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()
```

Solution: f is called

$$\sum_{j=1}^{n}\sum_{k=1}^{j} 1 = \sum_{j=1}^{n} j \leq \sum_{j=1}^{n} n \leq n^2 \leq \mathcal{O}(n^2)$$

times. Notice that

$$\sum_{j=1}^{n}\sum_{k=1}^{j} 1 = \sum_{j=1}^{n} j \geq \sum_{j=\lceil n/2 \rceil}^{n} n/2 \geq n^2/4,$$

so actually we have

$$\sum_{j=1}^{n}\sum_{k=1}^{j} 1 = \Theta(n^2).$$

b) Consider the snippet:

Algorithm 3

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

Solution: f is called

$$\sum_{j=1}^{n}\sum_{l=1}^{100}\sum_{k=j}^{n} 3 = \sum_{j=1}^{n} 100 \cdot (n-j+1) \cdot 3 \leq 300 \sum_{j=1}^{n} n \leq 300n^2 \leq \mathcal{O}(n^2)$$

times. Notice that

$$\sum_{j=1}^{n}\sum_{l=1}^{100}\sum_{k=j}^{n} 3 \geq \sum_{j=1}^{n}(n-j+1) \geq \sum_{j=1}^{\lceil n/2 \rceil}(n-j+1) \geq \sum_{j=1}^{\lceil n/2 \rceil} n/2 \geq n^2/4,$$

so actually we have

$$\sum_{j=1}^{n}\sum_{l=1}^{100}\sum_{k=j}^{n} 3 = \Theta(n^2).$$

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()
```

Solution: f is called

$$\sum_{k=1}^{n} 1 + \sum_{j=1}^{n}\sum_{k=j}^{n}(1 + \sum_{l=1}^{j}\sum_{m=1}^{j} 1) = n + \sum_{j=1}^{n}\sum_{k=j}^{n}(1+j^2) = n + \sum_{j=1}^{n}(n-j+1)(1+j^2)$$
$$\leq n + \sum_{j=1}^{n} n(n^2+1) \leq n + n^4 + n^2 \leq \mathcal{O}(n^4)$$

times. Notice that

$$\sum_{k=1}^{n} 1 + \sum_{j=1}^{n}\sum_{k=j}^{n}(1 + \sum_{l=1}^{j}\sum_{m=1}^{j} 1) \geq \sum_{j=1}^{n}(n-j+1)j^2 \geq \sum_{j=\lceil n/4\rceil}^{\lceil 3n/4\rceil}(n-j+1)j^2$$
$$\geq \sum_{j=\lceil n/4\rceil}^{\lceil 3n/4\rceil} n/4 \cdot (n/4)^2 \geq n^4/256,$$

so actually we have

$$\sum_{k=1}^{n} 1 + \sum_{j=1}^{n}\sum_{k=j}^{n}(1 + \sum_{l=1}^{j}\sum_{m=1}^{j} 1) = \Theta(n^4).$$

d) Consider the snippet:

Algorithm 5

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

Solution: f is called

$$\sum_{j=1}^{n}\sum_{l=0}^{\lfloor \log_{42} j\rfloor} 1 = \sum_{j=1}^{n}(\lfloor \log_{42} j\rfloor + 1) \leq n\log_{42} n + n \leq \mathcal{O}(n\log n)$$

times. This can be seen by defining $l = \log_{42} k$. Next we use that for $n \geq 42^2$, we have $\log_{42}(n/2) = \log_{42}(n) - \log_{42}(2) \geq \log_{42} n - 1 \geq (\log_{42} n)/2$, where the last step follows from $\log_{42} n \geq 2$. Therefore,

$$\sum_{j=1}^{n}\sum_{l=0}^{\lfloor \log_{42} j\rfloor} 1 = \sum_{j=1}^{n}(\lfloor \log_{42} j\rfloor + 1) \geq \sum_{j=\lceil n/2\rceil}^{n} \log_{42}(n/2) \geq \frac{n\log_{42}(n/2)}{2} \geq \frac{n\log_{42} n}{4} = \frac{n\log n}{4\log 42},$$

so actually we have

$$\sum_{j=1}^{n}\sum_{l=0}^{\lfloor \log_{42} j\rfloor} 1 = \Theta(n\log n).$$

*e) Consider the snippet:

Algorithm 6

for $j = 1, \ldots, n$ **do**
  for $k = 1, \ldots, j$ **do**
    for $\ell = 1, \ldots, k$ **do**
      for $m = \ell, \ldots, n$ **do**
        $f()$

