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 algorithm satisfies $T(n) \leq 3T(n/2) + 100n$, so $a = 3$ 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 *monotonously 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 $n \geq 10$.

Algorithm 2

(a)
```
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 $n = 2^k$.*

Algorithm 3

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

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."*

Note that we assume that A contains no duplicates, i.e. $A[i] \neq A[j]$ for all $i, j \in \{1 \dots n\}$ $i \neq j$. We say "the elements of A are pairwise distinct".

Further, the invariant should be "After j iterations of the outer for loop the j largest elements are at the correct place."

IB: $j = 1$

Assume the largest element is on position ℓ of the unmodified input array A.

After the first $\ell - 1$ iterations of the second for loop, it is still on position ℓ.

A: $\boxed{a_1 \mid a_2 \mid a_3 \mid \dots \mid a_{\ell-1} \mid a_\ell \mid \dots \mid a_n}$

$\max A = \max_{i \in \{1 \dots n\}} a_i = a_\ell$.

after $\ell - 1$ iterations we are here.

For all further steps with $i \geq \ell$, $A[i]$ contains the largest element and thus the largest element is swapped to position $i+1$. Hence, at the end of the inner for loop (end of first outer for loop iteration) the largest element is at position n.

IH:

Assume the invariant holds for $j = k$ for some $k \in \mathbb{N}$, i.e. after k iterations of the outer for loop the k largest elements are at the corret position.

IS: $k \rightsquigarrow k+1$

We must show that the invariant also holds for $j = k + 1$. By the induction hypothesis the k largest elements are at the correct position after k steps, i.e. at the positions $A[n - k + 1 \dots n]$. We now consider step $k + 1$. Note that in this iteration the positions of the k largest elements are not changed since for $i \geq n - k$, we will never have $A[i] > A[i + 1]$. Thus, in order to show $I(k + 1)$ it is enough to show that after step $k + 1$ also the $(k + 1)$st largest element is at the correct position. The $(k + 1)$st largest element is the largest element of $A[1 \dots n - k]$ (all elements that are larger than it come later by $I(k)$). Thus, by the argumentation in the base case, after $i = n - k - 1$ iterations in the second for-loop, it is at position $A[n - k]$. But for the other k iterations of the second for-loop, nothing changes as was already argued before (the largest elements do not change their position). Thus, after step $k + 1$, the $k + 1$ largest elements are at the correct position, which shows $I(k + 1)$.

Notice that there was not only made an argument for the $(k+1)^{th}$ largest element, but also how the rest of the array remains unchanged. This is really important, because the inner loop continues to loop until the end of

the array everytime. This motivates a slight modification of the Bubble Sort algorithm that reduces the number of steps taken in the inner loop. Can you find it?

Hint: the inner loop doesn't have to go until the end everytime, does it?

By the principle of mathematical induction, $I(j)$ is true for all $j \in \mathbb{N}$, $j \leq n$. In particular, $I(n)$ holds, which means that after the first n iterations the n largest elements are at the correct position. This shows that after n steps the array is sorted, which shows correctness of the Bubble Sort algorithm.

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

Suppose we are given a positive integer $N \in \mathbb{N}$, and a *monotonously 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.

The desired runtime already tells us a lot about how one might approach this exercise.

Algorithm 2

$T \leftarrow 1$
while $f(T) < N$ **do**
  $T \leftarrow T \cdot 2$
Return T

correctness: we compare N with $f(T_k)$ where $T_k = 2^k$ for each loop iteration k. If for any $\tilde{k}$ we have $f(T_{\tilde{k}}) \geq N$, we stop and return. Let $T_{ub} = T_{\tilde{k}}$.

To show that $T_{ub} \leq 2T$ notice that $f(T_{ub-1}) < N$. Since $f(T) \geq N$, by the monotonicity of f

$$T_{ub-1} \leq T \iff 2T_{ub-1} \leq 2T$$

and thus $T_{ub} \leq 2T$.

Runtime: Notice that $T_{\lceil \log_2 T \rceil} = 2^{\lceil \log_2 T \rceil} \geq T$, thus by the monotonicity of f we have

$f(T_{\lceil \log_2 T \rceil}) \geq f(T) \geq N$. Therefore the loop is executed at most $\lceil \log_2 T \rceil \leq O(\log T)$ times. Since f is called once per loop iteration, the number of calls to f is in $O(\log T)$ as well.

(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* $\{1, 2, \ldots, T_{\text{ub}}\}$. *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.*

refer to the official solutions.

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 $n \geq 10$.

refer to official solutions for (a)

Algorithm 3

(b)
```
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.*

Now for this exercise, you don't have to keep track of the exact number of calls to f, just the asymptotic complexity. For this type of exercise, if the algorithm is recursive, we want to use the master theorem.

