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

28. September 2020

Algorithms & Data Structures — Exercise sheet 2 — 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 5th.

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

Exercise 2.1 *Bounding an inductive sequence* **(1 point)**.

Consider a sequence of natural numbers $(a(n))_{n\in\mathbb{N}_0}$ that is defined by

$$a(0) = 1, \qquad a(1) = 2, \qquad a(n) = 2a(n-1) + 3a(n-2) \quad \text{for } n \geq 2.$$

In this exercise, we will show that this sequence grows exponentially fast.

a) Use induction to show that for all $n \geq 4$, we have $a(n) \geq e^n$.

Solution:

Our induction hypothesis for $n \in \mathbb{N}$ is that $a(n) \geq e^n$ *and* $a(n-1) \geq e^{n-1}$. In order to get a base case, we need to compute some of the first values of the sequence $a(n)$. We have

$$a(0) = 1, \quad a(1) = 2, \quad a(2) = 7, \quad a(3) = 20, \quad a(4) = 61, \quad a(5) = 182.$$

In particular, $a(4) \geq e^4 \approx 54.6$ and $a(5) \geq e^5 \approx 148.4$, so the induction hypothesis holds for the base cases $n = 4$ and $n = 5$.

Now let $n > 5$ and suppose that the induction hypothesis holds for $n-1$, i.e. that $a(n-1) \geq e^{n-1}$ *and* $a(n-2) \geq e^{n-2}$. Then we can compute

$$a(n) = 2a(n-1) + 3a(n-2) \geq 2e^{n-1} + 3e^{n-2} \overset{3\geq e}{\geq} 2e^{n-1} + e^{n-1} = 3e^{n-1} \overset{3\geq e}{\geq} e^n.$$

This shows that our induction hypothesis holds for all $n \geq 5$. Remembering what the induction hypothesis is, this proves that $a(n) \geq e^n$ for all $n \geq 4$.

b) Show that for all $n \geq 0$, $a(n) \leq \mathcal{O}(3^n)$.

Solution:

We will prove by induction that $a(n) \leq 3^n$ for all $n \in \mathbb{N}$, which implies in particular that $a(n) \leq \mathcal{O}(3^n)$.

Similarly to part **a**, our induction hypothesis for $n \in \mathbb{N}$ is $a(n) \leq 3^n$ and $a(n-1) \leq 3^{n-1}$. The base case $n = 1$ is satisfied since $a(0) = 1 \leq 3^0$ and $a(1) = 2 \leq 3^1$. Let $n > 1$ and assume by induction that $a(n-1) \leq 3^{n-1}$ and $a(n-2) \leq 3^{n-2}$. Then we have

$$a(n) = 2a(n-1) + 3a(n-2) \leq 2 \cdot 3^{n-1} + 3 \cdot 3^{n-2} = 2 \cdot 3^{n-1} + 3^{n-1} = 3 \cdot 3^{n-1} = 3^n.$$

This concludes the induction proof that $a(n) \leq 3^n$ for all $n \in \mathbb{N}$.

c) Suppose that we redefine the starting values of the sequence, i.e. that for some $b, b' \in \mathbb{N}$, $(a(n))_{n \in \mathbb{N}}$ is given by

$$a(0) = b, \qquad a(1) = b', \qquad a(n) = 2a(n-1) + 3a(n-2) \quad \text{for } n \geq 2.$$

Show that for any choices of $b, b' \in \mathbb{N}$ we still have $a(n) \leq \mathcal{O}(3^n)$ for all $n \geq 0$.

Solution:

Let $b, b' \in \mathbb{N}$ be any natural numbers. Define $C := \max\{b, \frac{b'}{3}\}$. We will show by induction that for all $n \in \mathbb{N}$, we have $a(n) \leq C \cdot 3^n$. Our induction hypothesis for $n \in \mathbb{N}$ is that $a(n) \leq C \cdot 3^n$ and $a(n-1) \leq C \cdot 3^{n-1}$.

