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

14. October 2019

Algorithms & Data Structures — Exercise sheet 4 — HS 19

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 21st.

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 $T : \mathbb{N} \to \mathbb{R}^+$ be a non-decreasing function such that for all $k \in \mathbb{N}$ and $n = 2^k$,*

$$T(n) \leq aT(n/2) + \mathcal{O}(n^b)$$

for some constants $a > 0$ and $b \geq 0$. Then

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

This theorem generalizes some results that you have 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 $T(n) \in \mathcal{O}(n^{\log_2 3})$. The other 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 $T(n) \in \mathcal{O}(\log n)$.

In parts a), b) and c) of the first exercise you will see some examples of recurrences that can be analyzed in $\mathcal{O}$-Notation using Master theorem. These three examples show that the bounds in Master theorem are tight.

Exercise 4.1 *Solving Recurrences* (2 points).

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

a) Let $f : \mathbb{N} \to \mathbb{R}^+$ be defined as $f(1) = c$, and $f(n) = 2f\left(\frac{n}{2}\right) + d \cdot n^2$ for $n \geq 2$. Using mathematical induction show that $f(n) = 2d \cdot n^2 + (c - 2d) \cdot n$.

Base case. Let $k = 0$, $n = 1$. $f(1) = c = 2d \cdot 1^2 + (c - 2d) \cdot 1$.

Induction Hypothesis. We assume that there is some $k \geq 0$ and $n = 2^k$ such that

$$f(n) = 2d \cdot n^2 + (c - 2d) \cdot n.$$

Inductive step ($k \to k+1$). We know that $f(2n) = 2f(n) + 4d \cdot n^2$. Using induction hypothesis for $f(n)$:

$$f(2n) = 2\left(2a \cdot n^2 + (c - 2d) \cdot n\right) + 4d \cdot n^2 = 2d \cdot (2n)^2 + (c - 2d) \cdot (2n).$$

as desired.

b) Let $f : \mathbb{N} \to \mathbb{R}^+$ be defined as $f(1) = c$, and $f(n) = 2f(\frac{n}{2}) + d \cdot n$, for $n \geq 2$. Using mathematical induction show that $f(n) = c \cdot n + d \cdot n \cdot \log_2 n$.

Base case. Let $k = 0$. Then

$$f(2^0) = f(1) = c = c \cdot 2^0 + a \cdot 2^0 \cdot 0.$$

Induction Hypothesis. We assume that for some $k \geq 0$ it holds that

$$f(2^k) = c \cdot 2^k + a \cdot 2^k \cdot k.$$

Inductive step ($k \to k+1$). We know that

$$f(2^{k+1}) = 2f(2^k) + a \cdot 2^{k+1}.$$

Using induction hypothesis for $f(2^k)$:

$$\begin{aligned} f(2^{k+1}) &= 2\left(c \cdot 2^k + a \cdot 2^k \cdot k\right) + a \cdot 2^{k+1} \\ &= c \cdot 2^{k+1} + a \cdot 2^{k+1} \cdot k + a \cdot 2^{k+1} \\ &= c \cdot 2^{k+1} + a \cdot 2^{k+1}(k+1), \end{aligned}$$

as desired.

c) Let $f : \mathbb{N} \to \mathbb{R}^+$ be defined as $f(1) = c$, and $f(n) = 8f(\frac{n}{2}) + d \cdot n^2$, for $n \geq 2$. Using mathematical induction show that $f(n) = (c + d) \cdot n^3 - d \cdot n^2$.

Base case. Let $k = 0$, $n = 1$. Then $f(1) = c = (c + a) \cdot 1^3 - a \cdot 1$.

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

$$f(n) = (c + a) \cdot n^3 - a \cdot n^2.$$

Inductive step ($k \to k+1$). We know that $f(2n) = 8f(n) + 4a \cdot n^2$. Using induction hypothesis for $f(n)$:

$$f(2n) = 8\left((c + a) \cdot n^3 - a \cdot n^2\right) + 4a \cdot n^2 = (c + a) \cdot (2n)^3 - a \cdot (2n)^2.$$

as desired.

d) Consider the following recursive function:

Algorithm 1 $g(m)$

```
if m > 1 then
    g(m − 1)
    g(m − 1)
    g(m − 1)
else
    f()
```

Determine the number of calls of the function f in $\mathcal{O}$-notation. Prove your result.

