Eidgenössische
Technische Hochschule
Zürich

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

Departement of Computer Science
Markus Püschel, David Steurer
Johannes Lengler, Gleb Novikov, Chris Wendler, Ulysse Schaller

12. October 2020

Algorithms & Data Structures — Exercise sheet 4 — HS 20

Exercise Class (Room & TA): _____

Submitted by: _____

Peer Feedback by: _____

Points: _____

The solutions for this sheet are submitted at the beginning of the exercise class on October 19th.

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

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

If the function T is increasing, then the condition $n = 2^k$ can be dropped. If (1) holds with "$=$", then we may replace $\mathcal{O}$ with Θ in the conclusion.

This theorem generalizes some results that you have seen in this course. For example, the running time of Karatsuba algorithm satisfies $T(n) \leq 3T(n/2) + 100n$, so $a = 3$ and $b = 1 < \log_2 3$, hence $T(n) \in \mathcal{O}(n^{\log_2 3})$. 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 $T(n) \in \mathcal{O}(\log n)$.

In parts a), b) and c) of the first exercise you will see some examples of recurrences that can be analyzed in $\mathcal{O}$-Notation using Master theorem. These three examples show that the bounds in Master theorem are tight.

Exercise 4.1 *Solving Recurrences* **(1 point)**.

For this exercise, assume that n is a power of two (that is, $n = 2^k$, where $k \in \{0, 1, 2, 3, 4, \ldots\}$).

a) Consider the following algorithm:

Algorithm 1 $g(n)$

```
if n > 1 then
    for i = 1, ..., 5 do
        for j = 1, ..., n do
            for k = 1, ..., n do
                f()
    g(n/2)
    g(n/2)
else
    f()
```

The number of calls of f is given by the recurrence relation $T(1) = 1$ and $T(n) = 2T(\frac{n}{2}) + 5n^2$ for $n \geq 2$. Using mathematical induction show that $T(n) = 10 \cdot n^2 - 9n$.

Base case. Let $k = 0$, $n = 1$. $T(1) = 1 = 10 \cdot 1^2 - 9 \cdot 1$.

Induction Hypothesis. We assume that there is some $k \geq 0$ and $n = 2^k$ such that

$$T(n) = 10n^2 - 9n.$$

Inductive step ($k \to k + 1$). We know that $T(2n) = 2T(n) + 5 \cdot (2n)^2 = 2T(n) + 20n^2$. Using induction hypothesis for $T(n)$:

$$T(2n) = 2\left(10n^2 - 9n\right) + 20n^2 = 40n^2 - 18n = 10 \cdot (2n)^2 - 9 \cdot (2n).$$

as desired.

b) Consider the following algorithm:

Algorithm 2 $g(n)$

```
if n > 1 then
    for i = 1, ..., 3n do
        f()
    g(n/2)
    g(n/2)
else
    f()
    f()
```

The number of calls of f is given by the recurrence relation $T(1) = 2$ and $T(n) = 2T(\frac{n}{2}) + 3n$, for $n \geq 2$. Using mathematical induction show that $T(n) = 2n + 3n \log_2 n$.

Base case. Let $k = 0$. Then

$$T(2^0) = T(1) = 2 = 2 \cdot 2^0 + 3 \cdot 2^0 \cdot \log_2 2^0.$$

Induction Hypothesis. We assume that for some $k \geq 0$ it holds that

$$T(2^k) = 2 \cdot 2^k + 3 \cdot 2^k \cdot \log_2 2^k = 2 \cdot 2^k + 3 \cdot 2^k \cdot k.$$

Inductive step ($k \to k+1$). We know that

$$T(2^{k+1}) = 2T(2^k) + 3 \cdot 2^{k+1}.$$

Using induction hypothesis for $T(2^k)$:

$$\begin{aligned} T(2^{k+1}) &= 2\left(2 \cdot 2^k + 3 \cdot 2^k \cdot k\right) + 3 \cdot 2^{k+1} \\ &= 2 \cdot 2^{k+1} + 3 \cdot 2^{k+1} \cdot k + 3 \cdot 2^{k+1} \\ &= 2 \cdot 2^{k+1} + 3 \cdot 2^{k+1}(k+1), \end{aligned}$$

as desired.

