Friday, March 11, 2022.

Data Structures and Algorithms LAB – Spring 2022

(BS-IT-F20 Morning & Afternoon)

Lab # 2

Instructions:

- Attempt the following tasks exactly **in the given order**.
- You must complete all tasks individually. Absolutely NO collaboration is allowed. Any traces of plagiarism/cheating would result in an **"F"** grade in this course and lab.
- **Indent** your code properly.
- Use meaningful variable and function names. Use the **camelCase** notation.
- Use meaningful prompt lines/labels for all input/output.
- Do NOT use any **global** or **static** variable(s). However, global named constants may be used.
- Make sure that there are **NO** ***dangling pointers*** or ***memory leaks*** in your programs.

Task # 1

Given the following declaration of the **SortedList** class which we have discussed in lecture as well:

```
class SortedList {
  private:
      int * arr;      // Array which contains the elements of the list Sorted in increasing order
      int maxSize;        // Max size (capacity) of the list
      int currSize;       // Current size of the list
  public:
      SortedList (int size);  // Constructor
      ~SortedList ();         // Destructor
      bool isEmpty ();        // Check if the list is empty
      bool isFull ();         // Check if the list is full
      bool insert (int val);  // Insert a new value in the list
      bool remove (int index, int& val);
                              // Remove the value stored at a particular index in the list
      void display ();        // Display the contents of list on screen
      ...
};
```

Implement the following public member functions of the **SortedList** class:

- **bool SortedList::replace (int index, int newVal);**

If the value of **index** is invalid, this function should not modify the list and should return **false**. If the value of **index** is valid, this function should replace the value stored at **index** with the value **newVal**. Then, it should *readjust* the order of values in the list, so that the resulting list is still **sorted** in increasing order. After that, this function should return **true**.

- `bool SortedList::binarySearch (int val);`

  This function should use the **Binary Search** algorithm to determine whether **`val`** is present in the list or not. If **`val`** is found in the list, then this function should return **true**. Otherwise, it should return **false**.

Task # 2

Given the following declaration of the **`UnsortedList`** class that we have discussed in lecture as well:

```
class UnsortedList {
  private:
      int * arr;          // Array which contains the elements of the list in Unsorted order
      int maxSize;        // Max size (capacity) of the list
      int currSize;       // Current size of the list
  public:
      UnsortedList (int size);        // Constructor
      ~UnsortedList ();               // Destructor
      bool isEmpty ();                // Check if the list is empty
      bool isFull ();                 // Check if the list is full
      bool insert (int val);          // Insert a new value in the list
      bool remove (int index, int& val);
                                // Remove the value stored at a particular index in the list
      void display ();                // Display the contents of list on screen
      ...
};
```

Implement the following public member functions of the **`UnsortedList`** class:

- `bool UnsortedList::removeMax (int& maxVal);`

  This function should **remove** the **first occurrence** of the **maximum**/largest value present in the list and store it in the reference parameter. After that this function should return **`true`**. If the list is empty, this function should return **`false`**.

- `void UnsortedList::reverse ();`

  This function should **reverse** the contents of the list object on which it has been called. You are NOT allowed to declare/use any temporary array or **`UnsortedList`** object in this function.

- **`void UnsortedList::combineList (const UnsortedList& list2);`**

  This function should combine the elements of current list object (on which this function has been called) and the elements of the list object **`list2`** that has been passed as a parameter into this function. For example, assuming that **`list1`** and **`list2`** are objects of **`UnsortedList`** class, this function will be used like this:

  ```
  list1.combineList (list2);
  ```

  For example, if **`list1`** contains **`{ 8 3 4 5 }`** and **`list2`** contains **`{ 2 9 }`**, then after the above statement, **`list1`** should contain **`{ 8 3 4 5 2 9 }`** and **`list2`** should remain *unchanged* i.e., it should still contain **`{ 2 9 }`**.

  In the above function, you may need to deallocate and reallocate the array inside the current **`UnsortedList`** object.

Task # 3

Implement the following public member functions of the **`SortedList`** class:

- **`int SortedList::removeAll (int val);`**

  This function should remove all occurrences of the value **`val`** from the list. This function should also return the count of the occurrences of **`val`** that were removed from the list. In this function, you are NOT allowed to traverse the list more than ONCE. You are also NOT allowed to declare/use any temporary array/object in this function.

- **`SortedList::SortedList (const SortedList& orig);`**

  Copy constructor

- **`SortedList& SortedList::operator = (const SortedList& rhs);`**

  Overloaded assignment operator

Task # 4

Implement the following public member functions of the **`UnsortedList`** class:

- **`int UnsortedList::replaceVal (int oldVal, int newVal);`**

  This function should replace all occurrences of **`oldVal`** in the list with the value **`newVal`**. This function should also return the **count** of the replacements that were performed by this function.

- **`bool UnsortedList::removeMin (int& minVal);`**

  This function should remove the **first occurrence** of the **minimum**/smallest value present in the list and store it in the reference parameter. After that this function should return **`true`**. If the list is empty, this function should return **`false`**.

- **`bool UnsortedList::removeLastOccurrence (int val);`**

  This function should remove the **last occurrence** of the value **`val`** from the list and return **`true`**. If the list is empty or **`val`** is not present in the list, this function should return **`false`**.

- `int UnsortedList::removeAll (int val);`

  This function should remove all occurrences of the value **`val`** from the list. This function should also return the count of the occurrences of **`val`** that were removed from the list. In this function, you are NOT allowed to traverse the list more than ONCE. You are also NOT allowed to declare/use any temporary array/object in this function. Moreover, your function should preserve the order of remaining elements of the list.

- `UnsortedList::UnsortedList (const UnsortedList& orig);`

  Copy constructor

- `UnsortedList& UnsortedList::operator = (const UnsortedList& rhs);`

  Overloaded assignment operator

VERY IMPORTANT

In the next Lab, you will need some or all of the functions from Today's Lab. So, make sure that you have the working implementation of **ALL** the functions of Today's Lab, when you come to the next Lab.