Exercise sheet 3.

Data structures — SoSe 2020

László Kozma, Katharina Klost

Due 12:00, May 15th, 2020

Exercise 1 Sorted matrices — $3 + 3 + 4$ *Points*

Consider an $m \times n$ matrix M with distinct integer elements with the m rows in increasing order left-to-right and the n columns in increasing order top-to-bottom.

(a) Given a value K, describe an efficient algorithm that finds out whether K appears in M. Ideally, the running time should be $O(m + n)$.

(b) Given a value K, describe an efficient algorithm to find *rank*(K), i.e. the number of entries $M_{i,j}$ of the matrix M such that $M_{i,j} \leq K$. Ideally, the running time should be $O(m + n)$.

(c) Suppose now that the rows of M are sorted, but its columns are not. Show that in $O(m + k)$ time we can select the k-th smallest element in m.

Hint: This is almost the same as exercise 3 in the previous exercise sheet, but now we have *selection from heaps* as a tool.

Bonus question: +5p: In question (c) if k is much larger than m, then $O(m+k)$ may be too wasteful. Try to find an alternative method that achieves a better running time in this case, e.g. $O(m \log k)$ or even $O(m \log \frac{k}{m})$. You can use any of the existing methods as subroutines.

Hint: Can you identify at least a constant fraction of the top-k elements? How would that help?

Exercise 2 Selection from $X + Y$ — 4 *Points*

Given two (unsorted) sets $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$, we want to find the k-th smallest of all possible pairwise sums $x_i + y_j$. In the lecture we argued that if we sort X and Y, we can reduce the problem to *selection from sorted matrices*, and the overall running time is $O(n \log n + k)$.

We would like to improve the running time to $O(n+k)$, so we must avoid the sorting step. Find a way to reduce the problem directly to *selection from heaps* that yields the given bound.

Hint: recall that building a binary heap from a list takes only linear time!

Another hint: it may be easier to build a heap of constant degree greater than 2.

Exercise 3 Subarray-selection $2 + 4$ *Points*

We are given an array with nonnegative entries $A = (a_1, \ldots, a_n)$. For two arbitrary indices $1 \leq \ell \leq j \leq n$, the *subarray-sum* between ℓ and j is defined as $\sum_{i=\ell}^{j} a_i$.

(a) Describe an $O(n)$-time preprocessing step, after which $a_{\ell,j}$ can be computed in constant time for arbitrary ℓ, j.

(b) Assuming that $a_{\ell,j}$ is available in constant time, give an efficient method to compute the k-th smallest subarray-sum. What is the running time in terms of n and k? Can you find the k-th *largest* subarray-sum more efficiently? You can use any of the existing methods as subroutines.

Total: 20 points. Have fun with the solutions!