auto that can be set by its constructor but defaults to “auto”. It also stores the mileage of the vehicle. It also has a destructor, member functions to access and change the name and mileage, and a virtual function `toString` that will print out the name and mileage.
   - `car` inherits off of `automobile`, it further stores the number of doors the car has. It has a constructor allowing input of a name and a default that uses “car” as the name. It also has a destructor and accessor and mutator for the doors. Its toString function will, in addition to printing out the name and mileage, will print out the number of doors.
   - `truck` inherits off of `automobile`, it further stores the length of the truck bed. It has a constructor allowing input of a name and a default that uses “truck” as the name. It also has a destructor and accessor and mutator for the bed length. Its toString function will, in addition to printing out the name and mileage, will print out the length of the bed.

   With these classes and functions in place the following program will produce the output below.

```
#include <iostream>
#include <vector>

#include "auto.h"
#include "car.h"
#include "truck.h"

using namespace std;

int main() {
    vector<automobile *> autos;
    car *carptr = new car;
    carptr->setDoors(4);
    carptr->setMiles(80000);
    autos.push_back(carptr);

    truck *truckptr = new truck("Sport Pickup");
    truckptr->setBedLength(6);
    truckptr->setMiles(35000);
    autos.push_back(truckptr);

    truckptr = new truck;
    truckptr->setBedLength(8);
    truckptr->setMiles(125300);
    autos.push_back(truckptr);

    carptr = new car("Corvette");
    carptr->setDoors(2);
    carptr->setMiles(15200);
    autos.push_back(carptr);

    for (auto v : autos)
        cout << v->toString() << endl;

    return 0;
}
```

   Output:

```
car  Miles: 80000  Doors: 4
Sport Pickup  Miles: 35000  Bed Length: 6
truck  Miles: 125300  Bed Length: 8
Corvette  Miles: 15200  Doors: 2
```

automobile class code

car class code

truck class code

4. (*20 points*) Operator Overloading: Say we have a class structure called `Point` that represents a point (x, y, z) in three dimensions. The data, x, y, and z are stored in an array of three doubles, specifically, `double P[3];` is its declaration. Write the following overloads for the $+$, $-$ and * operators. Give both the specification and implementation of each with no inline code. You may assume that the `Point` class has a non-default constructor that takes three double parameters, specifically, `Point p(x, y, z);` is a valid creation of the point `p`.

   - The addition of two points is another point and is done by adding the respective x, y, and z values. So in mathematical notation, $(x_1, y_1, z_1) + (x_2, y_2, z_2) = (x_1 + x_2, y_1 + y_2, z_1 + z_2)$.
   - The subtraction of two points is another point and is done by subtracting the respective x, y, and z values. So in mathematical notation, $(x_1, y_1, z_1) - (x_2, y_2, z_2) = (x_1 - x_2, y_1 - y_2, z_1 - z_2)$.
   - Multiplying a number and a point (called scalar multiplication) is just multiplying each entry by that number. So in mathematical notation, $a \cdot (x_1, y_1, z_1) = (a \cdot x_1, a \cdot y_1, a \cdot z_1)$. This can also be done on either side of the point, that is, $(x_1, y_1, z_1) \cdot a = (a \cdot x_1, a \cdot y_1, a \cdot z_1)$.

5. (*30 points*) Linked Lists:

   (a) Write an `appendNode` function for the templated linked list. The function will take in a single parameter of the templated type, create the node of `ListNode` type and attach it to the end of the list. The start of the list is a pointer called `head`. This implementation of the linked list does not have a tail pointer.

(b) Write a copy constructor for the templated linked list class. Assume the class name is `LinkedList`.

(c) Write a `deleteNode` function for the templated linked list. The function will take in a single parameter of the templated type and remove the first occurrance of that value from the list. If the value is not in the list then the list is unaltered. The start of the list is a pointer called `head`. This implementation of the linked list does not have a tail pointer.

6. (*10 points*) Stacks, Queues, and STL: Write a function called `balanced` that takes in a string (which represents a mathematical expression) and uses an STL stack to determine if the parentheses are balanced. The function will return true if the parentheses are balanced and false otherwise. For example, an input of `(x+4/(x-1))*(y+7)/(z*(x + 4*y))` would return true and an input of `x+4/(x-1))` would return false.