Solution: f is called

$$\sum_{j=1}^{n}\sum_{k=1}^{j}\sum_{l=1}^{k}\sum_{m=l}^{n} 1 \leq \sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{l=1}^{n}\sum_{m=1}^{n} 1 = n^4 \leq \mathcal{O}(n^4)$$

times. Notice that for $n \geq 4$

$$\sum_{j=1}^{n}\sum_{k=1}^{j}\sum_{l=1}^{k}\sum_{m=l}^{n} 1 \geq \sum_{j=\lceil \frac{2n}{3}\rceil}^{n}\ \sum_{k=\lceil \frac{n}{3}\rceil}^{\lceil \frac{2n}{3}\rceil}\ \sum_{l=1}^{\lceil \frac{n}{3}\rceil}\ \sum_{m=\lceil \frac{n}{3}\rceil}^{n} 1 \geq (\frac{n}{3} - 1)^4 \geq \frac{n^4}{12^4},$$

so actually we have

$$\sum_{j=1}^{n}\sum_{k=1}^{j}\sum_{l=1}^{k}\sum_{m=l}^{n} 1 = \Theta(n^4).$$

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:

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

Solution: Expanding the recurrence relation yields

$$\begin{aligned} T(n) &= 3T(n-1) - 2 \\ &= 3(3T(n-2) - 2) - 2 = 9T(n-2) - 8 \\ &= 9(3T(n-3) - 2) - 8 = 27T(n-3) - 26 \\ &\vdots \end{aligned}$$

The emerging pattern is

$$T(n) = 3^k T(n-k) - 3^k + 1,$$

and plugging in $k = n$ yields

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

Let us now prove (4) by induction on n. The base case holds since $2 \cdot 3^0 + 1 = 3 = T(0)$. Let $n \geq 1$ and assume that (4) holds for $n-1$, i.e. that $T(n-1) = 2 \cdot 3^{n-1} + 1$. Then using the recurrence relation we get

$$T(n) = 3T(n-1) - 2 = 3(2 \cdot 3^{n-1} + 1) - 2 = 2 \cdot 3^n + 1,$$

which shows that (4) holds for n and concludes the proof.

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*.*

Solution: Expanding the recurrence relation yields

$$\begin{aligned} T(n) &\leq 4T(n/2) + 3\log_2 n \\ &\leq 4(4T(n/4) + 3\log_2(n/2)) + 3\log_2 n \leq 16T(n/4) + 15\log_2 n \\ &\leq 16(4T(n/8) + 3\log_2(n/4)) + 15\log_2 n \leq 64T(n/8) + 63\log_2 n \\ &\vdots \end{aligned}$$

The emerging pattern is

$$T(n) \leq 4^k T(n/2^k) + (4^k - 1)\log_2 n,$$

and plugging in $k = \log_2 n$ yields

$$T(n) \leq 4^{\log_2 n} T(1) + (4^{\log_2 n} - 1)\log_2 n = n^2 + (n^2 - 1)\log_2 n = n^2(\log_2 n + 1) - \log_2 n. \tag{5}$$

Let us now prove (5) for $n = 2^m$ by induction on m. The base case holds since $1^2(\log_2 1 + 1) - \log_2 1 = 1 = T(1)$. Let $n \geq 2$ and assume that (5) holds for $n/2$, i.e. that

$$T(n/2) \leq (n/2)^2(\log_2(n/2) + 1) - \log_2(n/2).$$

Then, using the recurrence relation and the fact that $\log_2(n/2) = \log_2 n - 1$, we get

$$\begin{aligned} T(n) &\leq 4T(n/2) + 3\log_2 n \\ &\leq 4((n/2)^2(\log_2(n/2) + 1) - \log_2(n/2)) + 3\log_2 n \\ &= n^2 \log_2 n - 4(\log_2 n - 1) + 3\log_2 n \\ &= n^2 \log_2 + 4 - \log_2 n \\ &\leq n^2(\log_2 n + 1) - \log_2 n, \end{aligned}$$

which shows that (5) holds for n and concludes the inductive proof that

$$T(n) \leq n^2(\log_2 n + 1) - \log_2 n = n^2 \log_2 n + n^2 - \log_2 n.$$

Thus, $T(n) \leq n^2 \log_2 n + n^2$ and since $n^2 \log_2 n \leq \mathcal{O}(n^2 \log n)$ and $n^2 \leq \mathcal{O}(n^2 \log n)$, we conclude that $T(n) \leq \mathcal{O}(n^2 \log n)$.

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

$$\triangle^* = \max_{1 \leq a \leq b < c \leq n} \sum_{i=a}^{b} A_i - \sum_{j=b+1}^{c} A_j. \tag{6}$$

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 $\mathcal{O}(n)$.

Solution: We can rewrite $\triangle^*$ in the following way

$$\triangle^* = \max_{1 \leq b < n} \left(\max_{1 \leq a \leq b} \sum_{i=a}^{b} A_i + \max_{b < c \leq n} \left(- \sum_{j=b+1}^{c} A_j \right) \right) \tag{7}$$

because when b is fixed $\max\limits_{1 \leq a \leq b < c \leq n} \sum_{i=a}^{b} A_i - \sum_{j=b+1}^{c} A_j = \max\limits_{1 \leq a \leq b} \sum_{i=a}^{b} A_i + \max\limits_{b < c \leq n} \left(- \sum_{j=b+1}^{c} A_j \right)$.

