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

18 October 2021

Algorithms & Data Structures — Exercise sheet 4 — HS 21

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 25th.

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

The following theorem is very useful for running time analysis of divide-and-conquer algorithms.

Theorem 1 (Master theorem). *Let $a, C > 0$ and $b \geq 0$ be constants and $T : \mathbb{N} \to \mathbb{R}^+$ a function such that for all even $n \in \mathbb{N}$,*

$$T(n) \leq aT(n/2) + Cn^b. \tag{1}$$

Then for all $n = 2^k$, $k \in \mathbb{N}$,

- *If $b > \log_2 a$, $T(n) \leq O(n^b)$.*
- *If $b = \log_2 a$, $T(n) \leq O(n^{\log_2 a} \cdot \log n)$.*
- *If $b < \log_2 a$, $T(n) \leq O(n^{\log_2 a})$.*

If the function T is increasing, then the condition $n = 2^k$ can be dropped. If (1) holds with "=", then we may replace O with Θ in the conclusion.

This generalizes some results that you have already seen in this course. For example, the running time of Karatsuba algorithm satisfies $T(n) \leq 3T(n/2) + 100n$, so $a = 3$ and $b = 1 < \log_2 3$, hence $T(n) \leq O(n^{\log_2 3})$. Another example is binary search: its running time satisfies $T(n) \leq T(n/2) + 100$, so $a = 1$ and $b = 0 = \log_2 1$, hence $T(n) \leq O(\log n)$.

Exercise 4.1 *Applying Master theorem.*

For this exercise, assume that n is a power of two (that is, $n = 2^k$, where $k \in \{0, 1, 2, 3, 4, \ldots\}$).

a) Let $T(1) = 1, T(n) = 4T(n/2) + 100n$ for $n > 1$. Using Master theorem, show that $T(n) \leq O(n^2)$.

Solution: We can apply Theorem 1 with $a = 4$, $b = 1$ and $C = 100$. In this case, $b < \log_2 a$, and therefore the by the Master theorem we have $T(n) \leq O(n^{\log_2 a}) = O(n^2)$.

b) Let $T(1) = 5, T(n) = T(n/2) + \frac{3}{2}n$ for $n > 1$. Using Master theorem, show that $T(n) \leq O(n)$.

Solution: We can apply Theorem 1 with $a = 1$, $b = 1$ and $C = \frac{3}{2}$. In this case, $b > \log_2 a$, and therefore the by the Master theorem we have $T(n) \leq O(n^b) = O(n)$.

c) Let $T(1) = 4$, $T(n) = 4T(n/2) + \frac{7}{2}n^2$ for $n > 1$. Using Master theorem, show that $T(n) \leq O(n^2 \log n)$.

Solution: We can apply Theorem 1 with $a = 4$, $b = 2$ and $C = \frac{7}{2}$. In this case, $b = \log_2 a$, and therefore the by the Master theorem we have $T(n) \leq O(n^{\log_2 a} \cdot \log n) = O(n^2 \log n)$.

In the second exercise you will see some examples of recurrences that can be analyzed in O-Notation using Master theorem. These three examples show that the bounds in Master theorem are tight.

Exercise 4.2 *Solving Recurrences* **(1 point)**.

For this exercise, assume that n is a power of two (that is, $n = 2^k$, where $k \in \{0, 1, 2, 3, 4, \ldots\}$).

a) Consider the following algorithm:

Algorithm 1 $g(n)$

```
if n > 1 then
    for i = 1, 2 do
        g(n/2)
        g(n/2)
        for k = 1, ..., n do
            f()
else
    f()
```

The number of calls of f is given by the recurrence relation $T(1) = 1$ and $T(n) = 4T(\frac{n}{2}) + 2n$ for $n \geq 2$. Using mathematical induction show that the *closed-form expression* for $T(n)$ is $T(n) = 3n^2 - 2n$.

Hint: *Use induction over* $k = \log_2 n$.

Solution:

Base case. Let $k = 0$, $n = 1$. $T(1) = 1 = 3 \cdot 1^2 - 2 \cdot 1$.

