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, Markus Püschel, David Steurer
Kasper Lindberg, Kostas Lakis, Lucas Pesenti, Manuel Wiedmer

13 October 2025

Algorithms & Data Structures Exercise sheet 4 HS 25

The solutions for this sheet are submitted on Moodle until 19 October 2025, 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 $T(n) \leq O(\log n)$.

[1] For this asymptotic bound we assume $n \geq 2$ so that $n^{\log_2 a} \cdot \log n > 0$. Notice also that $a = 1$ leads to a logarithmic bound, i.e. $O(\log n)$.

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.

(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, slightly altered).** For each of the following claims, state whether it is true or false. You don't need to justify your answers.

claim	true	false
$\frac{n}{\log n} \geq \Omega(n^{0.9999})$	☐	☐
$\log_7(n^8) = \Theta(\log_3(n^{\sqrt{n}}))$	☐	☐
$3n^4 + n^2 + n \geq \Omega(n^2)$	☐	☐
$(*) \quad n! \leq O(n^{n/2})$	☐	☐

Note that the last claim is challenge. It was one of the hardest tasks of the exam. It would most likely be a necessary condition to solve such exercises to get a 6.0.

Sorting and Searching.

Exercise 4.3 *One-Looped Sort* **(1 point)**.

Consider the following pseudocode whose goal is to sort an array A containing n integers.

Algorithm 1 Input: array $A[0 \ldots n-1]$.

```
i ← 0
while i < n do
    if i = 0 or A[i] ≥ A[i − 1] then:
        i ← i + 1
    else
        swap A[i] and A[i − 1]
        i ← i − 1
```

(a) Show the steps of the algorithm on the input $A = [10, 20, 30, 40, 50, 25]$ until termination. Specifically, give the contents of the array A and the value of i after each iteration of the while loop.

(b) Explain why the algorithm correctly sorts any input array. Formulate a reasonable loop invariant, prove it (e.g., using induction), and then conclude using that invariant that the algorithm correctly sorts the array.

Hint: *Use the invariant "at the moment when the variable i gets incremented to a new value $i = k$ for the first time, the first k elements of the array are sorted in increasing order".*

(c) Give a reasonable running-time upper bound, expressed in O-notation.

Exercise 4.4 *Searching for the summit* **(1 point)**.

Suppose we are given an array $A[1 \ldots n]$ with n **unique** integers that satisfies the following property. There exists an integer $k \in [1, n]$, called the *summit index*, such that $A[1 \ldots k]$ is a strictly increasing array and $A[k \ldots n]$ is a strictly decreasing array. We say an array is **valid** is if satisfies the above properties.

(a) Provide an algorithm that find this k with worst-case running time $O(\log n)$. Give the pseudocode and give an argument why its worst-case running time is $O(\log n)$.

Note: Be careful about edge-cases! It could happen that $k = 1$ or $k = n$, and you don't want to peek outside of array bounds without taking due care.

(b) Given an integer x, provide an algorithm with running time $O(\log n)$ that checks if x appears in the (valid) array or not. Describe the algorithm either in words or pseudocode and argue about its worst-case running time.

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 $n \geq 10$.

(a)

Algorithm 2

```
i ← 1
while i ≤ n do
    j ← i
    while 2^j ≤ n do
        f()
        j ← j + 1
    i ← i + 1
```

Hint: To find the asymptotic bound, it might be helpful to consider n of the form $n = 2^k$.

(b)

Algorithm 3

```
function A(n)
    i ← 0
    while i < n^2 do
        j ← n
        while j > 0 do
            f()
            f()
            j ← j − 1
        i ← i + 1
    k ← ⌊n/2⌋
    for l = 0...3 do
        if k > 0 then
            A(k)
            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: *To deal with the recursion in the algorithm, you can use the master theorem.*

(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 $T(n' + 1) \geq T(n')$".*