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
François Hublet, Goran Zuzic, Tommaso d'Orsi, Jingqiu Ding

3 October 2022

Algorithms & Data Structures — Exercise sheet 2 — HS 22

The solutions for this sheet are submitted at the beginning of the exercise class on 10 October 2022.

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

You can use results from previous parts without solving those parts.

Exercise 2.1 *Induction.*

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

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

Solution:

- **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 every positive integer n.

(b) 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 $1 \leq i \leq n$,

$$\binom{n}{i} + \binom{n}{i-1} = \binom{n+1}{i} \,.$$

Solution:

We will use the identity from the hint to show (via mathematical induction) that

$$(1+x)^n = \sum_{i=0}^{n} \binom{n}{i} x^i \,.$$

- **Base Case.**
  Let $n = 1$. Then $(1+x)^1 = \binom{1}{0}x^0 + \binom{1}{1}x^1 = \sum_{i=0}^{n} \binom{n}{i} x^i$.
- **Induction Hypothesis.**
  Assume that the property holds for some positive integer k. That is,

$$(1+x)^k = \sum_{i=0}^{k} \binom{k}{i} x^i.$$

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

$$\begin{aligned}
(1+x)^{k+1} &= (1+x)(1+x)^k \\
&\overset{I.H.}{=} (1+x) \sum_{i=0}^{k} \binom{k}{i} x^i \\
&= \Big(\sum_{i=0}^{k} \binom{k}{i} x^i \Big) + \Big(\sum_{i=0}^{k} \binom{k}{i} x^{i+1} \Big) \\
&= \Big(\sum_{i=0}^{k} \binom{k}{i} x^i \Big) + \Big(\sum_{i=1}^{k+1} \binom{k}{i-1} x^i \Big) \\
&= \binom{k}{0} x^0 + \sum_{i=1}^{k} \Big(\binom{k}{i} x^i + \binom{k}{i-1} x^i \Big) + \binom{k}{k} x^{k+1} \\
&= \binom{k+1}{0} x^0 + \sum_{i=1}^{k} \binom{k+1}{i} x^i + \binom{k+1}{k+1} x^{k+1} = \sum_{i=0}^{k+1} \binom{k+1}{i} x^i.
\end{aligned}$$

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

Exercise 2.2 *Growth of Fibonacci numbers* **(1 point)**.

There are a lot of neat properties of the Fibonacci numbers that can be proved by induction. Recall that the Fibonacci numbers are defined by $f_0 = 0, f_1 = 1$ and the recursion relation $f_{n+1} = f_n + f_{n-1}$ for all $n \geq 1$. For example, $f_2 = 1, f_5 = 5, f_{10} = 55, f_{15} = 610$.

(a) Prove that $f_{n+1} \leq 1.75^n$ for $n \geq 0$.

Solution:

- **Base Case.** We prove that the inequality holds for $n = 0$ and $n = 1$.
  For $n = 0$: $f_1 = 1 \leq 1.75^0 = 1$, which is true. For $n = 1$: $f_2 = 1 \leq 1.75^1 = 1.75$, which is true.
- **Induction Hypothesis.** We assume that it is true for $n = k$ and $n = k + 1$, i.e.,

$$f_{k+1} \leq 1.75^k$$
$$f_{k+2} \leq 1.75^{k+1}$$

- **Inductive Step.** We must show that the property holds for $n = k + 2$, $k \geq 0$. We have:

$$\begin{aligned} f_{k+3} &= f_{k+2} + f_{k+1} \\ &\leq 1.75^{k+1} + 1.75^k \\ &= 1.75^k(1.75 + 1) \\ &= 1.75^k \cdot 2.75 \\ &\leq 1.75^k \cdot 3.0625 \\ &= 1.75^k \cdot 1.75^2 \\ &= 1.75^{k+2} \end{aligned}$$

  By the principle of mathematical induction, this is true for every integer $n \geq 0$.