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

Because the first double while loop is seperated from the for loop, we generate the following subproblems:

```
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)
```

The inner while loop always performs $2n$ calls to f. Thus

$$\sum_{i=0}^{n^2-1} 2n = 2n^3$$

The for loop has 4 iterations. $A(k)$ is called twice, so $4 \cdot 2 = 8$ times, but only if $k > 0$. For $n = 1$ we have $k = 0$, thus no recursive call is executed and we end up with $2n^3 = 2 \cdot 1^3 = 2$ calls, i.e. $T(1) = 2$. For $n \geq 2$ $k > 0$ and therefore $A(k)$ is executed twice each iteration, i.e. 8 times. Thus

$$T(n) = 2n^3 + 8T(\lfloor \tfrac{n}{2} \rfloor) \quad \text{for } n \geq 2.$$

Now we use the master theorem for $a = 8$, $b = 3$ and $c = 2$. Since $\log_b a = 3 = b$ we have

$$T(n) = \Theta(n^3 \log(n)) \quad \text{for } n \geq 2.$$

Notice for $n < 2$ we have $\log_2 n < 0$, which doesn't make sense in the context of O-notation.

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

Solution:

We show the statement suggested in the hint by mathematical induction.

- **Base Case.**
  We have $T(2) = 16 + 16T(1) = 32 \geq 2 = T(1)$, so the base case holds as the only $n' \in \mathbb{N}$ that is at most 1 is $n' = 1$.
- **Induction Hypothesis.**
  Assume that for some $k \in \mathbb{N}$ we have $T(k'+1) \geq T(k')$ for all $k' \in \mathbb{N}$ with $k' \leq k$.

IS: We show $T(k+2) \geq T(k+1)$.

We want to show

$$T(k+2) = 2(k+2)^3 + 8T\left(\left\lfloor \frac{k+2}{2} \right\rfloor\right) \geq 2(k+1)^3 + 8T\left(\left\lfloor \frac{k+1}{2} \right\rfloor\right) = T(k+1)$$

To do so, we have to show that

(1) $\left\lfloor \frac{k+1}{2} \right\rfloor \leq k$

(2) $\left\lfloor \frac{k+2}{2} \right\rfloor = \left\lfloor \frac{k+1}{2} \right\rfloor + 1$

So we can use our IH.

Notice that

$$\left\lfloor \frac{k+1}{2} \right\rfloor \leq \frac{k+1}{2} \overset{!}{\leq} k \iff k \geq 1 \quad (1)$$

Further we have:

- for $k = 2\ell + 1$ odd for some $\ell \in \mathbb{N}$

$$\left\lfloor \frac{k+2}{2} \right\rfloor = \left\lfloor \frac{2\ell+3}{2} \right\rfloor = \left\lfloor \ell + \frac{3}{2} \right\rfloor = \ell + 1 = \left\lfloor \frac{2\ell+2}{2} \right\rfloor = \left\lfloor \frac{k+1}{2} \right\rfloor$$

• for $k = 2\ell$ even for some $\ell \in \mathbb{N}$

$$\left\lfloor \frac{k+2}{2} \right\rfloor = \left\lfloor \frac{2\ell+2}{2} \right\rfloor = \ell + 1 = \left\lfloor \frac{2\ell+1}{2} \right\rfloor + 1 = \left\lfloor \frac{k+1}{2} \right\rfloor + 1$$

Meaning that $\left\lfloor \frac{k+2}{2} \right\rfloor$ is either $\left\lfloor \frac{k+1}{2} \right\rfloor + 1$ $\left\lfloor \frac{k+1}{2} \right\rfloor$. The first case satisfies (2) but the second case also works because (1) holds and we just say $T(\left\lfloor \frac{k+2}{2} \right\rfloor) = T(\left\lfloor \frac{k+1}{2} \right\rfloor)$.

Thus we can use the IH to get

$$T\left(\left\lfloor \frac{k+2}{2} \right\rfloor\right) \geq T\left(\left\lfloor \frac{k+1}{2} \right\rfloor\right)$$

which shows

$$T(k+2) = 2(k+2)^3 + 8T\left(\left\lfloor \frac{k+2}{2} \right\rfloor\right) \geq 2(k+1)^3 + 8T\left(\left\lfloor \frac{k+1}{2} \right\rfloor\right) = T(k+1).$$

By the principle of mathematical induction, for every $n \in \mathbb{N}$ we have for $n' \in \mathbb{N}$ with $n' \leq n$ that $T(n'+1) \geq T(n')$. In particular, $T(n+1) \geq T(n)$ is true for any $n \in \mathbb{N}$ and the function T is increasing.