Solution: Let $T(m)$ be the number of calls of f that g performs on input m. $T(m) = 3T(m-1)$ and $T(1) = 1$, so our guess is $T(m) = 3^{m-1}$. Let's prove it by induction.

- **Base Case.**
  Let $m = 1$. Then $T(m) = 1 = 3^{m-1}$.
- **Induction Hypothesis.**
  Assume that the property holds for some positive integer l. That is, $T(l) = 3^{l-1}$.
- **Inductive Step.**
  We must show that the property holds for $l + 1$.

$$T(l+1) = 3 \cdot T(l) \overset{\text{I.H.}}{=} 3 \cdot 3^{l-1} = 3^l.$$

By the principle of mathematical induction, this is true for any positive integer m.

Hence $T(m) = \mathcal{O}(3^m)$. Notice that this is tight, since $3^m \in \mathcal{O}(3^{m-1})$.

e) Consider the following recursive function (recall that in this exercise $n = 2^k$, where $k \in \mathbb{N}_0$):

Algorithm 2 $h(n)$

```
if n > 1 then
    f()
    h(n/2)
    f()
    h(n/2)
else
    f()
```

Using Master theorem, determine the number of calls of the function f in $\mathcal{O}$-notation.

Solution: Let $T(n)$ be the number of calls of f that h performs on input n. Since we call f two times on the input that is twice smaller, we have $T(n) = 2 + 2T(n/2)$ and $T(1) = 1$. Thus, by applying the Master theorem with $a = 2$ and $b = 0$, we get $T(n) \in \mathcal{O}(n^{\log_2 2}) = \mathcal{O}(n)$

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

Definition 1 (Ω-Notation)**.** Let $f : N \to \mathbb{R}^+$. $\Omega(f)$ is a set of all functions $g : N \to \mathbb{R}^+$ such that there exists $C > 0$ (that may depend on g) such that for all $n \in N$, $g(n) \geq Cf(n)$.

Definition 2 (Θ-Notation)**.** Let $f : N \to \mathbb{R}^+$. $\Theta(f) := \mathcal{O}(f) \cap \Omega(f)$.

As for $\mathcal{O}$-Notation, typically $N = \mathbb{N}$, but sometimes we will consider other sets, e.g. $N = \{2, 3, 4, \ldots\}$.

We will usually write $g \leq \mathcal{O}(f)$ instead of $g \in \mathcal{O}(f)$, $g \geq \Omega(f)$ instead of $g \in \Omega(f)$, and $g = \Theta(f)$ instead of $g \in \Theta(f)$.

Exercise 4.2 *Asymptotic notations.*

a) Show that $g \geq \Omega(f)$ if and only if $f \leq \mathcal{O}(g)$.
Solution:

Proof. To show the equivalence we show both directions:

The if-part $\Rightarrow$: If $g \geq \Omega(f)$ there exists a $C > 0$ such that for all $n \in N$ we have $Cf(n) \leq g(n)$. Thus, dividing by C on both sides yields $f(n) \leq \frac{1}{C}g(n)$, which means that $f \leq \mathcal{O}(g)$.

The only-if-part $\Leftarrow$: If $f \leq \mathcal{O}(g)$ there exists a $C > 0$ such that for all $n \in N$ we have $f(n) \leq Cg(n)$. Thus, dividing by C on both sides yields $\frac{1}{C}f(n) \leq g(n)$, which means that $g \geq \Omega(f)$.

□

b) 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:** $\Theta(n^2)$

c) **(This subtask is from January 2019 exam).** For each of the following claims, state whether it is true or false.

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

d) **(This subtask is from August 2019 exam).** For each of the following claims, state whether it is true or false.

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 $\mathcal{O}, \Omega$ and Θ. Thus, we only present the solution for the last one.

