Data Structures & Algorithms LAB
(BSCS-F18 Morning & Afternoon)
Lab # 3

Solve all tasks in order.

Task # 1

Write a function to determine and return the *k*th largest element of an array containing **n** elements. The prototype of your function should be:

int findKthLargest (int* A, int n, int k)

In the above prototype, **A** is the array containing **n** integers from which we want to find the **k**th largest element. You can assume that all elements of the array **A** are **unique** (i.e. there are no duplicates). In your function, you are NOT allowed to modify the contents of the array **A**.
Determine the **exact step count** of your implemented function and also determine its **time complexity** (in Big Oh notation).

Task # 2

Write a function to **reverse** the contents of a 1-D array of integers. You are NOT allowed to
allocate any new array within this function. Also determine the **exact step count** of your function as well as its **time complexity** (in Big Oh notation).

Task # 3

Given the following declaration of the **SortedList** class:

```
class SortedList {
private:
int * arr; // Array which contains the elements of the list in Sorted (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 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
...
};
```

You are required to 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.

```
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 **newVal**. Then, it should readjust the order of values so that the resulting list is still sorted in increasing order. After that, this function should return **true**.

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

Copy constructor

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

Overloaded assignment operator

Task # 4

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 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
...
};
```

You are required to 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 in the list.

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

Copy constructor

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

Overloaded assignment operator

```
int UnsortedList::removeMax ();
```

This function should remove and return the maximum/largest value present in the list.

```
int UnsortedList::removeMin ();
```

This function should remove and return the minimum/smallest value present in the list.

```
UnsortedList UnsortedList::combine (const UnsortedList& list2) const;
```

This function should combine the elements of current list object (on which this function has been called) with the list object (**`list2`**) that has been passed as a parameter into it. This function