c) Consider the following algorithm:

Algorithm 3 $g(n)$

```
if n > 1 then
    for i = 1, ..., 8 do
        g(n/2)
    for i = 1, ..., 4 do
        for j = 1, ..., n do
            for k = 1, ..., n do
                f()
else
    f()
    f()
    f()
```

The number of calls of f is given by the recurrence relation $T(1) = 3$ and $T(n) = 8T(\frac{n}{2}) + 4n^2$, for $n \geq 2$. Using mathematical induction show that $T(n) = 7n^3 - 4n^2$.

Base case. Let $k = 0$, $n = 1$. Then $T(1) = 3 = 7 \cdot 1^3 - 4 \cdot 1^2$.

Induction Hypothesis. We assume that for some $k \geq 0$ and $n = 2^k$ it holds that

$$T(n) = 7n^3 - 4n^2.$$

Inductive step ($k \to k+1$). We know that $T(2n) = 8T(n) + 4 \cdot (2n)^2 = 8T(n) + 16n^2$. Using induction hypothesis for $T(n)$:

$$T(2n) = 8\left(7n^3 - 4n^2\right) + 16n^2 = 56n^3 - 16n^2 = 7 \cdot (2n)^3 - 4 \cdot (2n)^2.$$

as desired.

The following definition is closely related to $\mathcal{O}$-Notation and is also useful in running time analysis of algorithms.

Definition 1 (Ω-Notation)**.** Let $f, g : N \to \mathbb{R}^+$. We say that $g \geq \Omega(f)$ if there exists $C > 0$ (that may depend on g) such that for all $n \in N$, $g(n) \geq Cf(n)$.

As for $\mathcal{O}$-Notation, typically $N = \mathbb{N}$, but sometimes we will consider other sets, e.g. $N = \{2, 3, 4, \ldots\}$.

Exercise 4.2 *Asymptotic notations.*

a) Show that $g \geq \Omega(f)$ if and only if $f \leq \mathcal{O}(g)$.
Solution:

Proof. To show the equivalence we show both directions:

The if-part $\Rightarrow$: If $g \geq \Omega(f)$ there exists a $C > 0$ such that for all $n \in N$ we have $Cf(n) \leq g(n)$. Thus, dividing by C on both sides yields $f(n) \leq \frac{1}{C}g(n)$, which means that $f \leq \mathcal{O}(g)$.

The only-if-part $\Leftarrow$: If $f \leq \mathcal{O}(g)$ there exists a $C > 0$ such that for all $n \in N$ we have $f(n) \leq Cg(n)$. Thus, dividing by C on both sides yields $\frac{1}{C}f(n) \leq g(n)$, which means that $g \geq \Omega(f)$.

□

b) As a reminder, we write $g = \Theta(f)$ if $g \leq \mathcal{O}(f)$ and $g \geq \Omega(f)$ at the same time. Describe the (worst-case) running time of the following algorithms in Θ-Notation.

1) Karatsuba algorithm. **Solution:** $\Theta(n^{\log_2(3)})$

2) Binary Search. **Solution:** $\Theta(\log_2(n))$

3) Bubble Sort. **Solution:** $\Theta(n^2)$

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

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

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.

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 $\mathcal{O}, \Omega$ and Θ. Thus, we only present the solution for the last one.