We have $a(0) = b \leq C = C \cdot 3^0$ and $a(1) = b' \leq 3C = C \cdot 3^1$, so the base case holds. Let $n > 1$ and assume by induction that $a(n-1) \leq C \cdot 3^{n-1}$ and $a(n-2) \leq C \cdot 3^{n-2}$. Similarly as in part **b**, we have

$$a(n) = 2a(n-1) + 3a(n-2) \leq 2C \cdot 3^{n-1} + 3C \cdot 3^{n-2} = C(2 \cdot 3^{n-1} + 3 \cdot 3^{n-2}) = C \cdot 3^n.$$

Thus, we have found $C > 0$ such that $a(n) \leq C \cdot 3^n$ for all $n \in \mathbb{N}$, which shows that $a(n) \leq \mathcal{O}(3^n)$.

Exercise 2.2 *Iterative squaring.*

In this exercise you are going to develop an algorithm to compute powers a^n, with $a \in \mathbb{Z}$ and $n \in \mathbb{N}$, efficiently. For this exercise, we will treat multiplication of two integers as a single elementary operation, i.e., for $a, b \in \mathbb{Z}$ you can compute $a \cdot b$ using one operation.

a) Assume that n is even, and that you already know an algorithm $A_{n/2}(a)$ that efficiently computes $a^{n/2}$, i.e., $A_{n/2}(a) = a^{n/2}$. Given the algorithm $A_{n/2}$, design an efficient algorithm $A_n(a)$ that computes a^n.

Solution:

Algorithm 1 $A_n(a)$

```
$x \leftarrow A_{n/2}(a)$
return $x \cdot x$
```

b) Let $n = 2^k$, for $k \in \mathbb{N}_0$. Find an algorithm that computes a^n efficiently. Describe your algorithm using pseudo-code.

Solution:

Algorithm 2 Power(a, n)

```
if $n = 1$ then
    return a
else
    $x \leftarrow$ Power$(a, n/2)$
    return $x \cdot x$
```

c) Determine the number of elementary operations (i.e., integer multiplications) required by your algorithm for part b) in $\mathcal{O}$-notation. You may assume that bookkeeping operations don't cost anything. This includes handling of counters, computing $n/2$ from n, etc.

Solution: Let $T(n)$ be the number of elementary operations that the algorithm from part b) performs on input a, n. Then

$$T(n) \leq T(n/2) + 1 \leq T(n/4) + 2 \leq T(n/8) + 3 \leq \ldots \leq T(1) + \log_2 n - 1 \leq \mathcal{O}(\log n)\,.$$

d) Let Power(a, n) denote your algorithm for the computation of a^n from part b). Prove the correctness of your algorithm via mathematical induction for all $n \in \mathbb{N}$ that are powers of two.

In other words: show that Power$(a, n) = a^n$ for all $n \in \mathbb{N}$ of the form $n = 2^k$ for some $k \in \mathbb{N}_0$.

- **Base Case.**
  Let $k = 0$. Then $n = 1$ and Power$(a, n) = a = a^1$.
- **Induction Hypothesis.**
  Assume that the property holds for some positive integer k. That is, Power$(a, 2^k) = a^{2^k}$.
- **Inductive Step.**
  We must show that the property holds for $k + 1$.

$$\text{Power}(a, 2^{k+1}) = \text{Power}(a, 2^k) \cdot \text{Power}(a, 2^k) \overset{\text{I.H.}}{=} a^{2^k} \cdot a^{2^k} = a^{2^{k+1}}\,.$$

By the principle of mathematical induction, this is true for any integer $k \geq 0$ and $n = 2^k$.

*e) Design an algorithm that can compute a^n for a general $n \in \mathbb{N}$, i.e., n does not need to be a power of two.

