PROBLEM 1:  (Sorting Match-up: (9 pts))

Below is the state of a set of numbers after each step of a sort. Initially the numbers are in random order (Step 0), and they progress to being sorted. For this problem, match the name of a sort to its steps below. Note that not all sorts listed will be matched to a set of steps.

Given these sorts that we studied in class:

- Bubblesort
- Quicksort
- Mergesort
- Heapsort
- Bucket sort
- Radix sort

Match them with these sorting steps:

Step 0:  423  471  462  513  629  9  683  419
Step 1:  471  462  423  513  683  629  9  419
Step 2:  9  513  419  423  629  462  471  683
Step 3:  9  419  423  462  471  513  629  683

- Which sort is this?

Step 0:  423  471  462  513  629  9  683  419
Step 1:  423  471  462  513  629  9  683  419
Step 2:  423  471  683  513  629  9  462  419
Step 3:  423  629  683  513  471  9  462  419
Step 4:  683  629  462  513  471  9  423  419
Step 5:  629  513  462  419  471  9  423  683
Step 6:  513  471  462  419  423  9  629  683
Step 7:  471  423  462  419  9  513  629  683
Step 8:  462  423  9  419  471  513  629  683
Step 9:  423  419  9  462  471  513  629  683
Step 10: 419  9  423  462  471  513  629  683
Step 11: 9  419  423  462  471  513  629  683

- Which sort is this?

Step 0:  423  471  462  513  629  9  683  419
Step 1:  423  471  462  513  629  9  683  419
Step 2:  423  462  471  513  629  9  683  419

- Which sort is this?

Step 3:  423  462  471  513  9  629  683  419
Step 4:  423  462  471  513  9  629  419  683
Step 5:  423  462  471  513  9  419  629  683
Step 6:  9  419  423  462  471  513  629  683

PROBLEM 2:  (Analysis: (8 pts))

Consider the most efficient implementation for each of the following operations.

- Consider operations applied to a balanced binary search tree with N nodes. What is the worst case time (big-Oh) of the operations?
  - Delete the minimum:
- List the nodes in reverse sorted order:

- Consider operations applied to a *balanced binary tree* with N nodes. What is the worst case time (big-Oh) of the operations?
  - Delete the minimum:
  - List the nodes in reverse sorted order:

PROBLEM 3: (Queued up: (18 pts))

Consider implementing a queue of Persons with a linked list, and consider the Person struct, and Queue class definitions and constructors shown on the Exam 2 Handout. Note that the linked list is hidden from the client in the private section.

For example, the following is a picture of a queue with 5 elements. Note that the smaller boxes represent a PersonNode and the larger boxes represent a Person.

![Queue diagram]

PART A (2 pts):
Write the member function `IsEmpty` whose header is given below. `IsEmpty` returns true if the queue is empty, false if the queue is not empty.

```cpp
bool Queue::IsEmpty() const
// postcondition: returns true if Queue is empty, false otherwise
{
}
```

PART B (8 pts):
Write the function `Enqueue` whose header is shown below. `Enqueue` inserts a Person into the queue (at the end of the linked list).

```cpp
void Queue::Enqueue(const Person & item)
// precondition: Queue is [e1, e2, ..., en] with n >= 0
// postcondition: Queue is [e1, e2, ..., en, item]
{
}
```
PART C (2 pts): If N is the number of Persons in the queue, what is the worst case running time (big-Oh) of the Enqueue function you wrote?

PART D (6 pts):
Write the function Dequeue whose header is shown below. Dequeue removes and returns the first person in the queue. Memory removed from the queue and no longer used should be returned to the memory heap.

```cpp
void Queue::Dequeue(Person & item)
// precondition: queue is not empty
// postcondition: first element in queue is removed and returned.
{
}
```

PROBLEM 4: (Printing Families: (12 pts))

Consider the definition of a GenNode for a general tree, shown on the Exam 2 Handout.

PART A (6 pts):
Write the function PrintFamily whose header is shown below. PrintFamily prints the value at a node, followed by a colon, followed by the names of its children all on one line.

For example, consider the general tree shown below.

```
Bob

Mary

Mel Gwen Joe

Paul

Doug

Eric Tom Bill Pete

Sue Beth Ellen Zoe Tim Don Lewis
```

If T is pointing to the root of the tree above, the output for the three calls PrintFamily(T), PrintFamily(T->child), and PrintFamily(T->child->sibling->sibling) would be:

Bob: Mary Alice Doug
Mary: Mel Gwen Joe
Doug: Eric Tom Bill Pete

Complete function PrintFamily below the following header.

```cpp
void PrintFamily(GenNode * T)
// precondition: T is not NULL
// postcondition: prints T and it’s children on one line
{
```
PART B (6 pts):
Write the function \textit{PrintAllFamilies} whose header is shown below. For each node in the tree, \textit{PrintAllFamilies} prints the value at the node, a colon, and a list of the node’s children all on one line.

If \textit{T} is pointing to the root of the tree in Part A, the output for the call \textit{PrintAllFamilies}(T) is partially shown below. The order the families are printed in may be different than the order shown:

Bob: Mary Alice Doug
Mary: Mel Gwen Joe
Mel:
Gwen: Sue
Sue:
Joe:
Alice: Paul
...

