Assignment Sheet 1

Hand-In: 22nd October 2024, 10:15am
Discussion: 24th October 2024, 10:15am

Task 1: Asymptotic Growth (1+1+1+2 Points) ★★★☆☆

Show the following statements:

a) $f(n) = \Omega(g(n))$ and $g(n) = \Omega(h(n))$ imply $f(n) = \Omega(h(n))$.

b) For arbitrary $b > 2$ it holds that $b^n = \omega(2^n)$.

c) For arbitrary $b > 1$ it holds that $\log_b(n) = O(\log_2 n)$.

d) For arbitrary $c > 0$ it holds that $\ln(n) = o(n^c)$.

For your solution of task d) use L'Hôpital's rule[1].

Task 2: Search Algorithms (1+2+2 Points) ★★☆☆☆

Let A be a sorted array of size n, i.e., an n-tuple. The SEARCH problem asks to determine whether A contains a given value x or not.
The algorithm `BinarySearch` solves the problem by comparing the value x with the median element of the current subarray $A[\ell, \dots, r]$ (initial $\ell = 1, r = n$). Based on the comparison's result, the algorithm then recurses either on the subarray with the smaller values or on the subarray with the larger values.

```
BinarySearch(A,ℓ,r,x)
if ℓ < r then
    m ← ⌊(ℓ+r)/2⌋;
    if A[m] ≤ x then
        return BinarySearch(A,ℓ,m,x);
    else
        return BinarySearch(A,m + 1,r,x);
    end
else
    if A[ℓ] = x then
        return true;
    else
        return false;
    end
end
```

If the current subarray contains only a single element, it is determined whether this element is the searched value x.

a) Provide a recurrence $T(n)$ that describes the run time of `BinarySearch`.

b) Prove that $T(n) = O(\log n)$. Apply the Master Theorem if possible.

c) `BinarySearch` is an example for a *comparison-based* search algorithm, i.e., it exclusively performs comparisons and value assignments. Prove that each comparison-based search algorithm requires $\Omega(\log n)$ comparisons in the worst case.

[1] See e.g. `https://en.wikipedia.org/wiki/L%27H%C3%B4pital%27s_rule`

Task 3: Correctness of `MergeSort` (5 Points)

★ ★ ★ ☆ ☆

Prove the correctness of the `MergeSort` algorithm from the lecture. To this end,

1. define a suitable loop invariant for the merging loop.
2. prove that the loop invariant is maintained.
3. inductively prove that MergeSort correctly sorts a list of n elements (you may assume n to be a power of 2).