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

23 September 2019

Algorithms & Data Structures — Exercise sheet 1 — HS 19

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 30th.

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

Exercise 1.1 *Primality* **(1 point)**.

Recall that a natural number $n > 1$ is called *prime* if it cannot be written as the product of two (strictly) smaller natural numbers.

Consider the following algorithm that, given a natural number $n > 1$ as input, decides whether n is prime or not:

Algorithm 1 SimplePrimalityTest(n)

```
k ← 2
while DIVIDES(k, n) is false do
    k ← k + 1
if k = n then
    return "n is prime"
else
    return "n is not prime"
```

where DIVIDES(k, n) is a procedure that returns true if and only if k divides n without a remainder.

Answer the following questions:

a) Let $T(n)$ be the number of calls of DIVIDES 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:

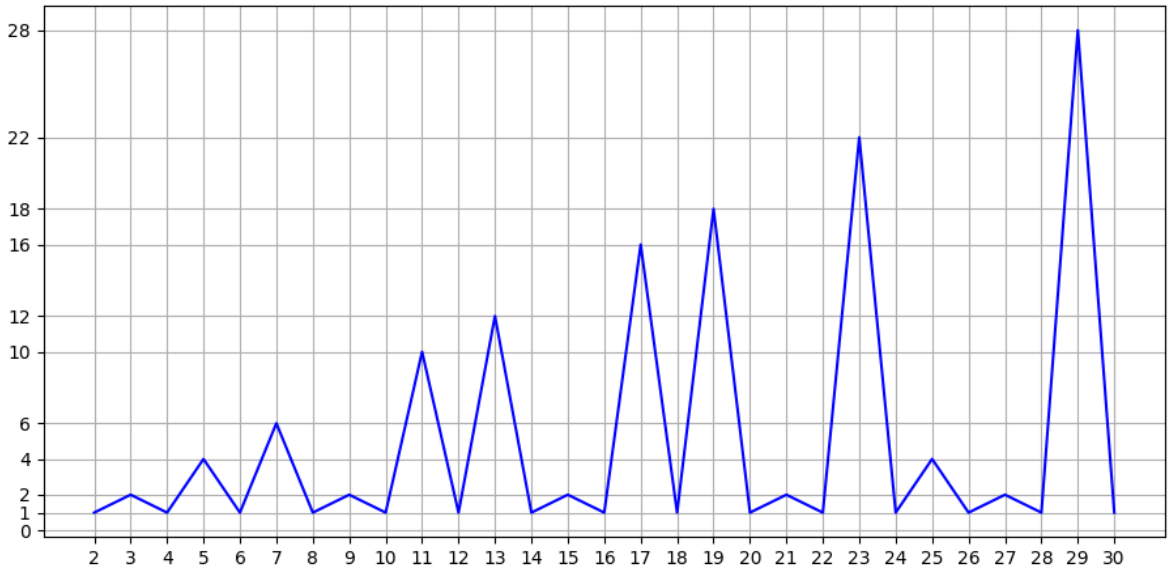

Figure 1: $T(n)$ for the first algorithm.

b) How many calls of DIVIDES does this algorithm make in the worst case? That is, what is the largest possible number of calls of the procedure DIVIDES 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 prime.

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

Solution: 1 if and only if n is even.

d) Show that there exists a function $f(n) < n$ such that if k exceeds $f(n)$, then n is prime. What is the smallest possible function $f(n)$ of the form $f(n) = n^\alpha$ (where $0 < \alpha < 1$) with this property?

Solution: Consider $f(n) = n^{0.5} = \sqrt{n}$. Assume that $n > 1$ is not prime. Let's take its smallest divisor $k > 1$. Since n is not prime, $n = k \cdot m$ for some $m > 1$ that is a divisor of n. Since k is the smallest devisor, $k \leq m$. Hence $k^2 \leq n$ and $k \leq \sqrt{n}$. Therefore, if during the execution of the algorithm k exceeds $\sqrt{n}$, n should be prime.

$f(n) = \sqrt{n}$ is the smallest possible function of the form n^α. Consider $n = p^2$ for any prime p. In this case the while loop stops when $k = p = \sqrt{n}$. Hence for such n, $f(n) \geq \sqrt{n}$.

Let f be the smallest possible function of the form $f(n) = n^\alpha$ from point d). Consider the following algorithm:

Algorithm 2 ImprovedPrimalityTest(n)

```
k ← 2
while k ≤ f(n) and DIVIDES(k, n) is false do
    k ← k + 1
if k > f(n) then
    return "n is prime"
else
    return "n is not prime"
```

e) Answer the questions from points a), b) and c) for the second algorithm. You can assume that in the while loop we do not call DIVIDES(k, n) if $k > f(n)$.

