

Departement of Computer Science

16 October 2023

Johannes Lengler, David Steurer

Lucas Slot, Manuel Wiedmer, Hongjie Chen, Ding Jingqiu

**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  $T : \mathbb{N} \rightarrow \mathbb{R}^+$  a function such that for all even  $n \in \mathbb{N}$ ,*

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

*Then for all  $n = 2^k$ ,  $k \in \mathbb{N}$ ,*

- *If  $b > \log_2 a$ ,  $T(n) \leq O(n^b)$ .*
- *If  $b = \log_2 a$ ,  $T(n) \leq O(n^{\log_2 a} \cdot \log n)$ .<sup>1</sup>*
- *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  $\Theta$  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

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

**Solution:**

We can apply the master theorem with  $a = 4$ ,  $b = 1$  and  $C = 100$ . In this case,  $b < \log_2 a$ , and therefore we have  $T(n) \leq O(n^{\log_2 a}) = O(n^2)$ .

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

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

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

**Solution:**

We can apply the master theorem with  $a = 1$ ,  $b = 1$  and  $C = \frac{3}{2}$ . In this case,  $b > \log_2 a$ , and therefore we have  $T(n) \leq O(n^b) = O(n)$ .

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

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

**Solution:**

We can apply the master theorem with  $a = 4$ ,  $b = 2$  and  $C = \frac{7}{2}$ . In this case,  $b = \log_2 a$ , and therefore we have  $T(n) \leq O(n^{\log_2 a} \cdot \log n) = O(n^2 \log n)$ .

**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
$\frac{n}{\log n} \leq O(\sqrt{n})$	<input type="checkbox"/>	<input type="checkbox"/>
$\log(n!) \geq \Omega(n^2)$	<input type="checkbox"/>	<input type="checkbox"/>
$n^k \geq \Omega(k^n)$ , if $1 < k \leq O(1)$	<input type="checkbox"/>	<input type="checkbox"/>
$\log_3 n^4 = \Theta(\log_7 n^8)$	<input type="checkbox"/>	<input type="checkbox"/>

**Solution:**

claim	true	false
$\frac{n}{\log n} \leq O(\sqrt{n})$	<input type="checkbox"/>	<input checked="" type="checkbox"/>
$\log n! \geq \Omega(n^2)$	<input type="checkbox"/>	<input checked="" type="checkbox"/>
$n^k \geq \Omega(k^n)$ , if $1 < k \leq O(1)$	<input type="checkbox"/>	<input checked="" type="checkbox"/>
$\log_3 n^4 = \Theta(\log_7 n^8)$	<input checked="" type="checkbox"/>	<input type="checkbox"/>

- (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
$\frac{n}{\log n} \geq \Omega(n^{1/2})$	<input type="checkbox"/>	<input type="checkbox"/>
$\log_7(n^8) = \Theta(\log_3(n^{\sqrt{n}}))$	<input type="checkbox"/>	<input type="checkbox"/>
$3n^4 + n^2 + n \geq \Omega(n^2)$	<input type="checkbox"/>	<input type="checkbox"/>
(*) $n! \leq O(n^{n/2})$	<input type="checkbox"/>	<input type="checkbox"/>

**Solution:**

claim	true	false
$\frac{n}{\log n} \geq \Omega(n^{1/2})$	<input checked="" type="checkbox"/>	<input type="checkbox"/>
$\log_7(n^8) = \Theta(\log_3(n^{\sqrt{n}}))$	<input type="checkbox"/>	<input checked="" type="checkbox"/>
$3n^4 + n^2 + n \geq \Omega(n^2)$	<input checked="" type="checkbox"/>	<input type="checkbox"/>
(*) $n! \leq O(n^{n/2})$	<input type="checkbox"/>	<input checked="" type="checkbox"/>

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.

**Solution:**

All claims except for the last one are easy to verify using either the theorem about the limit of  $\frac{f(n)}{g(n)}$  or simply the definitions of  $O$ ,  $\Omega$  and  $\Theta$ . Thus, we only present the solution for the last one.

Note that for all  $n \geq 1$ ,

$$n! \geq 1 \cdot 2 \cdots n \geq \lceil n/10 \rceil \cdots n \geq \lceil n/10 \rceil^{0.9n} \geq (n/10)^{0.9n}.$$

Let's show that  $(n/10)^{0.9n}$  grows asymptotically faster than  $n^{n/2}$ .

$$\lim_{n \rightarrow \infty} \frac{n^{n/2}}{(n/10)^{0.9n}} = \lim_{n \rightarrow \infty} 10^{0.9n} \cdot n^{-0.4n} = \lim_{n \rightarrow \infty} (10^{9/4}/n)^{0.4n} = 0.$$

Hence it is not true that  $(n/10)^{0.9n} \leq O(n^{n/2})$  and so it is not true that  $n! \leq O(n^{n/2})$ .