Hint: *Generalize the idea from part a) to the case where n is odd, i.e., there exists a $k \in \mathbb{N}$ such that $n = 2k + 1$.*

Solution:

Algorithm 3 Power(a, n)

```
if $n = 1$ then
    return a
else
    if $n$ is odd then
        $x \leftarrow$ Power$(a, (n-1)/2)$
        return $x \cdot x \cdot a$
    else
        $x \leftarrow$ Power$(a, n/2)$
        return $x \cdot x$
```

*f) Prove correctness of your algorithm in e) and determine the number of elementary operations in $\mathcal{O}$-Notation. As before, you may assume that bookkeeping operations don't cost anything.

Solution: Let's prove correctness.

- **Base Case.**
  Let $n = 1$. Then $\text{Power}(a, n) = a = a^1$.
- **Induction Hypothesis.**
  Assume that the property holds for all positive integers $m < n$. That is, $\text{Power}(a, m) = a^m$.
- **Inductive Step.**
  We must show that the property holds for n. If n is even,

$$\text{Power}(a, n) = \text{Power}(a, n/2) \cdot \text{Power}(a, n/2) \overset{\text{I.H.}}{=} a^{n/2} \cdot a^{n/2} = a^n.$$

  If n is odd,

$$\text{Power}(a, n) = a \cdot \text{Power}(a, (n-1)/2) \cdot \text{Power}(a, (n-1)/2) \overset{\text{I.H.}}{=} a \cdot a^{(n-1)/2} \cdot a^{(n-1)/2} = a^n.$$

  By the principle of mathematical induction, this is true for any integer $n \geq 1$.

Let $T(n)$ be the number of elementary operations that the algorithm Power performs on input a, n. Let's prove by induction that $T(n) \leq 2\log_2 n$.

- **Base Case.**
  Let $n = 1$. Then $T(n) = 0 \leq 2\log_2 n$.
- **Induction Hypothesis.**
  Assume that the property holds for all positive integers $m < n$. That is, $T(m) \leq 2\log_2 m$.
- **Inductive Step.**
  We must show that the property holds for n. If n is even,

$$T(n) \leq T(n/2) + 1 \overset{\text{I.H.}}{\leq} 2\log_2 n/2 + 1 < 2\log_2 n.$$

  If n is odd,

$$T(n) \leq T((n-1)/2) + 2 \overset{\text{I.H.}}{\leq} 2\log_2 (n-1)/2 + 2 < 2\log_2 n.$$

  By the principle of mathematical induction, this is true for any integer $n \geq 1$.

Exercise 2.3 *$\mathcal{O}$-Notation.*
Remark. In the following solutions, we prove the statements using the definition of $\mathcal{O}$-Notation. It is also perfectly fine to do it differently, for example using Theorem 1 from exercise sheet 0.

a) Write the following in the asymptotic $\mathcal{O}$-notation. Your answer should be simplified as much as possible. Unless otherwise stated, we assume $N = \mathbb{N} = \{1, 2, 3, \ldots\}$. You do not need to check that the involved functions take values in $\mathbb{R}^+$.

   1) $5n^3 + 40n^2 + 100$.
      Solution: $40n^2 \leq 40n^3$, $100 \leq 100n^3$, hence $5n^3 + 40n^2 + 100 \leq 145n^3$ for all $n \geq 1$, so $5n^3 + 40n^2 + 100 = \mathcal{O}(n^3)$

2) $2n \log_3 n^4$ with $N = \{2, 3, 4, \ldots\}$.
Solution: $\mathcal{O}(n \log n)$

We must show that for some positive C, and for all $n > 1$,

$$2n \log_3 n^4 \leq Cn \ln n.$$

This follows from a direct calculation. We use the formula for base change, $\log_b n = \frac{\log_a n}{\log_a b}$, and obtain

$$2n \log_3 n^4 = 8n \log_3 n = 8n \frac{\ln n}{\ln 3} = \frac{8}{\ln 3} n \ln n,$$

so the condition holds with $C := 8/\ln 3$.

b) Prove that if $f_1(x), f_2(x) \leq \mathcal{O}(g(x))$, then $f_1(x) + f_2(x) \leq \mathcal{O}(g(x))$.

