

---

## Advanced Methods for Sequence Analysis

---

*due: February 9th, before the lecture*

### Exercise 1

Show step by step how the linear-time algorithm constructs the Cartesian Tree for

$$B_j = [2, 4, 1, 5, 3, 1, 5, 7, 4, 2] .$$

### Exercise 2

Let  $T$  be a static tree with  $n$  nodes, not necessarily rooted. Show how to preprocess  $T$  in  $O(n)$  time, such that for given nodes  $v$  and  $w$ , the following type of “distance queries” can be answered on-line in constant time:  $\text{DIST}(v, w) = \text{length of the path from } u \text{ to } v$ .

### Exercise 3

Having finished your studies at Tübingen, you find a highly paid job in a bioinformatics company called XBIOCOMP, where it turns out that Dr. Dumb is your (even higher paid) project director (due to his successful research on suffix trees!). On your first day at XBIOCOMP, he explains to you their current project:

“Because we want to identify frequent motifs near the 3’ or 5’ ends of certain cDNA transcript, we often need to locate all occurrences of a DNA-pattern  $P$  in *prefixes* of some nucleotide sequence  $T_{1\dots n}$ . For this, we use the most fancy things that exist right now: suffix trees and arrays.

“So what we do is this: Somebody gives us a pattern  $P$  and a prefix length  $r$ . The goal is to report all occurrences of  $P$  in  $T_{1\dots r}$ . We first find the interval  $[i, j]$  in  $T$ ’s suffix array  $A$  that contains *all* occurrences of  $P$  in  $T$ . Using a suffix tree, this can be done in  $O(|P|)$  time. Then we step through this interval and return only those positions  $k \in \{i, i + 1, \dots, j\}$  for which  $A[k] \leq r$ , since only these are located in the length- $r$ -prefix  $T_{1\dots r}$ .”

Because you have heard Advanced Methods for Sequence Analysis, and know a lot about suffix arrays and range minimum queries, after some thinking you see that Dr. Dumb’s algorithm is not the best, and that can be improved. Explain how!

*Hint:* Consider the fact that in the worst case, a lot of unnecessary positions in  $[i, j]$  have to be scanned.

## Bonus Exercise I

We've seen in the lecture that a suffix array on a text of length  $n$  can be built in  $O(n)$  time. Now imagine a list  $L$  of  $n$  (unsorted) comparable objects  $l_1, \dots, l_n$ . Let's build a (conceptual) string  $S = l_1 l_2 \dots l_n$  and compute the suffix array  $A$  for  $S$  in linear time. Then  $S[A[1]], S[A[2]], \dots, S[A[n]]$  is the sequence  $L$  in *sorted* order. So we have shown that sorting  $n$  objects can be done in  $O(n)$  time.

Explain what is wrong with this "proof"!

## Bonus Exercise II

The RMQ-algorithm from the lecture assumes that the operation  $\lfloor \log_2(\cdot) \rfloor$  can be answered in constant time. If this is not the case: show how to simulate the operation in  $O(\log w)$  time with standard-instructions such as additions and bit-shifts, using an additional data structure of size  $O(\log w)$ . Here,  $w$  denotes the word-size of the processor (e.g., 32 or 64).

## Bonus Exercise III (8 points)

Implement the  $O(n \log n)$  algorithm for  $O(1)$ -RMQs (Sect. 9.3). Plug it into the algorithm for computing LCPs from Exercise Set no. 8 (Java-code on the website!). Measure the space advantage in comparison to the  $O(n^2)$ -method that we have used so far.

*Bonus exercises are meant to improve your points, if you still need/want them. Their points do not count for the total number of achievable points, which remains 64 after Xmas.*