Solution:

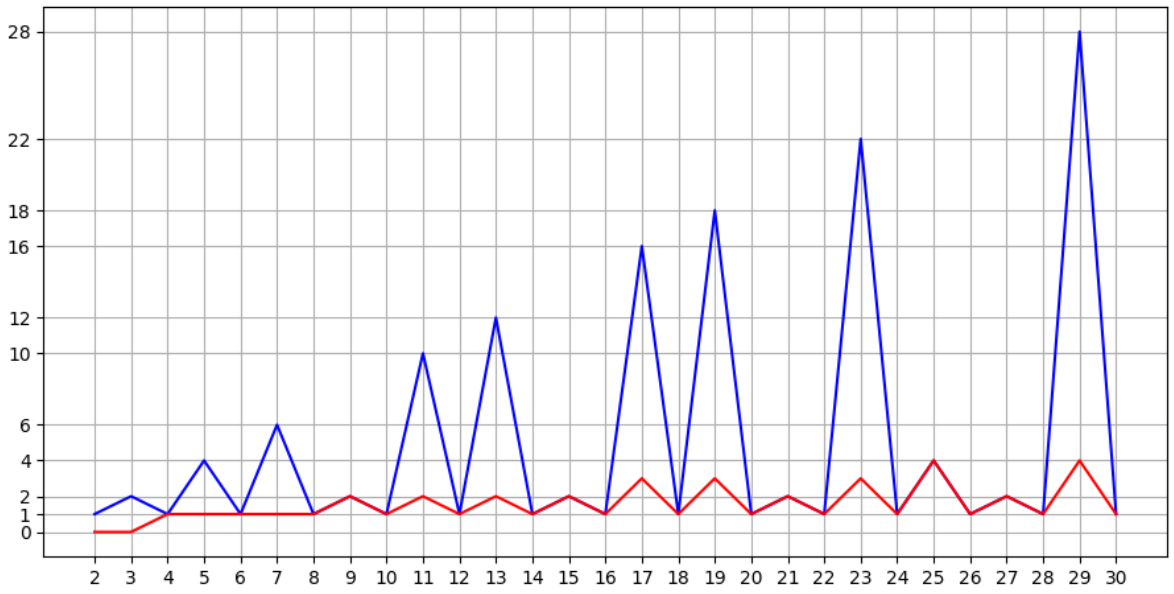

Figure 2: $T(n)$ for the first (blue) and the second (red) algorithms.

There are two worst cases: when n is prime and when n is a square of a prime number (that is, $n = p^2$ for some prime p). In this cases, number of calls of DIVIDES (i.e. $T(n)$) is $\lfloor\sqrt{n}\rfloor - 1$, where $\lfloor x\rfloor$ is the greatest integer $m \leq x$.

The best case is $T(n) = 0$, it corresponds to $n = 2$ and $n = 3$.

Alternative answer: asymptotically the best case is $T(n) = 1$, it corresponds to even $n \geq 4$.

f)* Is the running time of the first algorithm polynomial in the size of the input? That is, is there any constant integer $m > 0$ such that the (worst-case) running time of the algorithm is $\mathcal{O}(l^m)$, where l is the length of the input? What about the second algorithm?

Solution: The size of the input is $l = \lfloor\log_{10} n\rfloor + 1$ (number of digits in the decimal expression of n). The (worst-case) running time of the first algorithm is at least $n - 1$ (since we need at list $n - 1$ calls of DIVIDES for some inputs). Hence it is at least $10^{l-1} - 1$. Assume that there exists $m > 0$ such that $10^{l-1} - 1 \in \mathcal{O}(l^m)$. By applying L'Hôpital's rule m times to the fraction $x^m/(10^{x-1} - 1)$, we get that the limit of this fraction is 0. Hence by Theorem 1 from Exercise sheet 0, $10^{l-1} - 1 \notin \mathcal{O}(l^m)$, a contradiction. So the first algorithm is not polynomial in the size of the input.

Similarly, the running time of the second algorithm is at least $\sqrt{10^{l-1}} - 1 = (\sqrt{10})^{l-1} - 1$. By similar argument as for the first algorithm, $(\sqrt{10})^{l-1} - 1 \notin \mathcal{O}(l^m)$ for any constant integer $m > 0$. So the second algorithm is also not polynomial in the size of the input.