Solution: Since both f_1 and f_2 are $\mathcal{O}(g(x))$, we know that for some n_0, for all $x \geq n_0$, there exist positive real numbers C_1, C_2 that:

$$f_1(x) \leq C_1 g(x)$$

and

$$f_2(x) \leq C_2 g(x).$$

Then $f_1(x) + f_2(x) \leq C_1 g(x) + C_2 g(x) = (C_1 + C_2) g(x)$ Thus, if we set $C_3 := C_1 + C_2$ then for all $x \geq n_0$,

$$f_1(x) + f_2(x) \leq C_3 g(x).$$

Thus we have shown that $f_1(x) + f_2(x) \leq \mathcal{O}(g(x))$.

c) Let $f_1(x), f_2(x), g(x) > 0$. Prove or disprove the following.

1) If $f_1(x), f_2(x) \leq \mathcal{O}(g(x))$ then $\frac{f_1(x)}{f_2(x)} \leq \mathcal{O}(1)$.

Solution: We will disprove this. Let $f_1(x) = x$, $f_2(x) = \sqrt{x}$, and $g(x) = x$. $x \leq \mathcal{O}(x)$, $\sqrt{x} \leq \mathcal{O}(x)$, but $\frac{f_1(x)}{f_2(x)} = \sqrt{x} \nleq O(1)$.

2) If $f_1(x) \leq \mathcal{O}(g(x))$ and $f_2(x) \leq \mathcal{O}(\frac{1}{g(x)})$, then $f_1(x) f_2(x) \leq \mathcal{O}(1)$.

Solution: Because $f_1(x) \leq \mathcal{O}(g(x))$, there exists C_1 such that $f_1(x) \leq C_1 g(x)$ for all $x \geq 1$.

Because $f_2(x) \leq \mathcal{O}(\frac{1}{g(x)})$, there exists C_2 such that $f_2(x) \leq C_2 \frac{1}{g(x)}$ for all $x \geq 1$.

Assume $x \geq 1$. Then

$$f_1(x) f_2(x) \leq (C_1 g(x)) \left(C_2 \frac{1}{g(x)} \right) = C_1 C_2$$

Thus:

$$\exists C_3 > 0 . \forall x \geq 1, f_1(x) f_2(x) \leq C_3 * 1$$

In this case $C_3 = C_1 C_2$. From this we have shown that. $f_1(x) f_2(x) \leq \mathcal{O}(1)$.

Exercise 2.4 *Towers of Hanoi* **(2 points)**.

In this exercise you should design a recursive divide-and-conquer algorithm for solving the *Tower of Hanoi* puzzle. The puzzle consists of three rods A, B and C, and n disks of different sizes, which we number from 1 (smallest) to n (largest). The disks can slide onto any rod. The puzzle starts with all the disks stacked in ascending order (largest on bottom, smallest on top) on rod A (see Figure 1).

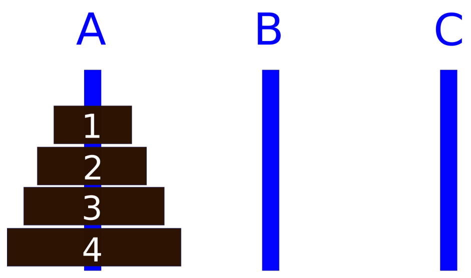

Figure 1: Initial state of the Tower of Hanoi game for $n = 4$.

Now, the goal is to move all the disks to rod C. When moving the disks the following rules must be obeyed:

1. Only one disk can be moved at a time.
2. Each move consists of taking the uppermost disk from one of the stacks and placing it on top of another stack or an empty rod.
3. No larger disk may be placed on a smaller disk.

a) Develop an algorithm that solves the problem for $n = 1$.

