MA-INF 1203 Discrete and Computational Geometry

Wintersemester 2019/20
Assignment 5

Deadline: **12** November before noon (To be discussed: **12/13**. November 2019)

1 Backwards analysis

Consider the algorithm `flatQuicksort` given below. The algorithm sorts the numbers stored in an array by repeatedly partitioning a range of the array with respect to a randomly chosen pivot element. Assume that the input numbers are pairwise distinct. In each iteration of the outer for-loop, the range between two previously chosen pivots is partitioned into three sets, namely, the elements smaller than the pivot, the pivot itself, and the elements larger than the pivot. This is done by swapping elements of the array. Give an upper bound on the total expected number of swaps done by the algorithm when called on an array of n distinct numbers. Use an indicator random variable $M(x, i) \in \{0, 1\}$ that evaluates to 1 if and only if the input number x is touched in the inner for-loop of the ith iteration of the outer for-loop.

2 Deterministic convex hull

Modify the two-dimensional convex hull algorithm from the lecture so that it runs in deterministic $O(n \log n)$ time in the worst case. Instead of maintaining the lists of points for each edge in lines 12 and 13, your algorithm should find the chain of edges to be removed in a different way. Your algorithm may maintain additional data structures or preprocess the data as you see fit. Analyse the running time and correctness of your modified algorithm.

3 Lower bound

Consider the following variant of the convex hull problem in $\mathbb{R}^3$. The input is a sequence of points $x_1, \ldots, x_n$ in $\mathbb{R}^3$ in general position and given in sorted order with respect to their third coordinate. The output is the graph of vertices and edges of $\text{conv}(\{x_1, \ldots, x_n\})$ given by a doubly connected edge list (DCEL). Assume that the DCEL also stores the coordinates of the vertices. Can you still show a lower bound of $\Omega(n \log n)$ by reduction from sorting? Explain your answer.

```
1  algorithm flatQuicksort(A,n)
2      // A is indexed from 1 to n
3
4      // fixed[i] maintains whether A[i] has been used as a pivot;
5      // includes sentinels at 0 and n+1, initialized to true,
6      // so that we do not go out of bounds on lines 22 and 24
7
8      fixed[0..n+1] := {true,false,false,...,false,true}
9
10     for i := 1 to n-1 do
11
12         // select random pivot
13         repeat
14              p := random number from {1,...,n}
15         until fixed[p] = false
16
17         pivot := A[p]
18
19         // scan left and right to find range
20         lo := p
21         hi := p
22         while not fixed[lo-1] do
23             lo := lo-1
24         while not fixed[hi+1] do
25             hi := hi+1
26
27         //move pivot to right end of range
28         swap A[p] with A[hi]
29
30         //partition the range
31         split := lo
32         for j := lo to hi do
33             if A[j] < pivot then
34                 swap A[split] with A[j]
35                 split := split + 1
36
37         //move pivot to splitting point
38         swap A[split] with A[hi]
39
40         fixed[split] := true
41
42      return
```