Consider $n!$ for an even n, by definition we have $n! = n(n-1)\cdots 1$. By grouping terms we can write $n!$ as a product of $\frac{n}{2}$ factors:

$$n! = \prod_{k=0}^{n/2-1} (n-k)(k+1).$$

To show that the claim is false we first show that each factor of this product is larger than n, for $n > 0$ and $k > 0$:

$$\begin{aligned} nk + n - k^2 - k > n &\Leftrightarrow nk > k^2 + k \\ &\Leftrightarrow n > n/2 + 1 > k + 1. \end{aligned}$$

Now, we can consider the quotient

$$\begin{aligned} \frac{n^{n/2}}{n!} &= \prod_{k=0}^{n/2-1} \frac{n}{(n-k)(k+1)} \\ &= \prod_{k=1}^{n/2-1} \frac{n}{(n-k)(k+1)}. \end{aligned}$$

We observe that, for $k, l \in \{0, \ldots, n/2-1\}$ with $k < l$, we have $(n-k)(k+1) < (n-l)(l+1)$ and that for $n > 4$, we have $\frac{n}{(n-1)(1+1)} < \frac{2}{3}$. Thus, for $n > 4$:

$$\begin{aligned} \frac{n^{n/2}}{n!} &\leq \prod_{k=1}^{n/2-1} \frac{n}{2n-2} \\ &\leq \left(\frac{2}{3}\right)^{n/2-1}. \end{aligned}$$

For $n \to \infty$ this value tends to zero. An analogous argument works for odd n. Thus, by Theorem 1.1. from the lecture we have that $n! \not\leq \mathcal{O}(n^{n/2})$.

Exercise 4.3 *Proving an invariant* **(1 point)**.

Let $n \in \mathbb{N}$ be an odd integer. Consider the following algorithm that starts with the list of all integers from 1 to $2n$ and returns a single integer:

Algorithm 4 $A(n)$

$L \leftarrow [1, 2, \ldots, 2n]$
while $\text{length}(L) > 1$ **do**
  Choose *any* two different elements a and b in L.
  Remove a and b from L, and add $|a - b|$ to L.
return $L[1]$

Here $\text{length}(L)$ denotes the number of elements contained in the list L, and $L[1]$ denotes its first element.

The goal of this exercise is to prove that, no matter how the two elements a and b are chosen, the algorithm will never return a zero.

a) Explain briefly why $A(n)$ always terminates. How many times does it enter the **while** loop ?

Solution: In each iteration of the **while** loop, $\text{length}(L)$ decreases by exactly 1. Therefore the list will eventually contain a single element, at which point the algorithm terminates by returning this element. The **while** loop is entered $2n - 1$ times.

b) Let $S(L) := \sum_{k \in L} k$ be the sum of all elements of L. Prove that the parity of $S(L)$ is an invariant of the algorithm, i.e. that after each iteration of the **while** loop, the value of S mod 2 is the same.

Solution:

Consider the list L at any point of the algorithm, and let L' denote the resulting list after going through the **while** loop once from L. We have to show that $S(L) \equiv S(L') \bmod 2$, which is equivalent to $S(L) - S(L') \equiv 0 \bmod 2$. Since L' is obtained from L by removing a and b and adding $|a - b|$, we have $S(L) - S(L') = a + b - |a - b|$. We distinguish two cases:

Case 1, $a \geq b$. In this case $|a-b| = a-b$, and therefore $S(L)-S(L') = a+b-(a-b) = 2b \equiv 0 \bmod 2$.

Case 2, $a < b$. In this case $|a-b| = b-a$, and therefore $S(L)-S(L') = a+b-(b-a) = 2a \equiv 0 \bmod 2$.

Therefore we always have $S(L)-S(L') \equiv 0 \bmod 2$ which proves that the parity of $S(L)$ is invariant.

c) Deduce that $A(n)$ never returns the number 0.

