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

05. October 2020

Algorithms & Data Structures — Exercise sheet 3 — 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 October 12th.

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

Exercise 3.1 *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 $\mathcal{O}$-notation (as tight and simplified as possible) and prove your result. For example, in the code fragment

Algorithm 1

```
for k = 1, ..., 100 do
    f()
```

the function f is called $\sum_{k=1}^{100} 1 = 100$ times, so the amount of calls is in $\mathcal{O}(1)$.

Hint: Note that you are required to prove two parts: that the $\mathcal{O}$-expression is correct, and that it is tight. This corresponds to an upper and a lower bound, respectively.

a) Consider the snippet:

Algorithm 2

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

b) Consider the snippet:

Algorithm 3

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

c) Consider the snippet:

Algorithm 4

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

d) Consider the snippet:

Algorithm 5

```
for j = 1, ..., n do
    k ← 1
    while k ≤ j do
        f()
        k ← 42 · k
```

*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
                f()
```

Exercise 3.2 *Solving Recurrences* **(1 Point)**.

In this exercise, we describe a technique that can be used to solve recurrences, i.e. this allows to derive a closed form formula from a recurrence relation. Consider for example the recurrence relation

$$T(n) \leq 2T(n-1) + 1, \quad \forall n \geq 1. \tag{1}$$

Given $T(0) = 3$, we want to find an upper bound for $T(n)$ that depends only on n (and *not* on $T(n-1)$). The idea is to repeatedly apply inequality (1) to get upper bounds in terms of $T(n-1)$, then $T(n-2)$, and so on, at each step getting closer to $T(0)$ (which is known). In this case, expanding the recurrence relation a few times yields

$$\begin{aligned} T(n) &\leq 2T(n-1) + 1 \\ &\leq 2(2T(n-2)+1) + 1 = 4T(n-2) + 3 \\ &\leq 4(2T(n-3)+1) + 3 = 8T(n-3) + 7 \\ &\leq 8(2T(n-4)+1) + 7 = 16T(n-4) + 15 \\ &\vdots \end{aligned}$$

We see an emerging pattern of the form

$$T(n) \leq 2^k T(n-k) + 2^k - 1. \tag{2}$$

Plugging $k = n$ in (2), we get the conjecture

$$T(n) \leq 2^n T(0) + 2^n - 1 = 4 \cdot 2^n - 1. \tag{3}$$

Now that we have a guess, we can then use the base case $T(0) = 3$ together with the recurrence relation (1) to actually prove (3) by induction.

a) Apply the same technique to find closed form formula for the following recurrence relation, and prove by induction that your claimed formula is correct:

$$T(0) = 3, \quad T(n) = 3T(n-1) - 2 \quad \forall n \geq 1\,.$$

b) Let

$$T(1) = 1, \quad T(n) \leq 4T(n/2) + 3\log_2 n \quad \forall n = 2^m, m \geq 1\,.$$

Apply the technique described above to prove that $T(n) \leq \mathcal{O}(n^2 \log n)$ (assuming $n = 2^m, m \geq 1$).

Hint: *Use the fact that* $\log_2(n/2^k) \leq \log_2 n$ *for all* $k \in \mathbb{N}$ *to simplify the formulas when you expand the recurrence relation. Your proof should use induction on* m.

Exercise 3.3 *Maximum-Subarray-Difference* **(1 Point)**.

Consider the following problem: Given an array $A \in \mathbb{Z}^n$ compute its maximum subarray difference, i.e., compute

$$\triangle^* = \max_{1 \leq a \leq b < c \leq n} \sum_{i=a}^{b} A_i - \sum_{j=b+1}^{c} A_j. \tag{4}$$

a) Provide an $\mathcal{O}(n)$ algorithm.

b) Justify your answer:

i) Prove the correctness of your algorithm.

ii) Prove that the asymptotic runtime of your algorithm is $\mathcal{O}(n)$.

Exercise 3.4* *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,

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

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?