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

21 September 2020

Algorithms & Data Structures — Exercise sheet 1 — 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 September 28th.

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

Exercise 1.1 *Proper Power* **(2 points)**.

A natural number $n > 1$ is called a *proper power* if there exist two natural numbers $a, b > 1$ such that n can be written as $n = a^b$.

Consider the following algorithm that, given a natural number $n > 1$ as input, decides whether n is a proper power or not. Note that the algorithm depends on a function f that determines how many for-loop iterations are performed at most. For now assume $f(n) = n$.

Algorithm 1 ProperPowerTest(n)

```
for a = 2, ..., f(n) do
    b ← 2
    while a^b < n do
        b ← b + 1
    if ISPOWER(n, a, b) then
      return "n is a proper power"
return "n is not a proper power"
```

The procedure ISPOWER(n, a, b) is called **once per iteration of the for-loop** and returns true if and only if $n = a^b$.

Answer the following questions:

a) Let $T(n)$ be the number of calls of ISPOWER that this algorithm makes on the input n. Draw a graph of $T(n)$ for $n = 2, 3, \ldots, 30$ (i.e., x-axis: n, y-axis: $T(n)$).

Solution:

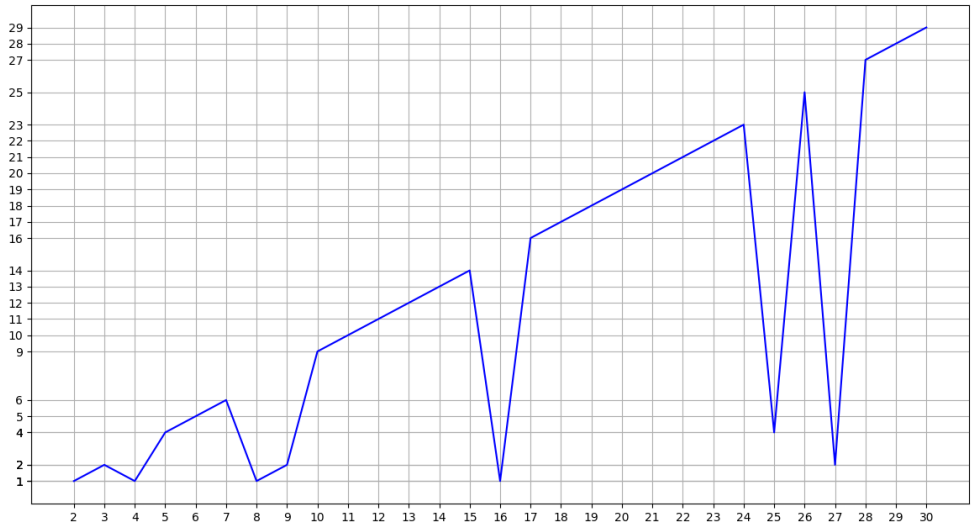

Figure 1: $T(n)$ for the first algorithm (with $f(n) = n$).

b) How many calls of ISPOWER does this algorithm make in the worst case? That is, what is the largest possible number of calls of the procedure ISPOWER that this algorithm can make (in terms of n)? Which numbers n correspond to this case?

Solution: $n - 1$ if and only if n is not a proper power.

c) How many calls of ISPOWER does this algorithm make in the best case? That is, what is the smallest possible number of calls of the procedure ISPOWER that this algorithm can make (in terms of n)? Which numbers n correspond to this case?

Solution: 1 if and only if n is a power of 2.

d) The for-loop in Algorithm 1 ranges from 2 to $f(n)$. Is it possible to perform fewer than $f(n) = n$ for-loop iterations in Algorithm 1? Determine the smallest value of $f(n)$ such that Algorithm 1 works.

Solution: $f(n) = \lfloor \sqrt{n} \rfloor$.

Consider the following algorithm. Note that the algorithm depends on the function $f(n)$, which determines the number of for-loop iterations. For now, please assume that $f(n) = n$.

Algorithm 2 ImprovedProperPowerTest(n)

```
for b = 2, ..., f(n) do
    a ← 2
    while a^b < n do
        a ← a + 1
    if ISPOWER(n, a, b) then
      return "n is a proper power"
return "n is not a proper power"
```

e) Find a function f such that $f(n)$ is as small as possible.

