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
Gleb Novikov, Tommaso d'Orsi, Ulysse Schaller, Rajai Nasser

04 October 2021

Algorithms & Data Structures — Exercise sheet 2 — HS 21

The solutions for this sheet are submitted at the beginning of the exercise class on October 11th.

Exercises that are marked by * are challenge exercises. They do not count towards bonus points. In this sheet, the only exercises that are counted towards bonus points are: 2.2, 2.4 and 2.5 (a, e, f). You can use results from previous parts without solving those parts.

Exercise 2.1 *Induction.*

1. Prove via mathematical induction that for all integers $n \geq 5$,
$$2^n > n^2 \,.$$

2. Let x be a real number. Prove via mathematical induction that for every positive integer n, we have
$$(1+x)^n = \sum_{i=0}^{n} \binom{n}{i} x^i \,,$$
where
$$\binom{n}{i} = \frac{n!}{i!(n-i)!} \,.$$
We use a standard convention $0! = 1$, so $\binom{n}{0} = \binom{n}{n} = 1$ for every positive integer n.

Hint: You can use the following fact without justification: for every $1 \leq i \leq n$,
$$\binom{n}{i} + \binom{n}{i-1} = \binom{n+1}{i} \,.$$

Asymptotic Notation

When we estimate the number of elementary operations executed by algorithms, it is often useful to ignore constant factors and instead use the following kind of asymptotic notation, also called O-Notation. We denote by $\mathbb{R}^+$ the set of all (strictly) positive real numbers and by $\mathbb{N}$ the set of all (strictly) positive integers.

Definition 1 (O-Notation)**.** Let $n_0 \in \mathbb{N}$, $N := \{n_0, n_0+1, \ldots\}$ and let $f : N \to \mathbb{R}^+$. $O(f)$ is the set of all functions $g : N \to \mathbb{R}^+$ such that there exists $C > 0$ such that for all $n \in N$, $g(n) \leq Cf(n)$.

In general, we say that $g \leq O(f)$ if Definition 1 applies after restricting the domain to *some* $N = \{n_0, n_0+1, \ldots\}$. Some sources use the notation $g = O(f)$ or $g \in O(f)$ instead.

Instead of working with this definition directly, it is often easier to use limits in the way provided by the following theorem.

Theorem 1 (Theorem 1.1 from the script). *Let $f : N \to \mathbb{R}^+$ and $g : N \to \mathbb{R}^+$.*

- *If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, then $f \leq O(g)$ and $g \not\leq O(f)$.*
- *If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = C \in \mathbb{R}^+$, then $f \leq O(g)$ and $g \leq O(f)$.*
- *If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, then $f \not\leq O(g)$ and $g \leq O(f)$.*

The theorem holds all the same if the functions are defined on $\mathbb{R}^+$ instead of N. In general, $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ is the same as $\lim_{x\to\infty} \frac{f(x)}{g(x)}$ if the second limit exists.

The following theorem can also be helpful when working with O-notation.
Theorem 2. *Let $f, g, h : \mathbb{N} \to \mathbb{R}^+$. If $f \leq O(h)$ and $g \leq O(h)$, then*

1. *For every constant $c \geq 0$, $c \cdot f \leq O(h)$.*
2. *$f + g \leq O(h)$.*

Notice that for all real numbers $a, b > 1$, $\log_a n = \log_a b \cdot \log_b n$ (where $\log_a b$ is a positive constant). Hence $\log_a n \leq O(\log_b n)$. So you don't have to write bases of logarithms in asymptotic notation, that is, you can just write $O(\log n)$.

Exercise 2.2 *Comparison of functions* **(1 point)**.

Prove or disprove the following statements:

a) $(n^2 - n + 1)^2 \leq O(n^4)$ and $n^4 \leq O((n^2 - n + 1)^2)$.

b) $\sqrt{n} \leq O(\sqrt[3]{n \log n})$.

c) $\log_{100}^2(n) \leq O(\log_2(n^{100}))$.

d) $\sum_{k=1}^{n} \left(k^2 e^k + \ln^3 k\right) \leq O(3^n)$.

e) $\sum_{k=0}^{n} 2^k \leq O(2^n)$.

Exercise 2.3 *Asymptotic growth of* $\ln(n!)$.

Recall that the factorial of a positive integer n is defined as $n! = 1 \times 2 \times \cdots \times (n-1) \times n$.

a) Show that $\ln(n!) \leq O(n \ln n)$.

Hint: *You can use the result of Exercise 1.3.c.1*

b) Show that $n \ln n \leq O(\ln(n!))$.

Hint: *You can use the result of Exercise 1.3.c.3*

Exercise 2.4 *Runtime of Iterative Algorithms* **(1 point)**.

Many algorithms are iterative in the sense that they repeat n iterations of some procedure. The number of iterations n is usually monotonically related to the size of the input, i.e., bigger inputs require more

iterations. Furthermore, it is often the case that later iterations take more time. More precisely, if $t(k)$ is the time taken by the k-th iteration, then $t(i) \leq t(j)$ for all $i \leq j$. Clearly, the total running time $T(n)$ of such an algorithm satisfies

$$T(n) = \sum_{k=1}^{n} t(k).$$

In this exercise, we are interested in analyzing the asymptotic growth of $T(n)$ in terms of that of $t(n)$. As we previously mentioned, in this exercise we always assume that the function $t : \mathbb{N} \to \mathbb{N}$ is nondecreasing.

a) Show that for arbitrary nondecreasing function $t : \mathbb{N} \to \mathbb{N}$, we always have $T(n) \leq O(n \cdot t(n))$.

b) Assume that $t(n)$ grows polynomially, i.e., there exist real numbers $\beta > 0$ and $C \geq 1$ such that for all $n \in \mathbb{N}$, $\frac{1}{C} \cdot n^{\beta} \leq t(n) \leq C \cdot n^{\beta}$. Show that $n \cdot t(n) \leq O(T(n))$.

