Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement of Computer Science
Johannes Lengler, David Steurer
Kasper Lindberg, Lucas Slot, Hongjie Chen, Manuel Wiedmer

14 October 2024

Algorithms & Data Structures — Exercise sheet 4 — HS 24

The solutions for this sheet are submitted on Moodle until 20 October 2024, 23:59.

Exercises that are marked by * are challenge exercises. They do not count towards bonus points.

You can use results from previous parts without solving those parts.

Master theorem. The following theorem is very useful for running-time analysis of divide-and-conquer algorithms.

Theorem 1 (master theorem). *Let $a, C > 0$ and $b \geq 0$ be constants and $T : \mathbb{N} \to \mathbb{R}^+$ a function such that for all even $n \in \mathbb{N}$,*

$$T(n) \leq aT(n/2) + Cn^b. \tag{1}$$

Then for all $n = 2^k$, $k \in \mathbb{N}$, the following statements hold

(i) If $b > \log_2 a$, $T(n) \leq O(n^b)$.

(ii) If $b = \log_2 a$, $T(n) \leq O(n^{\log_2 a} \cdot \log n)$.[1]

(iii) If $b < \log_2 a$, $T(n) \leq O(n^{\log_2 a})$.

If the function T is increasing, then the condition $n = 2^k$ can be dropped. If we instead have

$$T(n) \geq aT(n/2) + C'n^b, \tag{2}$$

then we can conclude that $T(n) \geq \Omega(n^b), T(n) \geq \Omega(n^{\log_2 a} \cdot \log n)$, and $T(n) \geq \Omega(n^{\log_2 a})$ in cases $(i), (ii)$, and (iii), respectively. Furthermore if (1) and (2) both hold (with possibly different constants $C \neq C'$), then similarly $T(n) = \Theta(n^b), T(n) = \Theta(n^{\log_2 a} \cdot \log n)$, and $T(n) = \Theta(n^{\log_2 a})$ in cases $(i), (ii)$, and (iii), respectively.

This generalizes some results that you have already seen in this course. For example, the (worst-case) running time of Karatsuba's algorithm satisfies $T(n) \leq 3T(n/2) + 100n$, so we have $a = 3$ and $b = 1 < \log_2 3$, hence $T(n) \leq O(n^{\log_2 3})$. Another example is binary search: its running time satisfies $T(n) \leq T(n/2) + 100$, so $a = 1$ and $b = 0 = \log_2 1$, hence $T(n) \leq O(\log n)$.

Exercise 4.1 *Applying the master theorem.*

For this exercise, assume that n is a power of two (that is, $n = 2^k$, where $k \in \mathbb{N}_0$). In the following, you are given a function $T : \mathbb{N} \to \mathbb{R}^+$ defined recursively and you are asked to find its asymptotic behavior by applying the master theorem.

[1] For this asymptotic bound we assume $n \geq 2$ so that $n^{\log_2 a} \cdot \log n > 0$.

(a) Let $T(1) = 1, T(n) = 4T(n/2) + 100n$ for $n > 1$. Using the master theorem, show that

$$T(n) \leq O(n^2).$$

(b) Let $T(1) = 5, T(n) = T(n/2) + \frac{3}{2}n$ for $n > 1$. Using the master theorem, show that

$$T(n) \leq O(n).$$

(c) Let $T(1) = 4, T(n) = 4T(n/2) + \frac{7}{2}n^2$ for $n > 1$. Using the master theorem, show that

$$T(n) \leq O(n^2 \log n).$$

Exercise 4.2 *Asymptotic notations.*

(a) **(This subtask is from January 2019 exam).** For each of the following claims, state whether it is true or false. You don't need to justify your answers.

claim	true	false
$\frac{n}{\log n} \leq O(\sqrt{n})$	☐	☐
$\log(n!) \geq \Omega(n^2)$	☐	☐
$n^k \geq \Omega(k^n)$, if $1 < k \leq O(1)$	☐	☐
$\log_3 n^4 = \Theta(\log_7 n^8)$	☐	☐

(b) **(This subtask is from August 2019 exam).** For each of the following claims, state whether it is true or false. You don't need to justify your answers.

claim	true	false
$\frac{n}{\log n} \geq \Omega(n^{1/2})$	☐	☐
$\log_7(n^8) = \Theta(\log_3(n^{\sqrt{n}}))$	☐	☐
$3n^4 + n^2 + n \geq \Omega(n^2)$	☐	☐
$(*) \quad n! \leq O(n^{n/2})$	☐	☐

Note that the last claim is challenge. It was one of the hardest tasks of the exam. If you want a 6 grade, you should be able to solve such exercises.