Hint: The correct f leads to an algorithm that requires less than $\lfloor \sqrt{n} \rfloor$ calls of ISPOWER.

Solution: $f(n) = \lfloor \log_2 n \rfloor$.

f) Answer the questions from points a), b) and c) for the second algorithm using your f from task e).

Solution:

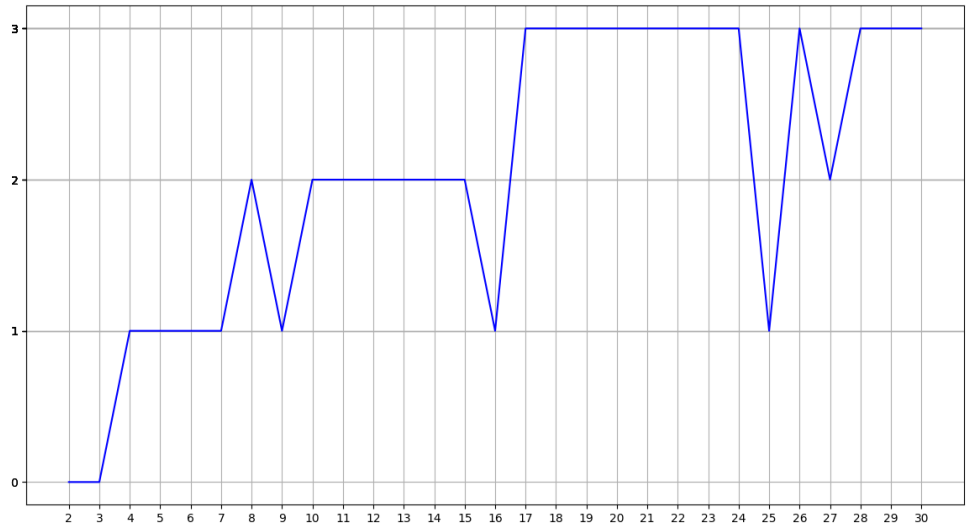

Figure 2: $T(n)$ for the second algorithm (with $f(n) = \lfloor \log_2(n) \rfloor$).

In the worst case, i.e., when n is not a proper power, Algorithm 2 requires $\lfloor \log_2(n) \rfloor - 1$ calls of ISPOWER. For $n > 3$, the best case is when n is a square and Algorithm 2 requires 1 call of ISPOWER (for $n = 2$ and $n = 3$ Algorithm 2 requires 0 calls of ISPOWER).

Exercise 1.2 *Induction.*

a) Prove by mathematical induction that for any positive integer n,

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}.$$

- **Base Case.**
  Let $n = 1$. Then:
  $$1 = \frac{1 \cdot 2}{2}.$$
- **Induction Hypothesis.**
  Assume that the property holds for some positive integer k. That is:
  $$1 + 2 + \cdots + k = \frac{k(k+1)}{2}.$$

- **Inductive Step.**
  We must show that the property holds for $k+1$ summands.

$$\begin{aligned} 1+2+\cdots+k+k+1 &\overset{\text{I.H.}}{=} \frac{k(k+1)}{2}+k+1 \\ &= \frac{k(k+1)+2(k+1)}{2} \\ &= \frac{(k+1)(k+2)}{2}\,. \end{aligned}$$

  By the principle of mathematical induction, this is true for any positive integer n.

b) Prove via mathematical induction that for all integers $n \geq 5$,

$$2^n > n^2\,.$$

- **Base Case.**
  Let $n = 5$. Then:

$$2^5 = 32 > 25 = 5^2\,.$$

- **Induction Hypothesis.**
  Assume that the property holds for some positive integer k. That is:

$$2^k > k^2\,.$$

- **Inductive Step.**
  We must show that the property holds for $k+1$.

$$\begin{aligned} 2^{k+1} &= 2 \cdot 2^k \\ &\overset{\text{I.H.}}{>} 2 \cdot k^2 \\ &= k^2 + k^2 \\ &\geq k^2 + 5k \\ &= k^2 + 2k + 3k \\ &\geq k^2 + 2k + 15 \\ &> k^2 + 2k + 1 \\ &= (k+1)^2\,. \end{aligned}$$

  By the principle of mathematical induction, this is true for any positive integer n.

Exercise 1.3 *$\mathcal{O}$-Notation.*

a) Prove or disprove the following statements:

1) $n^2 \leq \mathcal{O}(3n^4 + n^2 + n)$.

