Data Structures & Algorithms LAB – Fall 2015

(BS-SE-F14 Morning & Afternoon)

Lab # 2

Instructions:

- Make sure that there are no ***dangling pointers*** or ***memory leaks*** in your programs.
- Indent your code properly.
- Use meaningful variable and function names. Follow the naming conventions.
- Use meaningful prompt lines/labels for all input/output that is done by your programs.
- Implement all the functions **in the given order**.

Task # 1

You are given two arrays of integers, both containing exactly **n** integers. Write 3 different functions to determine the **intersection** (i.e. common elements) of these two arrays, as described below. Assume that there are **no duplicates** in either of the two arrays.

1.1 Implement the function:

```
int intersection1 (int* A, int* B, int* C, int n)
```

which takes three integer arrays of size **n** as parameters. The intersection of arrays **A** and **B** will be stored in the array **C**. This function will return the number of elements that were stored in array **C**.

The worst case time complexity of this function should be $O(n^2)$. You are NOT allowed to allocate any new array within this function.

1.2 Now, you can assume that both the input arrays (**A** and **B**) are **sorted** in **increasing order**. Implement the following function to determine the intersection of the two arrays:

```
int intersection2 (int* A, int* B, int* C, int n)
```

The worst case time complexity of this function should be $O(n \lg n)$. You are NOT allowed to allocate any new array within this function. (Hint: Binary search)

1.3 Still assuming that both the input arrays (**A** and **B**) are **sorted** in **increasing order**, implement the following function to determine the intersection of the two arrays:

```
int intersection3 (int* A, int* B, int* C, int n)
```

The worst case time complexity of this function should be $O(n)$. You are NOT allowed to allocate any new array within this function.

Write a main function which illustrates the working of the above 3 functions.

Task # 2

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 out of 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 # 3

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).