Indian Institute of Technology Kharagpur

CS29003: Algorithms Laboratory, Spring 2023

Test 1

2PM – 5PM 14th February, 2023

General Instructions (to be followed strictly)

Submit a single C/C++ source file.
Do not use global variables unless you are explicitly instructed so.
Do not use Standard Template Library (STL) of C++.
Use proper indentation in your code and include comments.
Name your file as `<roll_no>_t1.<extn>`
Write your name, roll number, and assignment number at the beginning of your program.
Submit on Moodle before the deadline. Submissions by any other means will not be considered for evaluation.

Let $\{P_1, P_2, \ldots, P_n\}$ be n points in d-dimensional space. Each P_i is given by d many coordinates. Let $P = (x_1, x_2, \ldots, x_d)$ and $Q = (y_1, y_2, \ldots, y_d)$ be two points and let $m_{P,Q}$ be the smallest index such that $x_{m(P,Q)} \neq y_{m(P,Q)}$. We say that P precedes Q in a lexicographic order, denoted $P \prec Q$, if $x_{m(P,Q)} < y_{m(P,Q)}$ and $m(P,Q) \leq n$; otherwise, if $x_{m(P,Q)} > y_{m(P,Q)}$, then $Q \prec P$. Your task is to arrange a given set $\{P_1, P_2, \ldots, P_n\}$ of points in the lexicographic order i.e., to find a permutation $\pi : [1, n] \to [1, n]$ such that $P_{\pi(1)} \preceq P_{\pi(2)} \preceq \cdots \preceq P_{\pi(n)}$.

(a) Write a function *compare()* that takes as input 2 points P_1, P_2 and outputs -1, 0 or 1 accordinig to whether $P_1 \preceq P_2$, $P_1 = P_2$ or $P_2 \preceq P_1$ respectively.

(b) Write a function *sort()* that given n points implements the Quicksort algorithm to sort the points in lexicographic order. Use the *compare* function for comparisons among points.

(c) Write a function *cleversort()* that modifies the Quicksort algorithm as follows. First choose a pivot point P. Partition the array $\mathbf{A}$ of points into 3 parts – $\mathbf{A}_<, \mathbf{A}_=, \mathbf{A}_>$ where

- $\mathbf{A}_<$ consists of all points Q such that $Q \prec P$ and $m(P,Q) = 1$,
- $\mathbf{A}_=$ consists of points Q such that $m(P,Q) > 1$, and
- $\mathbf{A}_>$ consists of all points Q such that $P \prec Q$ and $m(P,Q) = 1$.

Note that $\mathbf{A}_=$ consists of all points with at least the first coordinate matching with that of P.

Once the array is partitioned as above, call *cleversort()* recursively on the 3 partitions with the first coordinate being ignored for the middle partition.

(d) For both algorithms, you may choose the pivot at random. Why is *cleversort()* cleverer? Write down the expected complexity of both *sort()* and *cleversort()* in terms of n, d as comments preceding the respective function definitions.

In the *main()* function,

- Read d, the dimension and n, the number of points.
- Read the coordinates of n points. Assume that all points have integer coordinates.
- Call *sort()* on the n points read and then print the sorted array. Then print the total number of single integer comparisons made during the sort.
- Call *cleversort()* and print the sorted array. (You may store a copy of the original list of points for applying *cleversort()*.) Then print the total number of single integer comparisons made during the sort.

Do not use any built-in library functions. You may use a global variable only for counting the number of single integer comparisons made by the sorting algorithms.

Sample Output

```
$ ./a.out
3 10
Enter 10 points, one in each line
3 6 0
7 1 9
3 7 1
7 1 5
1 -1 4
3 6 -2
7 -9 2
6 2 1
7 4 6
0 7 -1

--- QUICKSORT ---
   Sorted Array:
      0 7 -1
      1 -1 4
      3 6 -2
      3 6 0
      3 7 1
      6 2 1
      7 -9 2
      7 1 5
      7 1 9
      7 4 6

   Number of single integer comparisons: 44


--- CLEVER QUICKSORT ---
   Sorted Array
      0 7 -1
      1 -1 4
      3 6 -2
      3 6 0
      3 7 1
      6 2 1
      7 -9 2
      7 1 5
      7 1 9
      7 4 6

   Number of single integer comparisons: 28
```

Note that the number of comparisons may not be the same as shown above since the pivot is chosen at random. In fact, they would be different each time you run the program on the same input.

Policy on Plagiarism

Academic integrity is expected from all the students. Ideally, you should work on the assignment/exam consulting only the material we share with you. You are required to properly mention/cite anything else you look at. Any student submitting plagiarised code will be penalised heavily. Repeated violators of our policy will be deregistered from the course. Read this to know what is plagiarism.