Marburg University

Department of Computer Science
Prof. Dr. Sebastian Wild

Date: 2026-06-05
Version: 2026-06-10 17:43

Exercise Sheet 8 for Algorithmen und Datenstrukturen (Sommer 2026)

Hand In: *Until 2026-06-13 18:00, on ILIAS.*

Problem 1

20 points

Formally prove the following: Any binary tree with n nodes has $n+1$ null pointers and height at least $\geq \lceil \log_2 n \rceil$.

Problem 2

40 points

In this exercise, you will show that Quicksort can be implemented with at most logarithmic extra space even in the worst case. In passing, we will learn a bit more about Quicksort's space usage in general.

While Quicksort (as we implemented it on the slides) does not require a buffer array, we do have to remember the boundaries of recursive calls on a call stack. Although the runtime / operation system does that for us, it does use some space.

For completeness, here is a pseudocode implementation which uses an explicit stack.

```
QUICKSORT(A[0..n)):
    // Input: n ≥ 1, and an array A
1   Stack = an empty stack of input ranges to sort.
2   Stack.push([0, n))
3   while ¬Stack.empty()
4       [ℓ, r) = Stack.pop()
5       if ℓ < r
6           m = PARTITION(A[ℓ..r), A[ℓ]).
7           Stack.push([ℓ, m))
8           Stack.push([m + 1, r))
9   end while
```

a) Try simulating this pseudocode on an already sorted list; since we always choose the first element as pivot, this leads to a worst-case choice of pivot. How does the stack size behave?

b) The trick to cut down Quicksort's space usage is to wisely choose the order in which the two subproblems after partitioning are dealt with. Any order is fine with respect to correctness since the two subproblems are independent; but the stack height is different!

The only change necessary to obtain a much smaller stack is to push the larger subproblem first:

QuicksortSmart($n, A[0..n)$):

```
    // Input: n ≥ 1, and an array A
1   Stack = an empty stack of input ranges to sort.
2   Stack.push([0, n))
3   while ¬Stack.empty()
4       [ℓ, r) = Stack.pop()
5       if ℓ < r
6           m = Partition(A[ℓ..r), A[ℓ]).
7           if m − ℓ < r − m
8               Stack.push([m + 1, r))
9               Stack.push([ℓ, m))
10          else
11              Stack.push([ℓ, m))
12              Stack.push([m + 1, r))
13  end while
```

Prove that *quicksort_smart* never needs more than $\lfloor \log_2(n) \rfloor + 1$ stack entries when `len(A)` is $n \geq 1$.

Hint: Use mathematical induction.
Call h_n the actual maximal stack height needed to sort a (sub)list of size n; we do not know h_n (and it might be quite complicated), but for sure there is such a number. Then prove by induction that $h_n \leq \lfloor \log_2(n) + 1 \rfloor$ for all $n \geq 1$.

Problem 3 (intervals)

40 points

Solve `marburg.kilonova.ro/problems/10` (intervals).

ILIAS Submission: In your submission to ILIAS, describe your algorithm according to the template, *Idea*, *Pseudocode*, *Correctness*, *Analysis*.
The pseudocode part can here an informal summary of your kilonova code submission.