Induction Hypothesis. We assume that for some $k \geq 0$ and $n = 2^k$ it holds that

$$T(n) = 3n^2 - 2n.$$

Inductive step ($k \to k + 1$). We know that $T(2n) = 4T(n) + 2 \cdot 2n = 4T(n) + 4n$. Using the induction hypothesis for $T(n)$, we have

$$T(2n) = 4 \cdot \left(3n^2 - 2n\right) + 4n = 12n^2 - 4n = 3 \cdot (2n)^2 - 2 \cdot (2n),$$

as desired.

b) Consider the following algorithm:

Algorithm 2 $g(n)$

```
if n > 1 then
    for i = 1, ..., 3n/2 do
        f()
    g(n/2)
else
    f()
    f()
    f()
    f()
    f()
```

Find the recurrence relation for the number of calls of f, the closed form expression for it, and using mathematical induction prove that it has this closed-form expression.

Hint: *The closed-form expression should be of the form $T(n) = a \cdot n + b$ for some real numbers a and b.*

Solution: The number of calls of f is given by the recurrence relation $T(1) = 5$ and $T(n) = T(\frac{n}{2}) + \frac{3}{2}n$ for $n \geq 2$.

Since $T(1) = 5$ and $T(2) = 8$, we can conclude that $a = 3$ and $b = 2$. We will now show by induction that $T(n) = 3n + 2$.

Base case. Let $k = 0$, $n = 1$. $T(1) = 5 = 3 \cdot 1 + 2$.

Induction Hypothesis. We assume that for some $k \geq 0$ and $n = 2^k$ it holds that

$$T(n) = 3n + 2.$$

Inductive step ($k \to k + 1$). We know that $T(2n) = T(n) + \frac{3}{2} \cdot 2n = T(n) + 3n$. Using the induction hypothesis for $T(n)$, we have

$$T(2n) = (3n + 2) + 3n = 6n + 2 = 3 \cdot (2n) + 2,$$

as desired.

c) Consider the following algorithm:

Algorithm 3 $g(n)$

```
if n > 1 then
    for i = 1, ..., 4 do
        g(n/2)
    for i = 1, ..., n/2 do
        for j = 1, ..., 7n do
            f()
else
    for i = 1, ..., 4 do
        f()
```

Find the recurrence relation for the number of calls of f, the closed form expression for it, and using mathematical induction prove that it has this closed-form expression.

Hint: *The closed-form expression should be of the form $T(n) = a \cdot n^2 \log_2 n + b \cdot n^2$ for some real numbers a and b.*

Solution: The number of calls of f is given by the recurrence relation $T(1) = 4$ and $T(n) = 4T(\frac{n}{2}) + \frac{7}{2}n^2$ for $n \geq 2$.

Since $T(1) = 4$ and $T(2) = 30$, we can conclude that $a = 7/2$ and $b = 4$. We will now show by induction that $T(n) = \frac{7}{2}n^2 \log_2 n + 4n^2$.

Base case. Let $k = 0$, $n = 1$. $T(1) = 4 = \frac{7}{2} \cdot 0 + 4 \cdot 1^2 = \frac{7}{2} \cdot 1^2 \cdot \log_2 1 + 4 \cdot 1^2$.

Induction Hypothesis. We assume that for some $k \geq 0$ and $n = 2^k$ it holds that

$$T(n) = \tfrac{7}{2}n^2 \log_2 n + 4n^2.$$

Inductive step ($k \to k+1$). We know that $T(2n) = 4T(n) + \frac{7}{2} \cdot (2n)^2 = 4T(n) + 14n^2$. Using the induction hypothesis for $T(n)$, we get

$$\begin{aligned} T(2n) &= 4\left(\tfrac{7}{2}n^2 \log_2 n + 4n^2\right) + 14n^2 = 16n^2 + 14n^2 \log_2 n + 14n^2 \\ &= 4 \cdot (2n)^2 + 14n^2 (1 + \log_2 n) = 4 \cdot (2n)^2 + \tfrac{7}{2}(2n)^2 \log_2(2n), \end{aligned}$$

as desired.

The following definitions are closely related to O-Notation and are also useful in running time analysis of algorithms.