7. (*20 points*) Recursion:

   (a) The D sequence is defined to be $D(1) = 1$, $D(2) = 1$, and

   $$D(n) = D(D(n-1)) + D(n-1-D(n-2))$$

   Write a recursive function D that takes in a single long parameter n and returns a long that is the value of $D(n)$.

(b) A permutation on a set is a rearrangement of the contents of that set. So if our set is $\{1, 2, 3\}$ then the set of all permutations is

```
1  2  3     1  3  2     2  1  3     2  3  1     3  1  2     3  2  1
```

We can find the set of all permutations on $\{1, 2, \ldots, n\}$ recursively by the following method. Put the numbers $1, 2, \ldots, n$ into a vector keep the first element as is and then recurse to find all the permutations of the rest of the numbers. Then replace the first number with the second and recurse on the rest, then swap the first number and the third and recurse, until all the entries were swapped.

For example, with three. Start with `1 2 3`, keep the 1 in its position recurse on the rest `2 3`, this will produce `2 3` and `3 2`, since the 1 did not change position we get the permutations `1 2 3` and `1 3 2`. Now we swap the 1 and 2 to get `2 1 3` and recurse to get `1 3` and `3 1`, so with the 2 gives us `2 1 3` and `2 3 1`. Now swap the 1 and 3 for `3 2 1` and recurse to get `2 1` and `1 2`, so with the 3 gives us `2 1 2` and `3 2 1`. Which is all of them.

Write a recursive function,

```
void permute(vector<int> permList, int index)
```

that will use the above method to permute the vector contents from the `index` to the end of the list. Once the index is the same as the size of the list you have a permutation so print out the contents of the vector.

8. (*25 points*) Binary Search Trees:

   (a) Write the recursive insert function for the templated binary search tree class. This function is to take a pointer to a tree node and a pointer to a new node and insert the new node in the correct position in the BST. That is, if we were to run the following code, the insert function will put this new node in the correct position in the tree.

   ```
   TreeNode *newNode = new TreeNode;
   newNode->value = item;
   newNode->left = newNode->right = nullptr;
   insert(root, newNode);
   ```

   (b) Write an in-order traversal function for the templated binary search tree class that displays the node value to the console screen.

(c) Write the three templated functions,

```
void deleteNode(T, TreeNode *&);
void makeDeletion(TreeNode *&);
void remove(T);
```

That together will delete an item from a binary search tree.

9. (*20 points*) Complexity:

   (a) 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?

   (b) 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			

(c) Prove that $T(n) = 2^{\sqrt{\lg n}}$ is $O(n^a)$ for any constant $a > 0$.

10. (*15 points*) Heaps: Given the specification for the Heap class below,

```
template <class T> class Heap {
  private:
    vector<T> data;
    int parent(int i) { return (i - 1) / 2; }
    int left(int i) { return 2 * i + 1; }
    int right(int i) { return 2 * i + 2; }
    void Heapify(int);
  public:
    Heap(){};
    void insert(T);
    T dequeue();
};
```

Write `insert`, `dequeue`, and `Heapify` without using functions from the algorithm library.

11. (*10 points*) Sorts: Write the templated Quick Sort functions to sort an array.

12. (*10 points*) Function Pointers: Consider the code segment below. Do not write the doubleArray, negateArray, or the sortArray functions, they are only there to make the rest of the main make sense.

    Write the templeted `apply` function, both the prototype and implementation, that will bring in three parameters. The first is an array of the templated type, the second is an integer representing the size of the array, the third is a pointer to a function that would support the doubleArray, negateArray, and sortArray functions. The implementation of the `apply` function is to call the function that the function pointer parameter is pointing to. So in the code below, `apply(A, sz, doubleArray);` would call the doubleArray function with parameters A and sz. The `apply(A, sz, sortArray);` would call the sortArray function with parameters A and sz.

```
#include <cstdlib>
#include <ctime>
#include <iostream>

using namespace std;

template <class T> void doubleArray(T[], int);
template <class T> void negateArray(T[], int);
template <class T> void sortArray(T[], int);

//  The prototype of the apply function would go here.

int main() {
    srand(time(0));
    int sz = 0;

    cout << "Input the number of values to store: ";
    cin >> sz;
    int *A = new int[sz];

    for (int i = 0; i < sz; i++)
        A[i] = rand();

    apply(A, sz, doubleArray);
    apply(A, sz, negateArray);
    apply(A, sz, sortArray);

    delete[] A;
    return 0;
}

// Implementation of the apply function would go here.
```