In writing \textit{PrintAllFamilies}, you may call the function \textit{PrintFamily} you wrote in Part A. Assume \textit{PrintFamily} works correctly, regardless of what you wrote in Part A.

Complete function \textit{PrintAllFamilies} below the following header. Assume \textit{PrintAllFamilies} is initially called on a tree (the root of the tree does not have a sibling).

\begin{verbatim}
void PrintAllFamilies(GenNode * T)
//postcondition: Prints ALL families, each family on one line
{
}
\end{verbatim}

PROBLEM 5: (Binary Tree Stuff: (23 pts))

Consider the definition for a Node in a binary tree, shown on the Exam 2 Handout.

PART A (7 pts):
Write the function \textit{Count} whose header is shown below. \textit{Count} returns the number of nodes containing the specified value.

For example, consider the binary tree shown below.
If \( T \) is pointing to the root of the tree above, the call \( \text{Count}(T, \text{"sue"}) \) returns 7, \( \text{Count}(T, \text{"jo"}) \) returns 3, and \( \text{Count}(T, \text{"josefina"}) \) returns 0.

Complete function \( \text{Count} \) below the following header.

```c
int \text{Count}(\text{Node} * T, \text{string} value)
// postcondition: returns number of nodes containing "value"
{
}
```

**PART B (2 pts):** Assume \( N \) is the number of nodes in Tree \( T \), and Tree is a completely balanced tree. Write a recurrence relation for the code you wrote in Part A. (you do not have to solve the recurrence relation)

**PART C (7 pts):**
Write the function \( \text{NamePathLength} \) whose header is shown below. \( \text{NamePathLength} \) returns the length of the longest path of nodes containing the given name starting at the given tree node. If the given tree node does not match the given name, return 0. The path length is defined by the number of nodes on the path.

Consider the binary tree \( T \) shown in part A. The call \( \text{NamePathLength}(T, \text{"jo"}) \) returns 1, as there is a path with just one occurrence of "jo" starting at \( T \). The call \( \text{NamePathLength}(T->\text{left}, \text{"sue"}) \) returns 4, as the longest path of consecutive sue’s starting at \( T->\text{left} \) is 4. The call \( \text{NamePathLength}(T->\text{left}, \text{"bill"}) \) returns 0, as the longest path of consecutive bill’s starting at \( T->\text{left} \) is 0 (that is, \( T->\text{left} \) is sue so it cannot have a path of consecutive bill’s).

Complete function \( \text{NamePathLength} \) below the following header.

```c
int \text{NamePathLength}(\text{Node} * T, \text{const string} & name)
{
}
```

**PART D (7 pts):**
Write the function \( \text{GrowLeaves} \) whose header is shown below. \( \text{GrowLeaves} \) adds a leaf (with the same value as its parent) everywhere it can in the original tree. Assume \( \text{GrowLeaves} \) is never called with an empty tree.
For example, the tree on the right below is the result of the call \textit{GrowLeaves}(T) applied to the tree T on the left below.

Complete function \textit{GrowLeaves} below the following header.

\begin{verbatim}
void GrowLeaves(Node * T)
  // precondition: T != NULL
  // postcondition: New leaf nodes with the same value as their parent
  // are added everywhere a leaf can be added.
{

}
\end{verbatim}

\begin{verbatim}
struct Person
{
  string name; // name of person
  int ID;      // identification number

  Person();       // constructor
  Person(string newName, int newID); // constructor
};

class Queue
{
  public:
    Queue( );     // construct empty queue
    Queue( const Queue & q ); // copy constructor
  ~Queue( );     // destructor
    const Queue & operator = ( const Queue & rhs );
    const Person & Front( ) const; // return front (no dequeue)
    bool IsEmpty( ) const;        // return true if empty else false
    int Length( ) const;          // return number of elements in queue
    void Enqueue( const Person & item ); // insert item (at rear)
    void Dequeue( );              // remove first element
    void Dequeue(Person & item );  // combine front and dequeue
    void MakeEmpty( );            // make queue empty
\end{verbatim}
private:
  struct PersonNode
  {
    Person * per;
    PersonNode * next;

    PersonNode(Person * newPer, PersonNode * newNext)
      : per(newPer), next(newNext)
    {
    }
  };

  int mySize;           // # of elts currently in queue
  PersonNode * myFront; // first element
  PersonNode * myBack;  // last element
};

Queue::Queue()
  : mySize(0), myFront(NULL), myBack(NULL)
// postcondition: the queue is empty
{ }

Person::Person()
  : name(""), ID(0)
{ }

Person::Person(string newName, int newID)
  : name(newName), ID(newID)
{ }

Definition for Problem 4

struct GenNode
{
  string name;
  GenNode * child;
  GenNode * sibling;
};

Definition for Problem 5

struct Node
{
  string name;
  Node * left;
  Node * right;
  int size;
};
Node * right;

Node (string newName, Node * lf, Node * rt);
};

Node::Node (string newName, Node * lf, Node * rt)
    : name(newName), left(lf), right(rt)
{ }