Solution:

Let L^* denote the final list (of length 1) obtained in the algorithm, i.e. $A(n)$ returns the integer $L^*[1]$. By part **b**, we know that $S(L^*) = L^*[1]$ has the same parity as $S([1, 2, \ldots, 2n])$, the sum of the elements of the initial list, which contains all integers from 1 to $2n$. We compute

$$S([1, 2, \ldots, 2n]) = \sum_{k=1}^{2n} k = \frac{2n(2n+1)}{2} = n(2n+1),$$

which is an odd number since n is odd by assumption and $2n + 1$ is always odd if $n \in \mathbb{N}$. Therefore, $L^*[1] \equiv S([1, 2, \ldots, 2n]) \equiv 1 \bmod 2$. In particular, it is impossible that $L^*[1]$ equals 0.

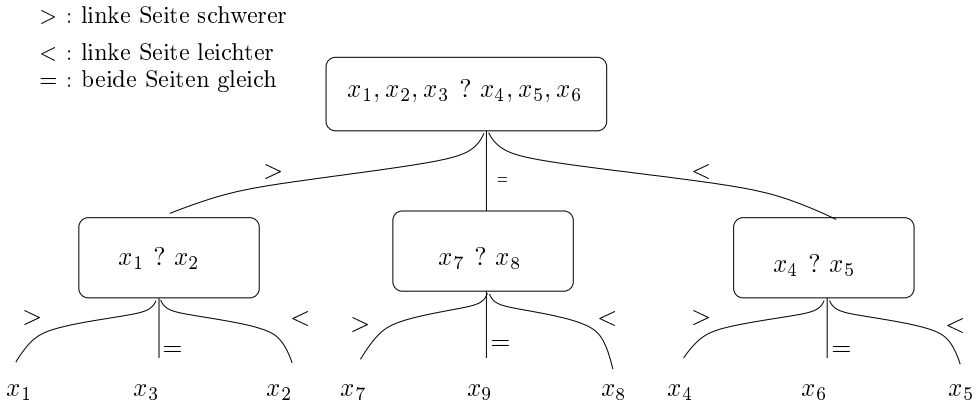
Figure 1: Balance scale.

Exercise 4.4 *Finding fake coins with a balance scale* **(1 point)**.

Imagine that you are given n '1 franc' coins of which k coins are fake. The fake coins are slightly heavier than the real ones, but all fake coins have the same weight. In order to determine which coins are fake, you are allowed to use a balance scale (see. Figure 1). Using the balance scale you can determine whether the coins you put onto the left side are heavier, lighter, or the same weight as the ones you put on the right side.

a) Consider the problem with $n = 9$ and $k = 1$. Draw a decision tree (called 'Entscheidungsbaum' in the lecture) for a strategy of your choice.

Solution:



b) Prove that for $k = 1$ even the best possible algorithm requires at least $\log_3(n) - 1$ comparisons to find the fake coin in the worst case.

Solution: Each node of the decision tree has at most three children. Thus, at depth i (root has depth 0) there are at most 3^i nodes.

At some nodes the algorithm prints out a result. Each possible outcome occurs in the tree, because each possible outcome must be recognized by a correctly working algorithm. Thus, the number of nodes must be at least as large as the number of possible outcomes n. For a decision tree of depth w, we have

$$n \leq \text{ number of nodes} \leq \sum_{i=0}^{w} 3^i = \frac{3^{w+1}-1}{2} < 3^{w+1}$$
$$\Rightarrow \log_3 n < w + 1 \Rightarrow w > \log_3 n - 1.$$

The depth w of the decision tree is equal to the number of comparisons required, which proves the claim.

c) For $k = 1$, provide an algorithm that finds the fake coin and requires exactly $\lceil \log_3(n) \rceil$ comparisons in the worst case.

Solution: We prove that such an algorithm exists by induction over $m := \lceil \log_3 n \rceil$. The proof yields the desired algorithm. The decision tree in a) already hints at the scheme of the algorithm.

Base case: For $m = 1$ we have $2 \leq n \leq 3$. We compare x_1 against x_2. If one of them is heavier, it is the fake coin. If they have the same weight, the coin x_3 must exist and it must be the fake coin.

Induction hypothesis: We assume that the there exists such an algorithm for some $m \in \mathbb{N}$.