   Solution: We have $n^2 \leq 1 \cdot (3n^4 + n^2 + n)$ for all natural numbers, therefore, $n^2 \leq \mathcal{O}(3n^4 + n^2 + n)$.

2) $\log_7(n^8) \leq \mathcal{O}(\log(n^{\sqrt{n}}))$.

Solution: We have $\log_7(n^8) = 8\log_7(n)$ and $\log(n^{\sqrt{n}}) = \sqrt{n}\log(n)$. Moreover, the fraction $\log_7(n)/\log(n) = 1/\log(7)$ is constant (see also remark on sheet 0). Hence, by Theorem 1 from Exercise sheet 0, $\log_7(n^8) \leq \mathcal{O}(\log(n^{\sqrt{n}}))$.

3) $n^{1/3} \leq \mathcal{O}(\frac{n}{\log n})$.

Solution: We consider the limit

$$\lim_{x\to\infty} \frac{x^{1/3}}{\frac{x}{\log x}} = \lim_{x\to\infty} \frac{\log(x)}{x^{2/3}}$$

and apply L'Hôpital's rule to obtain

$$\lim_{x\to\infty} \frac{(\log(x))'}{(x^{2/3})'} = \lim_{x\to\infty} \frac{x^{1/3}}{2/3x} = 0.$$

Hence, by Theorem 1 from Exercise sheet 0, $n^{1/3} \leq \mathcal{O}(\frac{n}{\log n})$.

4) $\sum_{k=0}^{n} k \leq \mathcal{O}(n\log n)$.

Hint: *You can use the formula from Exercise 1.2.a.*

Solution: We have $\sum_{k=0}^{n} k = \frac{n(n+1)}{2} = \frac{n^2}{2} + \frac{n}{2}$. Therefore, by applying Theorem 1 from Exercise sheet 0, $\sum_{k=0}^{n} k \not\leq \mathcal{O}(n\log n)$.

b) Place the following functions in order such that if f appears before g, it means that $f \leq \mathcal{O}(g)$. If for some functions it holds that $f \leq \mathcal{O}(g)$ and $g \leq \mathcal{O}(f)$, please indicate so.

$$n^2 + 2n + 1,\ \ n\sum_{k=0}^{n} k,\ \ \frac{n}{\ln n},\ \ n\ln(n^n),\ \ \sqrt{n}\ln(n),\ \ n\ln(n^2),\ \ \sum_{k=0}^{n} k^2,\ \ n\ln(2^n)$$

Hint: *You can use formulas from Exercise 1.2.a and Exercise 0.1.a.*

Solution:

$$\sqrt{n}\ln(n),\ \ \frac{n}{\ln n},\ \ n\ln(n^2),\ \ n^2 + 2n + 1,\ \ n\ln(2^n),\ \ n\ln(n^n),\ \ n\sum_{k=0}^{n} k,\ \ \sum_{k=0}^{n} k^2,$$

which can be straightforwardly obtained by considering the simplified expressions

$$\sqrt{n}\ln(n),\ \ \frac{n}{\ln n},\ \ 2n\ln(n),\ \ n^2 + 2n + 1,\ \ n^2\ln(2),\ \ n^2\ln(n),\ \ n\cdot\frac{n(n+1)}{2},\ \ \frac{n(n+1)(2n+1)}{6}$$

and applying Theorem 1 from Exercise sheet 0.

In four case, two terms are asymptotically equivalent and can be swapped:

- $n^2 + 2n + 1 \leq \mathcal{O}(n\ln(2^n))$
- $n\ln(2^n) \leq \mathcal{O}(n^2 + 2n + 1)$
- $n\sum_{k=0}^{n} k \leq \mathcal{O}(\sum_{k=0}^{n} k^2)$
- $\sum_{k=0}^{n} k^2 \leq \mathcal{O}(n\sum_{k=0}^{n} k)$

c)* Prove the following statements about $n!$:

1) $n! \leq n^n$.

Solution: By definition we have $n! = n(n-1)\cdots 1 \leq n \cdot n \cdots n = n^n$.

2) $\ln(n!) \leq \mathcal{O}(n \ln n)$.

