Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik, Anja Rey

Exercise Sheet 2

for the lecture on

Advanced Programming and Algorithms – Part II

This exercise sheet contains exercises for self-study and discussion.
If you would like feedback on your solutions, please upload a PDF via ILIAS until Monday, 22nd April, 12:30 pm.

Discussion in the exercise classes from 22nd April until 26th April, 2024.

Problem 1 to hand in: *Analyse a Divide & Conquer Algorithm*

Consider the following algorithm with an input array A consisting of integers, sorted ascendingly, and two indices ℓ and r, $0 \leq \ell \leq r \leq \text{length}(A) - 1$.

`do_something`(A, ℓ, r):

```
1 if ℓ = r then
2 |   return ∞
3 m ← ⌊(r+ℓ)/2⌋
4 a ← do_something(A, ℓ, m)
5 b ← do_something(A, m + 1, r)
6 c ← |A[m + 1] − A[m]|
7 return min{a, b, c}
```

a) Given a sorted array A, and, initially, $\ell = 0$ and $r = \text{length}(A) - 1$, briefly explain what the algorithm `do_something` does and what it returns.

b) Let the input size $n = r - l + 1$ be a power of two ($n = 2^k$ for some $k \in \mathbb{N}$). Analyse the asymptotic worst-case running time $T(n)$ of `do_something`:

- Write down which parts of the algorithm take which running time (up to constants). These can depend on the running times $T(n')$ of further calls of the algorithm with a smaller input n'.
- Provide a recurrence that describes $T(n)$ for each $n \geq 1$.
- Solve the recurrence to obtain a closed formula as an (ideally precise) upper bound for $T(n)$ which is the running time for an array of input length n.
- Provide a function as a representative asymptotic upper bound in $\mathcal{O}$-notation.

c) Provide a sketch of a correctness proof for this algorithm `do_something`.

d) In the algorithm above, we assume that the input array is sorted ascendingly. If this is not the case, how can we sort the array first? Briefly explain how it works and how this effects the overall running time of the algorithm.

Problem 2 for discussion: *Binary Search*

Let A be an array consisting of $n \geq 1$ distinct integers, sorted in descending order.

Design an algorithm that, given A and an integer x, returns index i if $A[i] = x$ and -1 if A does not contain x. The algorithm should work by the divide and conquer principle and run in logarithmic time $\mathcal{O}(\log n)$ in the input size n.

Optionally, you can use additional in- and output parameters to store indices or other temporary values.

For instance, given the following array and $x = 11$, the algorithm should return index 2.

0	1	2	3	4	5	6
17	13	11	9	7	5	3

Proceed as follows:

a) Describe the algorithm intuitively and provide pseudocode.

b) Argue why the running time of $\mathcal{O}(\log n)$ is fulfilled.

c) Argue why it works.

Problem 3 as a programming exercise: *Merge Function*

Implement the `merge` function as required for merge sort: Given an array A and three indices, `left`, `middle`, and `right` such that A[`left`..`middle`] and A[`middle` + 1..`right`] are sorted ascendingly, return A with A[`left`..`right`] sorted ascendingly.

At some point the algorithm should iterate over (a part of) the array. What would a loop invarant look like that can be used to show the correctness of the algorithm?

What would a unit test look like that asserts this property exemplarily?

How efficient is your implementation? What changes if we use different data structures?

Problem 4 for discussion: *Recurrences – General Case*

For the sake of simplicity, we assumed in several instances that an input size n is a power of b if a recursion depends on a subproblem of size n/b.

Why can we make this assumption without loss of generality?