Induction step $m \to m+1$**:** Consider a $n \in \mathbb{N}$ with $m + 1 = \lceil \log_3 n \rceil$. We must prove that we can find the fake coin within the n coins using $m + 1$ comparisons. In order to do so, we split the coins evenly into three sets M_1, M_2 and M_3. Each of the sets has either $\lceil n/3 \rceil$ or $\lceil n/3 \rceil - 1$ elements. Because there are three sets and $|M_1| + |M_2| + |M_3| = n$, two of them must have the same number of elements. W.l.o.g. let those sets be M_1 and M_2. We compare the set M_1 against M_2. If one of the sets is heavier, it contains the fake coin. If they have the same weight, the fake coin is in M_3.

In order to prove the claim, we now only need to prove that it is possible to find the fake coin in a set of $n' := \lceil n/3 \rceil$ coins using m comparisons. Because n satisfies $m + 1 = \lceil \log_3 n \rceil$, we have $m < \log_3 n \leq m + 1$. Thus, we also have $3^m < n \leq 3^{m+1}$ and in particular $3^{m-1} < n/3 \leq 3^m$. Because 3^m is an integer we even have $3^{k-1} < \lceil n/3 \rceil \leq 3^m$ which implies $m = \lceil \log_3 \lceil n/3 \rceil \rceil = \lceil \log_3 n' \rceil$. Now, we can apply our induction hypothesis to prove the step.

The existence of the desired algorithm now follows by the principle of mathematical induction. The proof was constructive, which means that it contains the algorithm.

d*) Prove that for $k \geq 1$ even the best algorithm requires at least $k \log_3(\frac{n}{k}) + (n-k) \log_3(\frac{n}{n-k}) - \mathcal{O}(\log(n))$ comparisons in the worst case.

For this exercise, you may use the so-called *Stirling approximation*

$$\ln(n!) = n \ln n - n + \Theta(\ln n)$$

without further justification.

Hint: Show using an decision tree that the number w of maximally required comparisons satisfies the inequality $3^{w+1} > \binom{n}{k}$.

Solution: The argument is the same one as in b), but, this time we have $\binom{n}{k}$ possible outcomes (recall that $\binom{n}{k}$ is the number of different ways to select k coins from n coins). Thus, we need the depth w to satisfy

$$\binom{n}{k} < 3^{w+1}. \tag{2}$$

Note that by the Stirling approximation

$$\log_3 n! = \frac{\ln n!}{\ln 3} = \frac{n \ln n - n + \Theta(\ln n)}{\ln 3} = n \log_3 n - n / \ln 3 + \Theta(\log n).$$

Solving (2) for w and using the above formula, we get:

$$\begin{aligned}
w &> \log_3 \binom{n}{k} - 1 \\
&= \log_3 \left(\frac{n!}{k!(n-k)!} \right) - 1 \\
&= \log_3 n! - \log_3 k! - \log_3 (n-k)! - 1 \\
&= n \log_3 n - k \log_3 k - (n-k) \log_3 (n-k) - n/\ln 3 + k/\ln 3 + (n-k)/\ln 3 \\
&\quad + \Theta(\log n) - \Theta(\log k) - \Theta(\log(n-k)) - 1.
\end{aligned}$$

Using the fact that $k, n-k \leq n$, we get that $\Theta(\log n) - \Theta(\log k) - \Theta(\log(n-k)) - 1 \geq -\mathcal{O}(\log n)$, and thus

$$\begin{aligned}
w &> n \log_3 n - k \log_3 k - (n-k) \log_3 (n-k) - \mathcal{O}(\log n) \\
&= k(\log_3 n - \log_3 k) + (n-k)(\log_3 n - \log_3 (n-k)) - \mathcal{O}(\log n) \\
&= k \log_3 \left(\frac{n}{k} \right) + (n-k) \log_3 \left(\frac{n}{n-k} \right) - \mathcal{O}(\log(n)).
\end{aligned}$$