## 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  $j = 1, \dots, n$  do
  for  $i = 1, \dots, n - 1$  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.”

**Solution:**

We prove the invariant in the hint by mathematical induction on  $j$ .

- **Base Case.**

We prove the statement for  $j = 1$ . Assume that the largest element of  $A$  is at position  $l$  in the beginning. After the first  $l - 1$  iterations of the second for-loop, it is still at position  $l$ . For all further steps with  $i \geq l$ ,  $A[i]$  contains the largest element and thus the largest element is swapped to position  $i + 1$ . Hence, in the end the largest element is at position  $n$ , which shows  $I(1)$ .

- **Induction Hypothesis.**

We assume that the invariant is true for  $j = k$  for some  $k \in \mathbb{N}$ ,  $k < n$ , i.e. after  $k$  iterations the  $k$  largest elements are at the correct position.

- **Inductive Step.**

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  $I(k + 1)$ .

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 *monotonically increasing* function  $f : \mathbb{N} \rightarrow \mathbb{N}$ , meaning that  $f(i) \geq f(j)$  for all  $i, j \in \mathbb{N}$  with  $i \geq j$ . Assume that  $\lim_{n \rightarrow \infty} f(n) = \infty$ . 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  $f$ .<sup>2</sup> Prove that your algorithm is correct, and uses

---

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

at most the desired number of function calls.

**Solution:**

We loop over  $k = 1, 2, 3, \dots$ , setting  $T_k = 2^k$ . We terminate the loop at step  $k$  if  $f(T_k) \geq N$ , and return  $T_{\text{ub}} = T_k$ . To see that  $T_{\text{ub}}$  satisfies  $T_{\text{ub}} \leq 2T$ , note that  $T_{k-1} < T$  (as otherwise, we would have terminated the loop at step  $k-1$ ). For the runtime, note that  $T_{\lceil \log_2 T \rceil} = 2^{\lceil \log_2 T \rceil} \geq T$ , meaning  $f(T_{\lceil \log_2 T \rceil}) \geq N$  (since  $f$  is monotonically increasing, and  $f(T) \geq N$  by assumption). We conclude that the loop is executed at most  $\lceil \log_2 T \rceil = O(\log T)$  times, making one call to  $f$  each loop.

---

**Algorithm 2**

---

```

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

```

---

- (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  $i_{\text{low}}$  and/or  $i_{\text{high}}$  so that the number of remaining options for  $T$  is halved.

**Solution:**

We first run the algorithm of part (a) to obtain an integer  $T_{\text{ub}}$  with  $T_{\text{ub}} \leq 2T$ , making  $O(\log T)$  function call to  $f$ . Then, we run a modified binary search on the array  $[1, 2, \dots, T_{\text{ub}}]$  to find its smallest element  $T$  for which  $f(T) \geq N$ . The steps are given in the pseudo-code below.

---

**Algorithm 3**

---

```

 $i_{\text{low}} \leftarrow 1$ 
 $i_{\text{high}} \leftarrow T_{\text{ub}}$ 
 $i_{\text{mid}} \leftarrow \lceil (i_{\text{low}} + i_{\text{high}})/2 \rceil$ 
while  $i_{\text{low}} < i_{\text{high}}$  do
    if  $f(i_{\text{mid}}) \geq N$  then
         $\triangleright$  This implies  $T \leq i_{\text{mid}}$ .
        if  $f(i_{\text{mid}} - 1) < N$  then
             $\triangleright$  This implies  $T \geq i_{\text{mid}}$ , thus  $T = i_{\text{mid}}$ .
            Return  $i_{\text{mid}}$ 
        else
             $\triangleright$  This implies  $i_{\text{low}} \leq T < i_{\text{mid}}$ .
             $i_{\text{high}} \leftarrow i_{\text{mid}} - 1$ 
             $i_{\text{mid}} \leftarrow \lceil (i_{\text{low}} + i_{\text{high}})/2 \rceil$ 
        else
             $\triangleright$  This implies  $i_{\text{mid}} \leq T \leq i_{\text{high}}$ .
             $i_{\text{low}} \leftarrow i_{\text{mid}}$ 
             $i_{\text{mid}} \leftarrow \lceil (i_{\text{low}} + i_{\text{high}})/2 \rceil$ 
Return  $i_{\text{low}}$ 

```

---

For correctness, note that at all times during the algorithm, we have  $i_{\text{low}} \leq T \leq i_{\text{high}}$ . The algorithm terminates only when it returns  $T = i_{\text{mid}}$ , or when  $i_{\text{low}} = i_{\text{high}}$ , in which case it also

returns  $T = i_{\text{low}}$ . For the number of calls to  $f$ , note that the algorithm makes at most 2 calls per iteration (of the outer while-loop). In each iteration, the value of  $(i_{\text{high}} - i_{\text{low}} + 1)$  is halved. As  $i_{\text{high}} - i_{\text{low}} + 1$  is initially equal to  $T_{\text{ub}} \leq 2T$ , we have at most  $\lceil \log_2(2T) \rceil$  iterations, leading to at most  $2 \cdot \lceil \log_2(2T) \rceil = O(\log T)$  function calls in total.

