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

11 October 2021

Algorithms & Data Structures — Exercise sheet 3 — HS 21

The solutions for this sheet are submitted at the beginning of the exercise class on October 18th.

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: 3.3.(a-c) and 3.4.(a-d). You can use results from previous parts without solving those parts.

Exercise 3.1 *Some properties of O-Notation.*

Let $f : \mathbb{R}^+ \to \mathbb{R}^+$ and $g : \mathbb{R}^+ \to \mathbb{R}^+$.

a) Show that if $f \leq O(g)$, then $f^2 \leq O(g^2)$. You can assume that $\lim_{x\to\infty} \frac{f(x)}{g(x)} = C \in \mathbb{R}_0^+$.

b) Give an example where $f \leq O(g)$, but $2^f \not\leq O(2^g)$.

Exercise 3.2 *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.

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.

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

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 $k \in \mathbb{N}_0$.

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 $k \in \mathbb{N}$ such that $n = 2k + 1$.*

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

Exercise 3.3 *Counting Operations in Loops* **(1 Point)**.

For the following code fragments count how many times the function f is called. Report the number of calls as nested sum, and then simplify your expression in O-notation (as tight and simplified as possible) and prove your result.

More precisely, let $T(n)$ be the number of calls of function f in the snippet. Find as simple as possible function $g(n)$ such that $T(n) \leq O(g(n))$ and $g(n) \leq O(T(n))$ and prove these bounds.

For example, in the snippet

Algorithm 1

```
for k = 1, ..., n do
    f()
```

the function f is called $\sum_{k=1}^{n} 1 = n$ times. It is clear that in this example $T(n) \leq O(n)$. Moreover, this bound is tight, that is, $n \leq O(T(n))$.

a) Consider the snippet:

Algorithm 2

```
for j = 1, ..., n do
    for k = 1, ..., n do
        for m = 1, ..., n do
            f()
```

How many times is the function f called?

b) Consider the snippet:

Algorithm 3

```
for j = 1, ..., n do
    k ← min(j, 100)
    for l = 1, ..., k do
        f()
```

How many times is the function f called?

c) Consider the snippet:

Algorithm 4

```
for j = 1, ..., n do
    if j^2 ≤ n then
        for k = j, ..., n do
            f()
            f()
            f()
```

How many times is the function f called?

Hint: *You may use the following fact without proof: For every $n \geq 2$, we have $\lfloor \sqrt{n} \rfloor \geq \frac{\sqrt{n}}{2}$ and $n - \lfloor \sqrt{n} \rfloor + 1 \geq \frac{n}{2}$, where $\lfloor x \rfloor$ is the largest integer satisfying $\lfloor x \rfloor \leq x$.*

d)* Consider the snippet:

Algorithm 5

```
for j = 1, ..., n do
    k ← 1
    l ← 0
    while k ≤ j do
        for m = 0, ..., l do
            f()
        k ← 13 · k
        l ← l + 1
```

How many times is the function f called?

e)* Consider the snippet:

Algorithm 6

```
for j = 1, ..., n do
    for k = 1, ..., j do
        for ℓ = 1, ..., k do
            for m = ℓ, ..., n do
                for o = 1, ..., 100 do
                    f()
```

How many times is the function f called?

Exercise 3.4 *Investing in the stock market* **(2 Points)**.

You have 100 CHF and you are considering investing it in the stock market. You heard from your friends that a particular stock is promising but you are not sure. You decided to analyze the performance of this stock during the recent past.