(b) Prove that $f_n \geq \frac{1}{3} \cdot 1.5^n$ for $n \geq 1$.

Solution:

- **Base Case.** We prove that the inequality holds for $n = 1$ and $n = 2$.
  For $n = 1$: $f_1 = 1 \geq 0.5 = \frac{1}{3} \cdot 1.5$, which is true.
  For $n = 2$: $f_2 = 1 \geq 0.75 = \frac{1}{3} \cdot 1.5^2$, which is true.
- **Induction Hypothesis.** We assume that it is true for $n = k$ and $n = k + 1$, i.e.,

$$f_k \geq \frac{1}{3} 1.5^k$$
$$f_{k+1} \geq \frac{1}{3} 1.5^{k+1}$$

- **Inductive Step.** We must show that the property holds for $n = k + 2$, $k \geq 1$. We have:

$$\begin{aligned} f_{k+2} &= f_{k+1} + f_k \\ &\geq \frac{1}{3}1.5^{k+1} + \frac{1}{3}1.5^k \\ &= \frac{1}{3}1.5^k \cdot (1.5 + 1) \\ &= \frac{1}{3}1.5^k \cdot 2.5 \\ &\geq \frac{1}{3}1.5^k \cdot 2.25 \\ &= \frac{1}{3}1.5^k \cdot 1.5^2 \\ &= \frac{1}{3}1.5^{k+2} \end{aligned}$$

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

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 2.3 *O-notation quiz.*

(a) Prove or disprove the following statements. Justify your answer.

(1) $n^{\frac{2n+3}{n+1}} = O(n^2)$

Solution:

True by Theorem 1, since

$$\lim_{n\to\infty} \frac{n^{\frac{2n+3}{n+1}}}{n^2} = \lim_{n\to\infty} n^{\frac{2n+3}{n+1}-2} = \lim_{n\to\infty} n^{\frac{2n+3-2n-2}{n+1}} = \lim_{n\to\infty} n^{\frac{1}{n+1}} = \lim_{n\to\infty} e^{\frac{\log n}{n+1}} = 1.$$

(2) $e^{1.2n} = O(e^n)$

Solution:

False by Theorem 1, since

$$\lim_{n\to\infty} \frac{e^{1.2n}}{e^n} = \lim_{n\to\infty} e^{1.2n-n} = \lim_{n\to\infty} e^{0.2n} = \infty.$$

(3) $\log(n^4 + n^3 + n^2) = O(\log(n^3 + n^2 + n))$

Solution:

True by Theorem 1, since

$$\begin{aligned}\lim_{n\to\infty} \frac{\log(n^3+n^2+n)}{\log(n^4+n^3+n^2)} &\overset{\text{L'Hôpital}}{=} \lim_{n\to\infty} \frac{\frac{3n^2+2n+1}{n^3+n^2+n}}{\frac{4n^3+3n^2+2n}{n^4+n^3+n^2}} = \lim_{n\to\infty} \frac{(3n^2+2n+1)(n^4+n^3+n^2)}{(n^3+n^2+n)(4n^3+3n^2+2n)} \\ &= \lim_{n\to\infty} \frac{3n^6+P(n)}{4n^6+Q(n)} = \lim_{n\to\infty} \left(\frac{3n^6}{4n^6+Q(n)} + \frac{P(n)}{4n^6+Q(n)}\right) \\ &= \frac{3}{4} + \lim_{n\to\infty} \frac{P(n)}{4n^6+Q(n)}\end{aligned}$$

where $\deg P = \deg Q = 5$. For all α and $k \leq 5$, $\frac{\alpha n^k}{4n^6+Q(n)} \leq \frac{\alpha n^k}{4n^6} = \frac{\alpha}{4} n^{k-6} \to 0$, hence $\frac{P(n)}{4n^6+Q(n)} \to 0$ by Theorem 2. Therefore,

$$\lim_{n\to\infty} \frac{\log(n^3+n^2+n)}{\log(n^4+n^3+n^2)} = \frac{3}{4}.$$

(b) Find f and g as in Theorem 1 such that $f = O(g)$, but the limit $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ does not exist. This proves that the first point of Theorem 1 provides a sufficient, but not a necessary condition for $f = O(g)$.

