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 1 of 2

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

Problem Statement

In this problem, you will implement another variation of quicksort algorithm, known as **"dual-pivot quicksort".** As the name suggests, in dual-pivot quicksort algorithm, we select two pivots (and partition the list into three sub-lists) instead of a single pivot (the details of the algorithm are given below).

Implement the **"dual-pivot quicksort"** algorithm and compare its performance with the traditional quicksort algorithm on multiple input data given as test cases.

Implementation:

`Student` `Record:`

- Name: single word (at most 20 characters)
- Marks: unsigned integer

Sorting would be performed based on marks of the students.

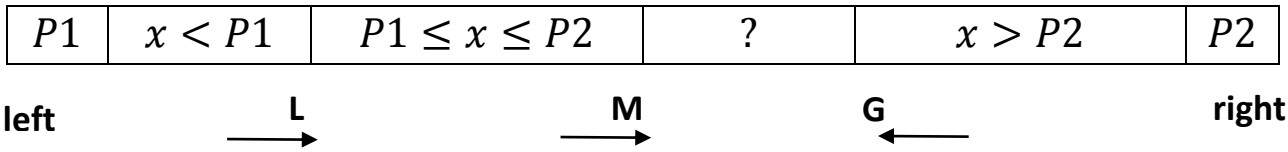
1. QuickInSort (with last element as pivot)

Implement a modified version of quick sort algorithm (call it $QuickInSort()$) on an array of student records (described later), where we call InsertionSort when the sizeof a sublist (as a result of partitioning) during the execution of simple quicksort becomes less than a threshold t. [Note that the threshold t should be parameterized (taken as input from $stdin$).]

2. QuickDualPivotSort (dual-pivot quicksort)

Implement another version of quick sort algorithm (call it $QuickDualPivotSort()$) on the same array of student records, where we call InsertionSort when the sizeof a sublist (as a result of partitioning) during the execution becomes less than a threshold t.

The dual-pivot quicksort algorithm uses partitioning a source array (say A) into three parts defined by two pivot elements P1 and P2 (see Figure).



The algorithm provides the following steps:

1. For small arrays (length <t), use the Insertion sort algorithm.
2. Choose A[left] as P1 and A[right] as P2. P1 must be less than P2, otherwise they are swapped.
3. Partition the list using P1 and P2 maintaining the invariant depicted in the figure above, maintaining sub-lists: A[left+1..L] of values less than P1, A[L+1..M] of values greater than or equal to P1 but less than or equal to P2, and A[G..right-1] of values greater than P2.
    a. In each step, the next element A[M+1] for M+1<G is compared with P1 and P2 (as required) and placed in the appropriate sublist and the boundaries are adjusted.
    b. Step a. is repeated until M+1 becomes G.
4. Swap A[left] with A[L]; swap A[right] with A[G] and update boundaries accordingly.

5. Steps 1 - 4 are repeated (i.e. recursively) for each sub-list: A[left..L-1], A[L+1..G-1], A[G+1..right]

Compare running time of $QuickInSort()$ and $QuickDualPivotSort()$

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 (0, 1, 2, 3, and -1). 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	Indicates that next M lines will contain data to be read. Each line will contain student's name followed by marks, separated by space. You must read the input into an array of student records (**read-array**).
1	$QuickInSort()$	1 t	First, make a temporary array and copy contents into it from the read-array. Call QuickInSort() over this temporary array, in which the partitioning will stop when size of list falls below *t*, and insertionSort() is called over that sublist. Also, print the whole sorted array at the end. Print each name and marks (tab separated) in a new line. Save the value of
2	$QuickDualPivotSort()$	2 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. Also print the whole sorted array at the end. Print each name and marks (tab separated) in a new line.
3	$Compare()$	3	Print all performance measures of QuickInSort() and QuickDualPivotSort() as following, where X1, Y1, X2, Y2, X3, Y3, X4, and Y4 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
-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 -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 QuickInSort swaps: QuickSort: 13 InsertionSort: 8 QuickInSort comparisons: QuickSort: 20 InsertionSort: 5 QuickDualPivotSort swaps: QuickSort: 12 InsertionSort: 7 QuickDualPivotSort comparisons: QuickSort: 20 InsertionSort: 4