Solution: On $\mathbb{R}^+$, $\ln x$ is monotonically increasing and we have $n \ln n = \ln(n^n)$. Therefore, for all $n \in \mathbb{N}$, $\ln(n!) \leq 1 \cdot (n \ln n)$, and $\ln(n!) \leq \mathcal{O}(n \ln n)$.

3) $\left(\frac{n}{2}\right)^{n/2} \leq n!$.

Solution: We have $n! = n(n-1)\cdots\lceil n/2\rceil\cdots 1 \geq n(n-1)\cdots\lceil n/2\rceil$, where $\lceil x\rceil$ is the smallest integer $m \geq x$. So there are $\lceil n/2\rceil$ factors that are at least $\lceil n/2\rceil \geq n/2$. Hence $n! \geq (\frac{n}{2})^{n/2}$.

4) $n \ln n \leq \mathcal{O}(\ln(n!))$.

Solution: By the monotonicity of the logarithm we have

$$\ln(n!) \geq \ln((\frac{n}{2})^{n/2}) = \frac{n}{2}(\ln n - \ln 2),$$

so $n \ln n \leq 2\ln(n!) + 2\ln 2$. By Theorem 1 from Exercise sheet 0, $n \ln n \leq \mathcal{O}(\ln(n!))$.

d)* Let $f : \mathbb{N} \to \mathbb{R}^+$ and $g : \mathbb{N} \to \mathbb{R}^+$. Is it always true that either $g \leq \mathcal{O}(f)$ or $f \leq \mathcal{O}(g)$ or both? What if both f and g are strictly increasing?

Solution: For general f and g it is not always true. For example, consider

$$f(n) = \begin{cases} n & \text{if } n \text{ is even} \\ 1 & \text{otherwise} \end{cases}$$

and $g(n) = \sqrt{n}$. Consider any constant $C > 0$. For all even $n > C^2$, $f(n) > Cg(n)$, and for all odd $n > C^2$, $g(n) > Cf(n)$.

For strictly increasing functions f and g the statement is also not true. A similar trick can be applied. For example, consider

$$f(n) = 2^{\lceil \frac{n+1}{2}\rceil n}$$

and

$$g(n) = 2^{(\lceil \frac{n}{2}\rceil + \frac{1}{2})n}.$$

Consider any constant $C > 0$. For all even $n > 2\log_2 C$ we have

$$f(n) = 2^{n^2/2+n} = 2^{n/2} \cdot 2^{n^2/2+n/2} > C \cdot 2^{n^2/2+n/2} = Cg(n),$$

and for all odd $n > 2\log_2 C$ we have

$$g(n) = 2^{n^2/2+n/2+n/2} = 2^{n/2} \cdot 2^{n^2/2+n/2} > C \cdot 2^{n^2/2+n/2} = Cf(n).$$

Exercise 1.4 *Divisibility check algorithm* **(1 point)**.

Consider the following algorithm that, given an n-digit number $a > 0$ with decimal expression $a_{n-1} \ldots a_0$, decides whether a is divisible by 19 or not:

Algorithm 3 DivisibilityCheck(a)

if $a < 19$ **then**
  return "a is not divisible by 19"
else if $a = 19$ **then**
  return "a is divisible by 19"
else
  $a \leftarrow a_{n-1} \dots a_1$
  $b \leftarrow a + 2 \cdot a_0$
  return DivisibilityCheck(b)

For example, if we feed the number 347 to DivisibilityCheck, it would go over the numbers

$$347 \quad \rightarrow \quad 34\cancel{7} + 2 \cdot 7 = 48 \quad \rightarrow \quad 4\cancel{8} + 2 \cdot 8 = 20 \quad \rightarrow \quad 2\cancel{0} + 2 \cdot 0 = 2,$$

and the algorithm would therefore return "347 is not divisible by 19" since it is called for the last time with the number $2 \neq 19$. The goal of this exercise is to prove the correctness of this algorithm.

a) Show that b is divisible by 19 if and only if a is divisible by 19.

Hint: *Write b in terms of a and a_0 only.*

Solution:

Notice that $a_{n-1} \dots a_1 = \frac{a-a_0}{10}$, and therefore $b = \frac{a-a_0}{10} + 2a_0 = \frac{a+19a_0}{10}$, or equivalently $a = 10b - 19a_0$.

We first show that if 19 divides b, then 19 divides a. If b is divisible by 19, there exists a natural number d such that $b = 19d$. But then $a = 10b - 19a_0 = 10 \cdot 19d - 19a_0 = 19(10d - a_0)$, so a is also divisible by 19.