Solution:

Let $f(n) = 1 + (-1)^n$ and $g(n) = 1$. We have $\frac{f(n)}{g(n)} = \frac{1+(-1)^n}{1} = 1 + (-1)^n$, which has no limit when $n \to \infty$.

Exercise 2.4 *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 fact that* $n! \leq n^n$ *for* $n \geq 1$ *without proof.*

Solution:

Solution: From the hint, we have $n! \leq n^n$, which implies that $\ln(n!) \leq n \ln n$ and thus $\ln(n!) \leq O(n \ln n)$.

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

Hint: *You can use the fact that* $\left(\frac{n}{2}\right)^{\frac{n}{2}} \leq n!$ *for* $n \geq 1$ *without proof.*

Solution:

From the hint, we have $n! \geq \left(\frac{n}{2}\right)^{n/2}$. Now by the monotonicity of the logarithm we have

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

so $n \ln n \leq 2\ln(n!) + 2\ln 2$. By Theorem 1, $n \ln n \leq O(\ln(n!))$.

Exercise 2.5 *Triplet Search* **(2 points)**.

Given an array of n integers, and an integer t, design an algorithm that checks if there exists three (not necessarily different) elements of the array a, b, c such that $a + b + c = t$.

(a) Design a simple $O(n^3)$ algorithm.

Solution:

The algorithm can simply check all n^3 triples $(A[i], A[j], A[k])$ of elements in A by using three nested loops with indices (i, j, k) that iterate over all integers in $[1, n]$. For each such triple, we check whether $A[i] + A[j] + A[k] = t$ and report success ("YES") if we ever find a satisfying triple. Otherwise, we return failure ("NO"). The pseudocode is given below.

Algorithm 1 Input: an array A of n integers, and an integer t.

```
for i = 1, 2, ..., n do
    for j = 1, 2, ..., n do
        for k = 1, 2, ..., n do
            if A[i] + A[j] + A[k] = t then
                return "YES" and exit
return "No"
```

The algorithm clearly works by checking all n^3 possibilities, hence it is trivially correct and its runtime is clearly $O(n^3)$.

(b) Suppose that elements of the array are integers in the range $[1, 100n]$, and that $t \leq 300n$. Design a better algorithm with runtime $O(n^2)$ to solve the same problem, assuming the constraints.

Hint: *You can use a separate array with $O(n)$ entries to help you. Start with the "naive" algorithm from (a) and try removing one of the loops with a smart lookup using the new array.*

Hint: *$a + b + c = t$ implies that $a = t - b - c$.*

Solution:

We use a separate $O(n)$-sized array $B[1 \dots 100n]$, which is originally initialized to value 0. First, in $O(n)$ time, we mark every entry that appears in A with a value of 1 in B: more precisely, we set $B[A[i]] \leftarrow 1$ for all $i \in \{1, \dots, n\}$. Then, we use two nested loops i, j to iterate over all possible n^2 pairs of elements from the array A. In each iteration, we check whether there exists an element $A[k]$ such that $A[i] + A[j] + A[k] = t$. In other words, we check if $t - A[i] - A[j]$ is in the array, which can be accomplished in $O(1)$ time by checking if $t - A[i] - A[j]$ fits within the range $[1, 100n]$ and then if $B[t - A[i] - A[j]] = 1$; if both of these are true, we output "Yes". At the end of the algorithm, if we didn't output yet, we output "No". A pseudocode equivalent to the above algorithm is given below.