Exercise 1.2 *Induction.*

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

$$2^1 + 2^2 + \cdots + 2^n = 2^{n+1} - 2\,.$$

- **Base Case.**
  Let $n = 1$. Then:
  $$2^1 = 2^2 - 2\,.$$
- **Induction Hypothesis.**
  Assume that the property holds for some positive integer k. That is:
  $$2^1 + 2^2 + \cdots + 2^k = 2^{k+1} - 2\,.$$
- **Inductive Step.**
  We must show that the property holds for $k + 1$ summands.
  $$\begin{aligned} 2^1 + 2^2 + \cdots + 2^k + 2^{k+1} &\overset{\text{I.H.}}{=} 2^{k+1} - 2 + 2^{k+1} \\ &= 2 \cdot 2^{k+1} - 2 \\ &= 2^{(k+1)+1} - 2\,. \end{aligned}$$

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

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

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

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, $\log_7(n^8) \in \mathcal{O}(\log(n^{\sqrt{n}}))$.

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

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

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

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

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

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

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:

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

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

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

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

   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, $n \ln n \in \mathcal{O}(\ln(n!))$.

d)* Let $f : \mathbb{N} \to \mathbb{R}^+$ and $g : \mathbb{N} \to \mathbb{R}^+$. Is it always true that either $g \in \mathcal{O}(f)$ or $f \in \mathcal{O}(g)$ 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 *Subtraction algorithm* **(2 points)**.

The goal of this exercise is to design a subtraction algorithm of natural numbers that uses additions, comparisons and some simple operations with digits. Consider two natural n-digit numbers a and b. Denote by $a_{n-1} \dots a_0$ and $b_{n-1} \dots b_0$ their decimal expressions.

a) Describe an algorithm that compares a and b. That is, your algorithm should output true if $a \geq b$ and false otherwise. You may assume that you have access to an elementary operation which compares two digits with each other.

Solution: We compare digits a_k with b_k starting from $k = n-1$. If $a_k = b_k$ we proceed to the next digit until either $a_k < b_k$ or $a_k > b_k$ or there is no next digit. If we reach the case $a_k < b_k$ we output false, otherwise we output true. Note that the algorithm also outputs true if $a = b$ because in this case the if-condition is applied with $k = 0$.

Algorithm 3 Compare(a, b)

```
k ← n − 1
while a_k = b_k and k > 0 do
    k ← k − 1
if a_k < b_k then
    return false
else
    return true
```

Let $\bar{b} = \underbrace{99\dots9}_{n \text{ times}} - b$. Note that $\bar{b}$ is an n-digit number with digits $\bar{b}_i = 9 - b_i$ for $i = 0, \dots, n-1$.

b) Show that $a - b + 10^n$ can be computed using one computation of $\bar{b}$ and two additions of (at most) $(n+1)$-digit numbers.

Solution: We observe that $a - b + 10^n$ is the same as $a + \bar{b} + 1$, and the latter can be computed by two additions from a and $\bar{b}$.

c) Show that if $a \geq b$, then $a - b$ can be obtained from $a - b + 10^n$ by removing the first digit.

Solution: Since a and b are n-digit numbers we have $c = a - b \leq a < 10^n$. Therefore, $c = c_{n-1} \dots c_0$ is a non-negative number with at most n digits, and thus $a - b + 10^n = 1_n c_{n-1} \dots c_0$.

d) Using points a), b) and c), design a subtraction algorithm of natural numbers. Specifically, given two decimal representations of n-digit numbers a and b as input, your algorithm should output the decimal representation of $a - b$. Here we assume that negative numbers are represented as decimal expressions that come after the '$-$' sign.

Solution: Given two inputs a and b we apply a) to determine whether $a \geq b$. If $a \geq b$, we apply b) and c) to obtain $a - b$. Else, we utilize the identity $a - b = (-1)(b - a)$, i.e., we apply b) and c) to obtain $b - a$ and attach a '$-$' sign to the result.

Algorithm 4 Subtract(a, b)

if Compare(a, b) is true **then**
    $r \leftarrow a + \bar{b} + 1$
    return $r_{n-1} \dots r_0$
else
    $r \leftarrow b + \bar{a} + 1$
    return $-r_{n-1} \dots r_0$

e) How many elementary operations does your algorithm take in the worst case? For the addition operations, you may use the general bound from the lecture for adding $(n + 1)$-digit numbers. Elementary operations are comparing two digits, adding two digits (possibly with a carry bit), removing the leading digit, inverting a digit (i.e., computing $\bar{b}_i$) and writing the '$-$' sign before the decimal representation of the number.

Solution: For a) we require at most n comparisons, for b) we require at most $n+2(n+1)$ operations (n for the computation of $\bar{b}$ and $2(n+1)$ for the two additions of (at most) n-digit numbers), and, for c) we require one operation. In the worst case b is greater than a and we have to perform the extra operation of attaching the '$-$' sign at the end. Thus, we require at most $2n + 2(n + 1) + 2 = 4n + 4$ operations, i.e., $\mathcal{O}(n)$ operations.