Conversely, if 19 divides a, then there exists a natural number d' such that $a = 19d'$. Thus $b = \frac{a+19a_0}{10} = 19\frac{d'+a_0}{10}$. By construction, we know that b is a natural number, and moreover 19 is prime, so in particular 10 and 19 do not have any common divisor (except 1). So $b = 19\frac{d'+a_0}{10}$ implies that b is also divisible by 19.

b) Show that if $a > 19$, then $b < a$.

Hint: *Consider the difference $a - b$.*

Solution:

Note that $b < a \Leftrightarrow a - b > 0$, so it is enough to show that $a - b > 0$. Using part **a**, we have

$$a - b = a - \tfrac{a+19a_0}{10} = \tfrac{1}{10}(9a - 19a_0).$$

We have $a_0 \leq 9$ since it is a digit of the number a. Therefore

$$a - b = \tfrac{1}{10}(9a - 19a_0) \geq \tfrac{1}{10}(9a - 19 \cdot 9) = \tfrac{9}{10}(a - 19),$$

which is stricly larger than 0 because $a > 19$.

c) Prove by mathematical induction that DivisibilityCheck is a correct algorithm.

Hint: *Use mathematical induction in the following variant:*

(a) Prove the statement for some base cases $a = 1, \dots, k$.

(b) For $m > k$, prove that if the statement is true for all $a = 1, \ldots, m-1$, then the statement is also true for $a = m$.

Then you can conclude that the statement is true for all natural numbers $a \geq 1$.

Solution:

First of all, notice that if we feed some $a > 0$ to the algorithm, then we will only have calls of DivisibilityCheck on positive numbers (i.e. we will never have $b = 0$). Indeed, if $1 \leq a \leq 19$, then DivisibilityCheck is called only once on a, and if $a > 19$, then in its decimal expression $a_{n-1} \ldots a_0$ we have $n \geq 2$ and $a_{n-1} > 0$, and thus b is also positive.

Now we can show the correctness of DivisibilityCheck(a) by induction on a. The induction hypothesis is: for all $1 \leq a' \leq a$, the output of DivisibilityCheck(a') is correct.

We start by showing the base case with $a = 19$. If $1 \leq a' \leq 19$, DivisibilityCheck(a') will output "a' is divisible by 19" if and only if $a' = 19$, which is clearly the correct output.

For the induction step, let $a > 19$ and suppose that the induction hypothesis holds for $a - 1$, i.e. the output is correct for all $1 \leq a' \leq a - 1$. We want to show that it also holds for a, i.e. that DivisibilityCheck(a') gives the correct output for all $1 \leq a' \leq a$. By the induction hypothesis, we already know that the output is correct for all $1 \leq a' \leq a - 1$, so we only need to show that DivisibilityCheck(a) gives the correct output. Since $a > 19$, there will be a call of DivisibilityCheck(b). We have shown above that $b \geq 1$. Since $a > 19$, by part **b** we also know that $b < a$. Thus $1 \leq b \leq a - 1$, so by the induction hypothesis DivisibilityCheck(b) gives the correct output. Therefore, DivisibilityCheck(a) returns "a is divisible by 19" if and only if b is divisible by 19, and by part **a** this concludes the proof of its correctness.

d)* Show that the number of recursive calls made by DivisibilityCheck(a) is $\mathcal{O}(\log a)$.

Solution:

Note that if $a > 42$, then (using $a_0 \leq 9$)

$$b = \tfrac{a+19a_0}{10} \leq \tfrac{a+19\cdot 9}{10} < \tfrac{a+4\cdot 43}{10} \leq \tfrac{a+4a}{10} = \tfrac{a}{2}.$$

Therefore, as long as $a > 42$, every call of DivisibilityCheck(a) yields a recursive call of DivisibilityCheck(b) with $b < a/2$. Moreover, when $1 \leq a \leq 42$, a call of DivisibilityCheck(a) either yields an answer (i.e. outputs "a is (not) divisible by 19"), or yields a call of DivisibilityCheck(b) with $b \leq a-1$ (by part **b**). Thus, the total number of recursive calls made by DivisibilityCheck(a) is at most $\log_2(a) + 42 = \mathcal{O}(\log a)$.