Data Structures & Applications
Fall 2020
Lab 11 – Sorting

Instructor: Saif Hassan **Date: 21st December 2020**

Instructions:

- At the end of this Lab, you will have to submit all files on LMS.
- Attempt all the tasks in any environment and submit code on LMS as well.
- File format should be **.zip/.rar** file containing required **.java** files and additional if required.
- File Name should be your **CMSID_Name_Lab11_Section.zip**.
- Create a project named **lab11_dsa** and perform following tasks.
- **.java** files should be as following:
    - SortingData**.java** → contains complete code.
    - IterativeMergeSort**.java** → contains complete code.
    - TwoSum.java → contains complete code.

Note: Labs submission without following above instructions will not be checked. (No excuse will be entertained.)

Task01: (Determining the order of data)

Click here to download the required files from.
Extract the zip file and create a Java project in Eclipse.

Look at the class SortingData in the sorting folder. The class contains the declaration of four arrays, whose elements are not given.

Consider the following two sorting algorithms:

- Mergesort. Note that in the best case, fewer comparisons are done during the execution of the function merge than in the worst case, because many elements can be copied to tmpArray using the second and third while loops.
- Quicksort using the following pivot selection:

```
int half = (left + right) / 2;
```

  Use the quicksort implementation given in sorting.QuickSort to make sure that your partition algorithm is exactly the same as the one used by the other students in the class.

Assign the array variables using literal arrays containing all integers from 0 to 31, following the example, such that

- the array SortingData.mergeSortBest uses the smallest possible number of comparisons using mergesort,
- the array SortingData.mergeSortWorst uses the largest possible number of comparisons using mergesort,
- the array SortingData.quickSortBest uses the smallest possible number of comparisons using quicksort with the pivot selection given above,
- the array SortingData.quickSortWorst uses the largest possible number of comparisons using quicksort with the pivot selection given above.

Task02: (Implement MergeSort using Iterative Approach)

Implement in the class sorting.IterativeMergeSort a version of mergesort that does not use recursion. Your class should not contain any (static or non-static) functions except for the function mergeSort itself. The program should not create any objects (using new) while the sorting is done; you may create objects before the first comparison.
Hint: Use arrays to simulate.

Task03: (No Submission)

Prove that any comparison-based algorithm to sort four elements requires at least five comparisons

Implement in the class sorting.SortingFour an algorithm for sorting four elements, which requires at most five comparisons.

Task04: (Solve in NlogN)

We are given an array that contains N numbers. We want to determine if there are two numbers whose sum equals a given number K. For instance, if the input is 8, 4, 1, and 6, and K is 10, then the answer is yes (4 plus 6 is 10). A number n may appear more than once in the input array; in that case and only in that case the sum may have the form n + n.

Give in the class sorting.TwoSum an algorithm to solve this problem in **O(N log N)** time.

Hint: Sort the items first!