You have a friend that had invested in this stock for n consecutive days in the recent past. You asked your friend about how much money she invested in this stock and how much her total investment worth was progressing every day. You gathered this information in two arrays $A = (A_1, \ldots, A_n)$ and $I = (I_1, \ldots, I_n)$. Here, A_i represents the additional amount of money that your friend invested on the i-th day. More precisely, if your friend bought on the i-th day then A_i would be positive, and if she sold on the i-th day, then A_i would be negative. The value of I_i is the total worth of her investment in the stock on the i-th day. Here is an illustrating example with $n = 4$:

i	1	2	3	4
A_i	150	150	200	-100
I_i	150	350	500	400

In the above example, your friend had invested for 4 days. She invested $A_1 = 150$ CHF on the first day. Since she had not invested anything prior to that day, we must have $I_1 = A_1 = 150$ CHF. On

the second day, she invested an additional $A_2 = 150$ CHF and the total worth of her investment was $I_2 = 350$ CHF. This means that right before buying the additional 150 CHF, her total investment in the stock was worth $350 - 150 = 200$ CHF, which is an indication of an increase in the price of the stock from the first to the second day. On the last day of investment ($i = 4$), she sold 100 CHF worth of her investment, and the remaining total worth of her investment after the selling operation was 400 CHF.

a) Let $1 \leq i \leq n-1$. Suppose that you had invested 1 CHF on the i-th day and sold the entire investment on the $(i+1)$-th day. Show that you would have got $\dfrac{I_{i+1} - A_{i+1}}{I_i}$ CHF in return.

b) Let $1 \leq i \leq j \leq n$. Suppose that you had invested 100 CHF on the i-th day and sold the entire investment on the j-th day. Show that your profit is equal to

$$100 \cdot \prod_{k=i}^{j-1} \frac{I_{k+1} - A_{k+1}}{I_k} - 100.$$

Note that in the above equation, we adopt the convention that if $i = j$, then $\displaystyle\prod_{k=i}^{j-1} \frac{I_{k+1} - A_{k+1}}{I_k} = 1$.

You are interested in finding the maximum profit that you could have made in a single buy-sell operation by investing 100 CHF. Here, you would buy 100 CHF worth of the stock on some day i where $1 \leq i \leq n$ and then sell the entire investment another day j where $i \leq j \leq n$.

We first assume that all $A_1, \ldots, A_n$ and $I_1, \ldots, I_n$ are positive, and that $I_k > A_k$ for every k.

c) Describe how you can use the maximum subarray-sum algorithm that you learned in class in order to devise an algorithm that computes the maximum profit in $O(n)$ time. You can assume that arithmetic operations (such as addition, subtraction, multiplication and division) as well as logarithms and exponentials are elementary. This means that the computation of $\log$ and $\exp$ take one unit of time each.

Hint: *You can use the fact that the logarithm is a strictly increasing function that turns products into sums.*

d) Now assume that the logarithm and exponential operations are expensive so that we would like to avoid using them. Explain how you can modify the maximum subarray sum algorithm in order to solve the problem using only elementary arithmetic operations such as addition, subtraction, multiplication and division. The running time of the algorithm should remain in $O(n)$.

e)* Explain why your algorithm in part d) is correct.

f)* Now we consider the general case where $A_1, \ldots, A_n$ and $I_1, \ldots, I_n$ can be either positive or negative[1]. Describe an algorithm that computes the maximum profit that you could have made in a single buy-sell operation by investing 100 CHF. Does your algorithm run in linear time $O(n)$?

Exercise 3.5* *Maximum-Submatrix-Sum.*

Provide an $\mathcal{O}(n^3)$ time algorithm which given a matrix $M \in \mathbb{Z}^{n \times n}$ outputs its maximal submatrix sum S. That is, if M has some non-negative entries,

$$S = \max_{\substack{1 \leq a \leq b \leq n \\ 1 \leq c \leq d \leq n}} \sum_{i=a}^{b} \sum_{j=c}^{d} M_{ij},$$

[1]If you are wondering how it is possible that the numbers in $I_1, \ldots, I_n$ can be negative, check the oil price crash of 2020.

and if all entries of M are negative, $S = 0$.

Justify your answer, i.e. prove that the asymptotic runtime of your algorithm is $\mathcal{O}(n^3)$.

Hint: *You may want to start by considering the cumulative column sums*

$$C_{ij} = \sum_{k=1}^{i} M_{kj}.$$

How can you compute all C_{ij} efficiently? After you have computed C_{ij}, how you can use this to find S?