Hint: *Use the fact that for every $n \geq 2$, we have $\sum_{k=\lceil \frac{n}{2} \rceil}^{n} k^{\beta} \geq \frac{n}{2} \cdot \left(\frac{n}{2}\right)^{\beta}$, where $\lceil x \rceil$ denotes the smallest integer ℓ satisfying $\ell \geq x$.*

c) Show that for arbitrary nondecreasing function $t : \mathbb{N} \to \mathbb{N}$, we always have $t(n) \leq O(T(n))$.

d) Assume that $t(n)$ grows exponentially, i.e., there exist real numbers $\alpha > 1$ and $C \geq 1$ such that for all $n \in \mathbb{N}$, $\frac{1}{C} \cdot \alpha^{n} \leq t(n) \leq C \cdot \alpha^{n}$. Show that $T(n) \leq O(t(n))$.

Hint: *Use Exercise 1.2.*

Remark. Together, the results of a) and b) imply that if $t(n)$ grows polynomially, then $T(n)$ has the same asymptotic growth as $n \cdot t(n)$. It is possible to show a similar result if $t(n)$ is bounded or grows logarithmically. The results of c) and d) imply that if $t(n)$ grows exponentially, then $T(n)$ has the same asymptotic growth as $t(n)$.

Exercise 2.5 *The Euclidean Algorithm For Finding the Greatest Common Divisor* **(1 point)**.

We say that $a \in \mathbb{N}$ divides $b \in \mathbb{N}$ if there exists $q \in \mathbb{N}$ such that $b = aq$. The greatest common divisor $\gcd(n, m)$ of $n \in \mathbb{N}$ and $m \in \mathbb{N}$ is the greatest integer $d \in \mathbb{N}$ that divides both n and m. In this exercise, we are interested in computing $\gcd(n, m)$ as efficiently as possible.

Note that since $\gcd(n, m) = \gcd(m, n)$, we may assume without loss of generality that $n \geq m$. So in this exercise we always assume $n \geq m$.

The simplest algorithm that we can think of is the following procedure:

- Check whether $d = m$ is a common divisor for n and m. If yes, output d. Otherwise, go to the next step.
- Check whether $d = m - 1$ is a common divisor for n and m. If yes, output d. Otherwise, go to the next step.
- Then, we similarly check $m - 2, m - 3, \ldots, 1$.

Let $T_{\text{simple}}(n, m)$ be the number of iterations (steps) that are taken by the above simple algorithm to find the greatest common divisor.

a) For fixed $n \geq 1$, determine $\min_{1 \leq m \leq n} T_{\text{simple}}(n, m)$ and $\max_{1 \leq m \leq n} T_{\text{simple}}(n, m)$?

Hint: *You may use the following fact without proof: For every positive integer* n, $\gcd(n, n-1) = 1$.

In the remaining of this exercise, we will describe an algorithm that can compute $\gcd(n, m)$ much more efficiently. This algorithm is based on applying divisions with remainders. For every $a, b \in \mathbb{N}$, there exist unique nonnegative integers q, r, such that $b = aq + r$ and $0 \leq r < b$. The number q (respectively, r) is called *the quotient (respectively, the remainder) of the division of* b *by* a.

Let q and r be the quotient and remainder of the division of n by m, i.e., $n = qm + r$ and $0 \leq r < m$.

b)* Show that if $r = 0$, then $\gcd(n, m) = m$.

c)* Show that if $r > 0$, then $\gcd(n, m) = \gcd(m, r)$.

The above two properties suggest the following algorithm to compute $\gcd(n, m)$:

- Define $n_0 = n$ and $n_1 = m$.
- For every $i \geq 1$, as long as, $n_i > 0$, we perform the division with remainder of n_{i-1} by n_i, i.e., we find the unique q_i, r_i that satisfy $n_{i-1} = q_i n_i + r_i$ and $0 \leq r_i < n_i$, and then we define $n_{i+1} = r_i$.
- When we reach i for which $n_i = 0$, we stop and output n_{i-1}.

This is called the Euclidean algorithm for computing the greatest common divisor.

d)* Show that the above algorithm correctly computes the greatest common divisor of n and m.

e) Show that for every $i \geq 2$, we have $n_i < \frac{n_{i-2}}{2}$.

Let $T_E(n, m)$ be the number of iterations that are taken by the Euclidean algorithm to find the greatest common divisor of n and m (where $n \geq m$).

f) Show that $T_E(n, m) \leq O(\log(n))$.

g)* Compute the greatest common divisor of 139472305678615 and 273426584985 using the Euclidean algorithm. If you had used the "naive simple algorithm" that we mentioned at the beginning of the exercise, would you have been able to compute the GCD of these two numbers using only pen and paper?