Definition 1 (Ω-Notation)**.** Let $n_0 \in \mathbb{N}$, $N := \{n_0, n_0 + 1, \ldots\}$ and let $f : N \to \mathbb{R}^+$. $\Omega(f)$ is the set of all functions $g : N \to \mathbb{R}^+$ such that $f \in O(g)$. One often writes $g \geq \Omega(f)$ instead of $g \in \Omega(f)$.

Definition 2 (Θ-Notation)**.** Let $n_0 \in \mathbb{N}$, $N := \{n_0, n_0 + 1, \ldots\}$ and let $f : N \to \mathbb{R}^+$. $\Theta(f)$ is the set of all functions $g : N \to \mathbb{R}^+$ such that $f \in O(g)$ and $g \in O(f)$. One often writes $g = \Theta(f)$ instead of $g \in \Theta(f)$.

Exercise 4.3 *Asymptotic notations.*

a) Describe the (worst-case) running time of the following algorithms in Θ-Notation.

   1) Karatsuba algorithm. **Solution:** $\Theta(n^{\log_2(3)})$

   2) Binary Search. **Solution:** $\Theta(\log_2(n))$

   3) Bubble Sort. **Solution:** $\Theta(n^2)$

b) **(This subtask is from January 2019 exam).** For each of the following claims, state whether it is true or false. You don't need to justify your answers.

claim	true	false
$\frac{n}{\log n} \leq O(\sqrt{n})$	☐	☐
$\log(n!) \geq \Omega(n^2)$	☐	☐
$n^k \geq \Omega(k^n)$, if $1 < k \leq O(1)$	☐	☐
$\log_3 n^4 = \Theta(\log_7 n^8)$	☐	☐

claim	true	false
$\frac{n}{\log n} \leq O(\sqrt{n})$	☐	☒
$\log n! \geq \Omega(n^2)$	☐	☒
$n^k \geq \Omega(k^n)$	☐	☒
$\log_3 n^4 = \Theta(\log_7 n^8)$	☒	☐

c) **(This subtask is from August 2019 exam).** For each of the following claims, state whether it is true or false. You don't need to justify your answers.

claim	true	false
$\frac{n}{\log n} \geq \Omega(n^{1/2})$	☐	☐
$\log_7(n^8) = \Theta(\log_3(n^{\sqrt{n}}))$	☐	☐
$3n^4 + n^2 + n \geq \Omega(n^2)$	☐	☐
$(*) \quad n! \leq O(n^{n/2})$	☐	☐

claim	true	false
$\frac{n}{\log n} \geq \Omega(n^{1/2})$	☒	☐
$\log_7(n^8) = \Theta(\log_3(n^{\sqrt{n}}))$	☐	☒
$3n^4 + n^2 + n \geq \Omega(n^2)$	☒	☐
$(*) \quad n! \leq O(n^{n/2})$	☐	☒

Note that the last claim is challenge. It was one of the hardest tasks of the exam. If you want a 6 grade, you should be able to solve such exercises.

Solution: All claims except for the last one are easy to verify using either the theorem about the limit of $\frac{f(n)}{g(n)}$ or simply the definitions of O, Ω and Θ. Thus, we only present the solution for the last one.

Note that for all $n \geq 1$,

$$n! \geq 1 \cdot 2 \cdots n \geq \lceil n/10 \rceil \cdots n \geq \lceil n/10 \rceil^{0.9n} \geq (n/10)^{0.9n}\,.$$

Let's show that $(n/10)^{0.9n}$ grows asymptotically faster than $n^{n/2}$.

$$\lim_{n\to\infty} \frac{n^{n/2}}{(n/10)^{0.9n}} = \lim_{n\to\infty} 10^{0.9n} \cdot n^{-0.4n} = \lim_{n\to\infty} (10^{9/4}/n)^{0.4n} = 0\,.$$

Hence it is not true that $(n/10)^{0.9n} \leq O(n^{n/2})$ and so it is not true that $n! \leq O(n^{n/2})$.

Exercise 4.4 *Exchange sort* **(1 point)**.

