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
Lucas Slot, Manuel Wiedmer, Hongjie Chen, Ding Jingqiu

16 October 2023

Algorithms & Data Structures — Exercise sheet 4 — HS 23

The solutions for this sheet are submitted at the beginning of the exercise class on 23 October 2023.

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 $n = 2^k$, $k \in \mathbb{N}$,

- *If $b > \log_2 a$, $T(n) \leq O(n^b)$.*
- *If $b = \log_2 a$, $T(n) \leq O(n^{\log_2 a} \cdot \log n)$.*[1]
- *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 (1) holds with "$=$", then we may replace O with Θ in the conclusion.

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 $k \in \mathbb{N}_0 := \mathbb{N} \cup \{0\}$).

(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

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

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

(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 Bubble Sort* **(1 point)**.

Recall the bubble sort algorithm that was introduced in the lecture.

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

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

Prove correctness of this algorithm by mathematical induction.

Hint: *Use the invariant $I(j)$ that was introduced in the lecture: "After j iterations the j largest elements are at the correct place."*

Exercise 4.4 *Exponential search* **(1 point)**.

Suppose we are given a positive integer $N \in \mathbb{N}$, and a *monotonically increasing* function $f : \mathbb{N} \to \mathbb{N}$, meaning that $f(i) \geq f(j)$ for all $i, j \in \mathbb{N}$ with $i \geq j$. Assume that $\lim_{n\to\infty} f(n) = \infty$. We are tasked to determine the *smallest* integer $T \in \mathbb{N}$ such that $f(T) \geq N$.

(a) Describe an algorithm that finds an *upper bound* $T_{\text{ub}} \in \mathbb{N}$ on T, such that $f(T_{\text{ub}}) \geq N$ and $T_{\text{ub}} \leq 2T$, making $O(\log T)$ function calls to f.[2] Prove that your algorithm is correct, and uses at most the desired number of function calls.

(b) Describe an algorithm that determines the *smallest* integer $T \in \mathbb{N}$ such that $f(T) \geq N$, making $O(\log T)$ function calls to f. Prove that your algorithm is correct, and uses at most the desired number of function calls.

Hint: *Consider using a two-step approach. In the first step, apply the algorithm of part (a). For the second step, modify the binary search algorithm and apply it to the array $\{1, 2, \dots, T_{\text{ub}}\}$. Use helper variables $i_{\text{low}}, i_{\text{high}} \in \mathbb{N}$, that satisfy $i_{\text{low}} \leq T \leq i_{\text{high}}$ at all times during the algorithm. In each iteration, update i_{low} and/or i_{high} so that the number of remaining options for T is halved.*

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
```

[2] For the asymptotic bounds here and also in the following we assume $T \geq 2$ such that $\log(T) > 0$.

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