Sorting and Searching.

Exercise 4.3 *Formal proof of correctness for Insertion Sort* **(1 point)**.

Algorithm 1 Insertion Sort (input: array $A[1 \dots n]$).

```
for j = 1, ..., n do
    if j > 1 then
        for i = j − 1, ..., 1 do
            if A[i + 1] < A[i] then
                Swap A[i + 1] and A[i]
```

Prove correctness of this algorithm by mathematical induction.

Hint: *Use the invariant $I(j)$: "After j iterations the first j elements are sorted."*

Exercise 4.4 *Searching in a weirdly-sorted array* **(1 point)**.

Let $n \geq 2$ and suppose we are given an array $A[1 \dots n]$ containing n unique integers, that satisfies the following property: there is an integer $1 \leq k \leq n-1$ such that the subarrays $A[1 \dots k]$ and $A[k+1 \dots n]$ are sorted (in ascending order), and $A[n] < A[1]$. We call such an array *weirdly-sorted*, and we call k the *pivot*.

(a) Given a weirdly-sorted array $A[1 \dots n]$ containing n unique integers, provide an algorithm in pseudocode that finds the pivot $1 \leq k \leq n-1$ such that the subarrays $A[1 \dots k]$ and $A[k+1 \dots n]$ are sorted (in ascending order). The runtime of your algorithm should be at most $O(\log n)$.

Hint: *Be careful of edge-cases.*

Hint: *For an index $1 \leq m \leq n$, think of a simple condition involving $A[1], A[n]$ you could check to see if m is to the left or to the right of the pivot.*

(b) Given a weirdly-sorted array $A[1 \dots n]$ containing n unique integers, and an integer $\ell \in \mathbb{N}$, provide an algorithm in pseudocode that determines whether A contains ℓ as an entry. The runtime of your algorithm should be at most $O(\log n)$. You may use the algorithm of part (a) as a subroutine even if you did not solve that part. You may also use algorithms from the lecture as subroutines.

Exercise 4.5 *Counting function calls in loops (cont'd)* **(1 point)**.

For each of the following code snippets, compute the number of calls to f as a function of $n \in \mathbb{N}$. We denote this number by $T(n)$, i.e. $T(n)$ is the number of calls the algorithm makes to f depending on the input n. Then T is a function from $\mathbb{N}$ to $\mathbb{R}^+$. For part (a), provide **both** the exact number of calls and a maximally simplified asymptotic bound in Θ notation. For part (b), it is enough to give a maximally simplified asymptotic bound in Θ notation. For the asymptotic bounds, you may assume that $n \geq 10$.

Algorithm 2

(a)

```
i ← 1
while i ≤ n do
    j ← 1
    while ⁱ√j ≤ n do
        f()
        j ← j + 1
    i ← i + 1
```

Hint: *You may use the formula for a finite geometric series without proof*

$$\sum_{i=0}^{n} ar^i = \frac{a(r^{n+1}-1)}{r-1} \text{ for } r \neq 1.$$

Algorithm 3

(b)

```
function A(n)
    i ← 1
    while i ≤ n do
        j ← i
        while j ≤ n do
            f()
            f()
            j ← j + 1
        i ← i + 1
    k ← ⌊n/2⌋
    for ℓ = 1...3 do
        if k > 0 then
            A(k)
```

You may assume that the function $T : \mathbb{N} \to \mathbb{R}^+$ denoting the number of calls of the algorithm to f is increasing.

Hint: *Recall exercise 0.1. If $T(n) = aT(n/2) + g(n)$ for some function $g(n)$, then find a bound $Cn^b \leq g(n) \leq C'n^b$ for two constants C and C' and then use the Θ version of the master theorem. Equivalently show that $g(n) = \Theta(n^b)$.*

(c)* Prove that the function $T : \mathbb{N} \to \mathbb{R}^+$ from the code snippet in part (b) is indeed increasing.

Hint: *You can show the following statement by mathematical induction: "For all $n' \in \mathbb{N}$ with $n' \leq n$ we have $T(n'+1) \geq T(n')$".*