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 $T(n) \leq O(n^2)$.

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

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

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 $k = \log_2 n$.*

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.*

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*.*

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.

   2) Binary Search.

   3) Bubble Sort.

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 $1 < k \leq O(1)$	☐	☐
$\log_3 n^4 = \Theta(\log_7 n^8)$	☐	☐

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

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.

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.

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.

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

Let $A[1 \dots n][1 \dots n]$ 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 $i, j \in \{1, \dots, n\}$ 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?

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.