*Note: we could have set  $i_{\text{low}} = T_{\text{ub}}/2 + 1$  at the start of the algorithm (instead of  $i_{\text{low}} = 1$ ), but this does not lead to an asymptotic improvement in the number of function calls!*

**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  $\Theta$  notation. For part (b), it is enough to give a maximally simplified asymptotic bound in  $\Theta$  notation. For the asymptotic bounds, you may assume that  $n \geq 10$ .

---

**Algorithm 4**

---

(a)  $i \leftarrow 1$   
**while**  $i \leq n$  **do**  
     $j \leftarrow i$   
    **while**  $2^j \leq n$  **do**  
         $f()$   
         $j \leftarrow j + 1$   
     $i \leftarrow i + 1$

---

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

**Solution:**

If  $i \leq \lfloor \log_2 n \rfloor$ , the inner loop performs  $\sum_{j=i}^{\lfloor \log_2 n \rfloor} 1 = \lfloor \log_2 n \rfloor - i + 1$  calls to  $f$ . If  $i > \lfloor \log_2 n \rfloor$ , it performs none. The full algorithm thus performs  $\sum_{i=1}^{\lfloor \log_2 n \rfloor} (\lfloor \log_2 n \rfloor - i + 1) = \lfloor \log_2 n \rfloor (\lfloor \log_2 n \rfloor + 1)/2 = \Theta((\log n)^2)$  calls to  $f$ .

---

**Algorithm 5**

---

(b) **function**  $A(n)$   
     $i \leftarrow 0$   
    **while**  $i < n^2$  **do**  
         $j \leftarrow n$   
        **while**  $j > 0$  **do**  
             $f()$   
             $f()$   
             $j \leftarrow j - 1$   
         $i \leftarrow i + 1$   
     $k \leftarrow \lfloor \frac{n}{2} \rfloor$   
    **for**  $l = 0 \dots 3$  **do**  
        **if**  $k > 0$  **then**  
             $A(k)$   
             $A(k)$

---

You may assume that the function  $T : \mathbb{N} \rightarrow \mathbb{R}^+$  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.

**Solution:**

Given  $i$ , the innermost loop performs  $\sum_{j=1}^n 2 = 2n$  calls to  $f$ . Hence, the second loop (guarded by  $i < n^2$ ) performs  $\sum_{i=0}^{n^2-1} 2n = 2n^3$  calls to  $f$ .

If  $\lfloor \frac{n}{2} \rfloor = 0$  (i.e.  $n = 1$ ), then  $k = 0$ , so the algorithm makes just 2 calls to  $f$ . Thus, we have  $T(1) = 2$ . For  $n \geq 2$ , we have  $k = \lfloor \frac{n}{2} \rfloor > 0$  and thus we get the following relation  $T(n) = 2n^3 + 8T(\lfloor \frac{n}{2} \rfloor)$ . Using the master theorem with  $a = 8$ ,  $b = 3$  and  $C = 2$ , we get  $(\log_2(8) = 3) T(n) = \Theta(n^3 \log(n))$  for any integer  $n \geq 2$  (we need  $n \geq 2$  so that  $\log(n) > 0$ ) since  $T$  is increasing. In conclusion, the algorithm performs  $\Theta(n^3 \log(n))$  calls to the function  $f$ .

(c)\* Prove that the function  $T : \mathbb{N} \rightarrow \mathbb{R}^+$  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  $k' \leq k$ .

- **Inductive Step.**

We must show that  $T(k + 2) \geq T(k + 1)$ . Together with the induction hypothesis this shows that  $T(k' + 1) \geq T(k')$  for all  $k' \in \mathbb{N}$  with  $k' \leq k + 1$ .

We have that

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

By the induction hypothesis

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

This is true since either  $\lfloor \frac{k+2}{2} \rfloor = \lfloor \frac{k+1}{2} \rfloor$  or  $\lfloor \frac{k+2}{2} \rfloor = \lfloor \frac{k+1}{2} \rfloor + 1$ . For the first case the inequality is actually an equality and the second case is covered by the induction hypothesis. Using the relation from above we get

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

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.