Consider $n!$ for an even n, by definition we have $n! = n(n-1)\cdots 1$. By grouping terms we can write $n!$ as a product of $\frac{n}{2}$ factors:

$$n! = \prod_{k=0}^{n/2-1} (n-k)(k+1).$$

To show that the claim is false we first show that each factor of this product is larger than n, for $n > 0$ and $k > 0$:

$$\begin{aligned} nk + n - k^2 - k > n &\Leftrightarrow nk > k^2 + k \\ &\Leftrightarrow n > n/2 + 1 > k + 1. \end{aligned}$$

Now, we can consider the quotient

$$\begin{aligned} \frac{n^{n/2}}{n!} &= \prod_{k=0}^{n/2-1} \frac{n}{(n-k)(k+1)} \\ &= \prod_{k=1}^{n/2-1} \frac{n}{(n-k)(k+1)}. \end{aligned}$$

We observe that, for $k, l \in \{0, \ldots, n/2-1\}$ with $k < l$, we have $(n-k)(k+1) < (n-l)(l+1)$ and that for $n > 4$, we have $\frac{n}{(n-1)(1+1)} < \frac{2}{3}$. Thus, for $n > 4$:

$$\begin{aligned} \frac{n^{n/2}}{n!} &\leq \prod_{k=1}^{n/2-1} \frac{n}{2n-2} \\ &\leq \left(\frac{2}{3}\right)^{n/2-1}. \end{aligned}$$

For $n \to \infty$ the product of $n/2$ factors < 1 is zero. An analogous argument works for n odd. Thus, by the Theorem 1.1. from the lecture we have that $n! \not\leq \mathcal{O}(n^{n/2})$.

Exercise 4.3 *Search with an oracle* **(1 point)**.

Let $f : \mathbb{N} \to \mathbb{R}^+$ be a strictly increasing function such that $f(1) = 1$ and $\lim_{n\to\infty} f(n) = \infty$. You don't know anything else about f, but you can call f at any input you want and get its value. You are given some number $n \in \mathbb{N}$, your goal is to find $m \in \mathbb{N}$ such that $f(m) \leq n$, and $f(m+1) > n$ (note that for any n such m always exists). Describe a procedure of finding such m.

A trivial solution to this problem is to call f at $i = 2, 3, 4, \ldots$ until you find an i with $f(i) > n$. Then you output $m := i - 1$. This algorithm needs $\Theta(m)$ calls, where m is the *output* of the algorithm.

Find a procedure that is more efficient, i.e., that needs asymptotically strictly less than $\Theta(m)$ calls. How many calls does your solution need asymptotically?

Hint: *Start with finding some number M such that $f(M) > n$. The trivial algorithm needs $\Theta(m)$ steps to achieve this. Find a more efficient strategy.*

Solution: We find M by starting with $M = 1$, and then doubling M until we have $f(M) > n$. After that, we use binary search for the value n in the interval $[1, M]$ in order to determine the value of m. A pseudocode description is:

Algorithm 3 $findm(f, n)$

```
M ← 1
while f(M) ≤ n do
    M ← 2M
return BinarySearch(n, [f(1), ..., f(M)])
```

We could optimize slightly by searching in the array $[f(M/2), f(M)]$ instead of $[f(1), f(M)]$.

Other than in the lecture, we do not search in an array that is explicitly given. Instead, we search in the implicitly given array $[f(1), f(2), \ldots, f(M)]$, but we never need to construct this array explicitly. We only need to compare n with the entries of the i-th cell for some i, and we can do this by simply checking "$f(i) < n$?", which we can do even if we haven't constructed the array explicitly. In other words, whenever we need a value from the array, we can just generate it on the fly.

The proposed algorithm consists of two parts: 1. determining M, which requires $\Theta(\log(m))$ steps, and, 2. finding m in the set $\{M/2, \ldots, M\}$ using an adapted binary search that performs $\Theta(\log(M))$ steps. For our search strategy for M, we know $M < 2m$. Thus, the runtime of our solution is in $\Theta(\log(m))$ as desired.