Solution: If there is only one disk we can just move the disk from rod A to rod C.

b) Assume that you have an algorithm `Move(source, target, spare)` that can move $n - 1$ disks from a source rod to a target rod using a spare rod and use it to solve the puzzle with n disks.

Solution:

Algorithm 4 `SolveHanoiWithHelper(source, target, spare,` n`)`

```
Move(source, spare, target)
Move the remaining disk from source to target.
Move(spare, target, source)
```

c) Make use of the insights you gained in a) and b) in order to complete the pseudo-code of `SolveHanoi`. Calling `SolveHanoi`(A, C, B, n) should solve the puzzle.

Solution:

Algorithm 5 `SolveHanoi(source, target, spare,` n`)`

```
if $n > 0$ then
    SolveHanoi(source, spare, target, $n - 1$)
    Move the uppermost disk from source to target.
    SolveHanoi(spare, target, source, $n - 1$)
```

d) Proof the correctness of `SolveHanoi`(A, C, B, n) for all $n \in \mathbb{N}$ by induction.

Solution:

Base Case: For $n = 1$ the algorithm calls `SolveHanoi(source, spare, target, 0)`, which does nothing because the last argument is 0. Next, the disk is moved from `source` to `target`. Finally, `SolveHanoi(spare, target, source, 0)` is called, which does nothing because the last argument is 0. Thus, the disk is at the target rod and the algorithm solved the problem.

Induction Hypothesis: `SolveHanoi(source, target, spare,` k`)` correctly moves k disks from the source rod to the target rod using the spare rod for some $k \in \mathbb{N}$.

Induction Step $k \to k+1$: `SolveHanoi(source, spare, target,` $k+1$`)` calls `SolveHanoi(source, spare, target,` k`)`, which according to the induction hypothesis moves k disks from the source rod to the spare rod using the target rod as auxiliary rod. Then, the last disk is moved from `source` to `target`. Finally, `SolveHanoi(spare, target, source,` k`)` is called, which according to the induction hypothesis moves k disks from the spare rod to the target rod using the source rod as auxiliary rod. Thus, all the $k+1$ disks are on the target rod and the algorithm solved the problem.

e) How many moves are performed by `SolveHanoi(`A, C, B, n`)` in order to solve the puzzle? **Hint:** Let T_n denote the number of moves required by `SolveHanoi(`A, C, B, n`)` and T_{n-1} the number of moves required by `SolveHanoi(`$*$, $*$, $*$, $n-1$`)` (where $*$ is a placeholder). Think about how T_n and T_{n-1} relate to each other. The relationship between T_n and T_{n-1} is called a *recurrence relation.*

Solution: The recurrence relation for `SolveHanoi` is given by $T_n = 2T_{n-1} + 1$ and $T_1 = 1$. Expanding it yields

$$\begin{aligned} T_n &= 2T_{n-1} + 1 \\ &= 2(2T_{n-2} + 1) + 1 = 4T_{n-2} + 3 \\ &= 4(2T_{n-3} + 1) + 3 = 8T_{n-3} + 7 \\ &= 8(2T_{n-4} + 1) + 7 = 16T_{n-4} + 15 \\ &\vdots \end{aligned}$$

We see an emerging pattern of the form

$$T_n = 2^k T_{n-k} + 2^k - 1. \tag{1}$$

Plugging $k = n-1$ in (1) and using the fact that $T_1 = 1$, we would get

$$T_n = 2^{n-1} T_1 + 2^{n-1} - 1 = 2^n - 1. \tag{2}$$

Now let's actually prove (2) by induction on n. The base case $n = 1$ holds since $2^1 - 1 = 1 = T_1$. Suppose that (2) holds for some $n \geq 1$ and let's show it holds for $n+1$ as well. Using the recurrence relation and the induction hypothesis, we have

$$T_{n+1} = 2T_n + 1 = 2 \cdot (2^n - 1) + 1 = 2^{n+1} - 1,$$

which concludes the inductiove proof of (2).