Thus, let $P_b := \max\limits_{1 \leq a \leq b} \sum_{i=a}^{b} A_i$ and let $M_b := \max\limits_{b < c \leq n} - \sum_{j=b+1}^{c} A_j$. We utilize the facts that all P_b's and M_b's can be computed in linear time using

$$P_0 = 0 \quad \text{and} \quad P_b = \max(A_b, P_{b-1} + A_b) \tag{8}$$

and

$$M_n = 0 \quad \text{and} \quad M_b = \max(-A_{b+1}, M_{b+1} - A_{b+1}). \tag{9}$$

Once we have computed the P_b and M_b, we can compute the maximum in linear time as $\triangle^* = \max_{1 \leq b < n} P_b + M_b$. To clarify the computation, consider the following pseudocode.

Algorithm 7 MaximumSubarrayDifference(A)

```
P_0 = 0
M_n = 0
for b ∈ {1, ..., n} do
    P_b ← max(A_b, P_{b−1} + A_b)
    M_{n−b} ← max(−A_{n−b+1}, M_{n−b+1} − A_{n−b+1})
△ ← 0
for b ∈ {1, ..., n − 1} do
    if △ < P_b + M_b then
        △ ← P_b + M_b
return △
```

Correctness: The correctness of our algorithm only depends on the correctness of our recurrences for P_b and M_b. We show the correctness of the recurrence for P_b, i.e., that $P_b = \max_{1 \leq a \leq b} \sum_{i=a}^{b} A_i$, by mathematical induction, the recurrence for M_b can be proved analogously.

Base case $b = 1$: $P_b = \max(A_1, 0 + A_1) = A_1 = \max_{1 \leq a \leq 1} \sum_{i=a}^{1} A_i$.

Induction hypothesis: For some $k \geq 1$ we have $P_k = \max_{1 \leq a \leq k} \sum_{i=a}^{k} A_i$.

Induction step $k \to k + 1$: $P_{k+1} = \max(A_{k+1}, P_k + A_{k+1})$. Thus, by definition of the maximum $P_{k+1} = P_k + A_{k+1}$ if P_k is positive and A_{k+1} else. By the induction hypothesis we have $P_{k+1} = \max_{1 \leq a \leq k+1} \sum_{i=a}^{k+1} A_i = \begin{cases} P_k + A_{k+1} & \text{if } P_k > 0, \\ A_{k+1} & \text{else.} \end{cases}$

Runtime: Both for-loops perform n iterations with a constant amount of operations per iteration. Thus, the proposed algorithm is in $\mathcal{O}(n)$.

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,

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

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?

Solution: We start with the computation of a matrix of cumulative column sums

$$C_{ij} = \sum_{k=0}^{i} M_{kj}.$$

Then for each pair of rows a and b, $a \leq b$, we compute an array of column sums inside the stripe between a and b, that is

$$A_j = \sum_{i=a}^{b} M_{ij} = C_{bj} - C_{a-1j}, \quad 0 \leq j < n.$$

(If $a = 0$, $A_j = C_{bj}$).

Then we use a procedure MaxSubarraySum(A) which returns maximal subarray sum of A in time $\mathcal{O}(n)$. Maximal subarray sum of A is equal to

$$P(a, b) = \max_{0 \leq c \leq d < n} \sum_{i=a}^{b} \sum_{j=c}^{d} M_{ij}.$$

To find maximal submatrix sum, we maximize $P(a, b)$. For more details, see the pseudocode below.

Algorithm 8 Computation of max submatrix sum

```
procedure MaxSubmatrixSum(M)
    C ← M
    for 1 ≤ i < n do
        for 0 ≤ j < n do
            C_ij ← C_{i-1 j} + M_ij
    S ← 0
    for 0 ≤ a < n do
        for a ≤ b < n do
            for 0 ≤ j < n do
                if a = 0 then
                    A_j ← C_bj
                else
                    A_j ← C_bj − C_{a-1 j}
            S ← max{S, MaxSubarraySum(A)}
    return S
```

Computing cumulative column sum matrix takes time $\mathcal{O}(n^2)$.
There are $\mathcal{O}(n^2)$ pairs of rows (a, b) and for each pair we perform $\mathcal{O}(n)$ operations: $\mathcal{O}(n)$ operations to compute the array of column sums, $\mathcal{O}(n)$ operations to find maximal subarray sum and $\mathcal{O}(1)$ operations to compare maximal subarray sum with the current maximal value.
Hence the total number of operations is $\mathcal{O}(n^3)$.