Birla Institute of Technology & Science, Pilani
2nd Semester 2016-17 - CS F211 – Data Structures and Algorithms

---

Lab 7 (Evaluation 2): 2nd March 2017

Time: 170 minutes **Marks: 15+ 15 = 30**

Instructions:

- *This test consists of two problems (Problem 1 and Problem 2) specified in two different files.*
- All input expressions should be read from stdin (scanf) and output should be printed on stdout (printf).
- For first 150 minutes, only a subset of test cases will be visible to students after submitting the code on the portal. Only in last 20 minutes, all test cases will be made visible.
- At the end of 170 minute period, the online system will stop evaluating the submissions but it will accept it for additional 10 minutes. At the end of 180 minute period, it will stop accepting the submissions.
- Only the last submission by the student for each problem will be considered for evaluation, irrespective of earlier correct submission.
- Assuming that a problem contains M marks, in case of (Run-error/Compiler-error/Timelimit-error), evaluation will be done for M/2 marks only.
- Total marks of each problem contains some marks for modularity and proper structuring of code.
- All submitted source code will be later checked manually by the instructor and final marks will be awarded. Any case of plagiarism and/or hard coding of test cases will fetch 0 marks for the problem/evaluation component.
- Make sure to return 0 from the main() function in case of normal termination.

Problem 2 of 2

Expected Time: 85 minutes **Marks: 15**

Problem Statement

Implement another variation of quicksort algorithm, based on a variation of partitioning referred to as **"3-way partitioning".** Note that in this case also, we use the technique of calling InsertionSort when the size of a sublist (as a result of partitioning) during the execution becomes less than a threshold t**.**

3-way Partitioning and 3-way QuickSort

Implement a another version of quick sort algorithm (call it Quick3waySort()) on the same array of student records, where partitioning procedure divides the input array

into 3 sublists: one sublist, A[left..L], of values less than p, a second sublist A[L+1..E] of values equal to p, and a third sublist A[E+1..right] of values greater than p, where p is the pivot value. Now only two of the three sublists, i.e. A[left..L] and A[E+1..right] are to be sorted recursively. Note the values A[L+1..E] are already in place.

Note: Use last element as the pivot element.

Compare the running time of:

1. $QuickInSort()$,
2. $QuickDualPivotSort()$
3. $Quick3waySort()$

We will use "total number of comparison operations" and "total number of swaps" performed during the execution of these algorithms. For each algorithm, separately keep track of comparisons and swaps taken in insertion sort and quick sort. Only count the comparisons that are done between two elements of the array.
Note: Refer to given test case to follow the order of printing of these measures.

Input format

Each line will start with one of the following key values (1, 2, 3). Each key corresponds to a function and has a pattern. Implement the following functions according to given pattern and write a driver function which can call the functions according to given key value.

Key	Function to call	Format	Description
0	$readData()$	0 M	Same as problem 1
1	$QuickInSort()$	1 t	Same as problem 1
2	$QuickDualPivotSort()$	2 t	Same as problem 1
3	$Quick3waySort()$	3 t	First, make a temporary array and copy contents into it from the read-array. Call QuickDualPivotSort over this temporary array, in which the partitioning will stop when size of list falls below t, and insertionSort() is called over that sublist. Print the whole sorted array at the end. Print each name and marks (space separated) in a new line.
4	$Compare()$	4	Print all performance measures of the three algorithms as following, where X1, Y1, X2, Y2, X3, Y3, X4, Y4, X5, and Y5 will replaced with actual values (be careful with spelling and case of each character): QuickInSort swaps: QuickSort: X1 InsertionSort: Y1 QuickInSort comparisons: QuickSort: X2 InsertionSort: Y2 QuickDualPivotSort swaps: QuickSort: X3 InsertionSort: Y3 QuickDualPivotSort comparisons: QuickSort: X4 InsertionSort: Y4 Quick3waySort swaps: QuickSort: X5 InsertionSort: Y5 Quick3waySort comparisons: QuickSort: X6 InsertionSort: Y6
-1			Stop the program

Test case

Sample Input	Sample Output
0 10 n1 62 n2 80 n3 52 n4 91 n5 6 n6 3 n7 88 n8 92 n9 43 n10 90 1 3 2 3 3 3 4 -1	n6 3 n5 6 n9 43 n3 52 n1 62 n2 80 n7 88 n10 90 n4 91 n8 92 n6 3 n5 6 n9 43 n3 52 n1 62 n2 80 n7 88 n10 90 n4 91 n8 92 n6 3 n5 6 n9 43 n3 52 n1 62 n2 80 n7 88 n10 90 n4 91 n8 92 QuickInSort swaps: QuickSort: 13 InsertionSort: 8 QuickInSort comparisons: QuickSort: 20 InsertionSort: 5 QuickDualPivotSort swaps: QuickSort: 12 InsertionSort: 7 QuickDualPivotSort comparisons: QuickSort: 20 InsertionSort: 4 Quick3WaySort swaps: QuickSort: 29 InsertionSort: 11 Quick3WaySort comparisons: QuickSort: 49 InsertionSort: 7