Algorithm 2 Input: an array A of n integers, and an integer t.

```
B[1...100n] ← (0, 0, ..., 0)
for i = 1, 2, ..., n do
    B[A[i]] ← 1
for i = 1, 2, ..., n do
    for j = 1, 2, ..., n do
        if 1 ≤ t − A[i] − A[j] ≤ 100n AND B[t − A[i] − A[j]] = 1 then
            return "YES" and exit
return "No"
```

It is clear from the algorithm that the runtime is $O(n^2)$ as the initialization of B took $O(n)$, and then each iteration over the n^2 pairs took $O(1)$ time, for a total of $O(n) + n^2 \cdot O(1) = O(n^2)$ time.

Finally, we argue correctness. If there are no satisfying triplets in the input, the algorithm clearly outputs "NO" as whenever the algorithm outputs "YES" it finds a satisfying triplet in the input (namely, $A[i]$, $A[j]$, and some $A[k] = t - A[i] - A[j]$ which is guaranteed to exist by algorithm design). If there exists a satisfying triplet (say) i^*, j^*, k^*, then during the nested loop iteration at some point we will have $i = i^*$ and $j = j^*$. At that point, $t - A[i^*] - A[j^*]$ is within $[1, 100n]$ and $t - A[i^*] - A[j^*]$ exists in the array A (namely, as $A[k^*]$), hence it would be found. This concludes the correctness proof.

(c)* Suppose now that, unlike in (b), we don't have a bound on the size of the integers elements of A nor on t (but we can still perform arithmetic operations on them in $O(1)$ time). However, they are given in increasing order in A, i.e., $A[1] \leq A[2] \leq \dots A[n]$. Design an $O(n^2)$ algorithm to solve the same problem, assuming the constraints.

Hint: *Exploit the increasing order of A to leverage the computation done in the previous step to help you in the next one.*

Solution:

We use two nested loops i, j to iterate over all possible n^2 pairs of elements from the array A, in order. However, in parallel to j, we will also store a value k satisfying the invariant that k

is the smallest index in $[1, n]$ such that $A[i] + A[j] + A[k] \geq t$. Each time we increment j we need to keep decreasing k until the invariant would start failing. At that point, we check whether $A[i]+A[j]+A[k] = t$ and report we found a satisfying triple if this condition is ever true. Otherwise, we report failure at the end. The pseudocode implementing this algorithm follows.

Algorithm 3 Input: an array A of n integers, and an integer t.

```
for i = 1, 2, ..., n do
    k ← n
    for j = 1, 2, ..., n do
        while k ≥ 1 and A[i] + A[j] + A[k − 1] ≥ t do    ▷ Can we decrease k and keep the invariant?
            k ← k − 1
        if A[i] + A[j] + A[k] = t then
            return "YES" and exit
return "No"
```

The above invariant is kept true (after time while loop terminates) since each time we increment j, the value of $A[i] + A[j] + A[k]$ increases, hence we need to decrease k (zero or possibly large) number of times until decreasing it any further would make $A[i] + A[j] + A[k] < t$. The while loop directly implements this, hence the invariant is clearly satisfied.

For correctness, if there are no satisfying triplets, the algorithm cannot find any (since the algorithm outputs "YES" only upon explicitly finding a satisfying triplet). On the other hand, if there exists a satisfying triplet i^*, j^*, k^* such that $A[i^*] + A[j^*] + A[k^*] = t$, then at some point we will have $i = i^*, j = j^*$. Then, by the invariant, k will be the largest index with $A[i^*] + A[j^*] + A[k] \geq t$. Since there exists a value k^* such that $A[i^*] + A[j^*] + A[k^*] = t$ and A is sorted, the largest value of k must satisfy also $A[i^*] + A[j^*] + A[k] = t$. This proves correctness of the algorithm.

For runtime, there are at most n iterations of the i loop. In each such iteration of i, k can be decremented a total of n times, hence the total number of decrements is at most n (for each fixed value i). Hence, the total number of times the while loop iterates (in the entire program) is $O(n^2)$. All other operations beside the while-loop are also $O(n^2)$, hence the runtime is $O(n^2)$.