Consider the following algorithm, which takes an unsorted array $A = A[1, \ldots, n]$ as input and returns the array in ascending order:

Algorithm 4 ExchangeSort(A)

```
for 1 ≤ i ≤ n do
    for i + 1 ≤ j ≤ n do
        if A[j] < A[i] then
            T ← A[j]
            A[j] ← A[i]
            A[i] ← T
return A
```

a) Formulate an invariant INV(i) that holds at the end of the i-th iteration of the outer for-loop.

Solution: After i iterations of the for-loop, the i first entries of the array are the i smallest entries of the input array A, sorted in ascending order.

b) Using the invariant from part (a), prove the correctness of the algorithm. Specifically, prove the following three assertions:

(i) INV(1) holds.

(ii) If INV(i) holds, then INV$(i+1)$ holds (for all $1 \leq i < n$).

(iii) INV(n) implies that ExchangeSort(A) correctly sorts the array A.

Solution:

(i) The first inner for-loop compares the (current) first element of the array with all other elements, and swaps them if the former is larger than the latter. Therefore, after the 1st iteration of the outer for-loop, the first element of the array is the smallest element of A, which means that INV(1) holds.

(ii) Let $1 \leq i < n$. Assuming that INV(i) holds, we know that before the $(i+1)$st iteration of the outer for-loop, the i first entries of the array are the i smallest entries of the input array A sorted in ascending order. During the $(i+1)$st iteration, the smallest element among the remaining part of the array (namely $A[i+1, \ldots, n]$) will be placed at the $(i+1)$st position, so that now the the $i+1$ first entries of the array are the $i+1$ smallest entries of the input array in ascending order. Therefore, INV$(i+1)$ holds.

(iii) INV(n) means that the array contains the n smallest values of A (i.e. all values of A, since A has length n) in increasing order. So the array after the n-th iteration, which is the array returned by ExchangeSort(A), is indeed the sorted array.

Exercise 4.5 *Searching in Nice Matrices* **(1 point)**.

Let $A[1 \ldots n][1 \ldots n]$ be an $n \times n$ matrix. We say that the matrix A is *nice* if it satisfies the following two properties:

- For every $1 \leq i \leq n$ and every $1 \leq j_1 < j_2 \leq n$, we have $A[i][j_1] < A[i][j_2]$.
- For every $1 \leq i_1 < i_2 \leq n$ and every $1 \leq j \leq n$, we have $A[i_1][j] < A[i_2][j]$.

We are given a value $b \in \mathbb{Z}$ that is guaranteed to be present in a nice matrix A. We are asked to search for indices $i, j \in \{1, \ldots, n\}$ such that $A[i][j] = b$. A trivial solution would be to scan all the entries of A until finding b. This would take $O(n^2)$ time. In this exercise, we would like to find better solutions.

a) An algorithm that is better than the trivial solution is one that applies binary search on every row. What is the runtime of this algorithm?

Solution: Since every binary search takes $O(\log n)$, and since we apply binary search on (at most) n rows, the runtime of the algorithm is $O(n \log n)$.

Now we would like to find an algorithm that is even better than the one in a). Let $1 \leq i_1 \leq i_2 \leq n$ and $1 \leq j_1 \leq j_2 \leq n$ and assume that we already know that b is present in the submatrix $A[i_1 \dots i_2][j_1 \dots j_2]$. Notice that by comparing $A[i_2][j_1]$ to b, we will end up with one of the following three possibilities:

- $A[i_2][j_1] = b$, which means that we found b.
- $A[i_2][j_1] < b$, which means that b must be present in the submatrix $A[i_1 \dots i_2][(j_1 + 1) \dots j_2]$.
- $A[i_2][j_1] > b$, which means that b must be present in the submatrix $A[i_1 \dots (i_2 - 1)][j_1 \dots j_2]$.

b) Use the above observation to write the pseudocode of an algorithm that solves the search problem in $O(n)$ runtime.

Solution:

Algorithm 5 Searching in a Nice Matrix

```
procedure searchNiceMatrix(A, b)
    i_1 ← 1
    i_2 ← n
    j_1 ← 1
    j_2 ← n
    while i_1 ≤ i_2 and j_1 ≤ j_2 do
        if A[i_2][j_1] = b then
            return (i_2, j_1)
        else if A[i_2][j_1] < b then
            j_1 ← j_1 + 1
        else
            i_2 ← i_2 − 1
```