Albert-Ludwigs-Universität, Institut für Informatik
Prof. Dr. Fabian Kuhn
M. Ahmadi, P. Schneider

November 16, 2017

# Algorithm Theory - Winter Term 2017/2018
# Exercise Sheet 1

**Hand in by Thursday 10:15, November 2, 2017**

## Exercise 1: Landau Notation - Formal Proofs (2+2+3+3 Points)

For a function $f(n)$, the set $\mathcal{O}\big(f(n)\big)$ contains all functions $g(n)$ that are *asymptotically* not growing faster than $f(n)$. This is formalized as follows:

$$\mathcal{O}\big(f(n)\big) = \{g(n) \mid \exists c > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0 : g(n) \leq cf(n)\}$$

The set $\Omega\big(f(n)\big)$ contains all functions $g(n)$ with $f(n) \in \mathcal{O}\big(g(n)\big)$. Finally, $\Theta\big(f(n)\big)$, contains all functions $g(n)$ for which both $f \in \mathcal{O}\big(g(n)\big)$ and $g \in \mathcal{O}\big(f(n)\big)$ is true. For each pair of functions from (a) to (c) **prove** whether $g(n) \in \mathcal{O}\big(f(n)\big)$ or $g(n) \in \Omega\big(f(n)\big)$ or both, i.e., $g(n) \in \Theta\big(f(n)\big)$.

***Note****: You do not have to prove negative results $g(n) \notin \mathcal{O}\big(f(n)\big)$, it suffices to claim these correctly.*

(a) $g(n) = 100n,\ f(n) = 0.1 \cdot n^2$

(b) $g(n) = \sqrt[3]{n^2},\ f(n) = \sqrt{n}$

(c) $g(n) = \log_2(2^n \cdot n^3),\ f(n) = n$ ***Hint****: You may use that $\log_2 n \leq n$ for all $n \in \mathbb{N}$.*

(d) Fill out the following table (with ✓or ✗) based on whether the statement is true or false.

***Note****: You loose $\frac{1}{2}$ points for each wrong cell.*

| | $g \in \mathcal{O}(f)$ | $f \in \mathcal{O}(g)$ | $g \in \Omega(f)$ | $f \in \Omega(g)$ | $g \in \Theta(f)$ | $f \in \Theta(g)$ |
|---|---|---|---|---|---|---|
| a) | | | | | | |
| b) | | | | | | |
| c) | | | | | | |

## Sample Solution

(a) It is $100n \in \mathcal{O}(0.1n^2)$. To show that we require constants $c, M$ such that $100n \leq c \cdot 0.1n^2$ for all $n \geq M$. Obviously this is the case for $c = 1000$ and $M = 1$. However $0.1n^2 \notin \mathcal{O}(100n)$!

(b) We have $f(n) \in \mathcal{O}(g(n))$. Let $c := 1$ and $M := 1$. Then we have

$$\begin{aligned} & g(n) \leq c \cdot f(n) \\ \iff\ & n^{1/2} \leq n^{2/3} \\ \iff\ & 1 \leq n^{1/6} \\ \iff\ & 1 \leq n \end{aligned}$$

which is satisfied for $n \geq M = 1$. Consequently $g(n) \in \Omega(f(n))$. However $g(n) \notin \mathcal{O}(f(n))$!

(c) $g(n) \in \mathcal{O}(f(n))$ holds. We give $c > 0$ and $M \in \mathbb{N}$ such that for all $n \geq M : \log_2(2^n \cdot n^3) \leq c \cdot n$.

$$\log_2(2^n \cdot n^3) = \log_2(2^n) + \log_2(n^3) = n + 3 \cdot \log_2(n) \overset{\text{Hint}}{\leq} n + 3n = 4n.$$

Thus $\log_2(2^n \cdot n^3) \leq c \cdot n$ for $n \geq M := 1$ and $c := 4$.

We also have that $f(n) \in \mathcal{O}(g(n))$ holds because for all $n \in \mathbb{N}$:

$$f(n) = n \leq n + 3 \cdot \log_2(n) = \log_2(2^n \cdot n^3) = g(n).$$

Consequently we have $g(n) \in \Theta(f(n))$, i.e. $\mathcal{O}(f) = \mathcal{O}(g)$.

We obtain the following table:

| | $g \in \mathcal{O}(f)$ | $f \in \mathcal{O}(g)$ | $g \in \Omega(f)$ | $f \in \Omega(g)$ | $g \in \Theta(f)$ | $f \in \Theta(g)$ |
|---|---|---|---|---|---|---|
| a) | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ |
| b) | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ |
| c) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

## Exercise 2: Sort Functions by Asymptotic Growth (5 Points)

Using the $\mathcal{O}$-notation definition, give an ordered sequence of the following functions based on their asymptotic growth . Between each consecutive elements $g$ and $f$ in your list, insert either $\prec$ (in case $g \in \mathcal{O}(f)$ *and* $f \notin \mathcal{O}(g)$) or $\simeq$ (in case $g \in \mathcal{O}(f)$ *and* $f \in \mathcal{O}(g)$).

***Note**: You loose $\frac{1}{2}$ points for each error.*

| | | | |
|---|---|---|---|
| $n^2$ | $\sqrt{n}$ | $2^n$ | $\log(n^2)$ |
| $3^n$ | $n^{100}$ | $\log(\sqrt{n})$ | $(\log n)^2$ |
| $\log n$ | $10^{100}n$ | $n!$ | $n \log n$ |
| $n \cdot 2^n$ | $n^n$ | $\sqrt{\log n}$ | $n$ |

## Sample Solution

$$\begin{array}{lllllllll}
 & \sqrt{\log n} & \prec & \log(\sqrt{n}) & \simeq & \log n & \simeq & \log(n^2) \\
\prec & (\log n)^2 & \prec & \sqrt{n} & \prec & n & \simeq & 10^{100}n \\
\prec & n \log n & \prec & n^2 & \prec & n^{100} & \prec & 2^n \\
\prec & n \cdot 2^n & \prec & 3^n & \prec & n! & \prec & n^n
\end{array}$$

## Exercise 3: Recurrence Relations (3+3+3 Points)

Let $\alpha, \beta$ be *constants* and let $T(n)$ be a *monotonously increasing* function in $n$ with

$$T(1) \leq \alpha, \quad T(n) \leq 4 \cdot T(n/4) + \beta \cdot n.$$

(a) Guess an upper bound for $T(n)$ (as tight as possible, with the given knowledge about $T(n)$, e.g. by repeated substitution, cf. lecture). You may assume that $n = 4^k$ for some $k \in \mathbb{N}$.

(b) Prove your upper bound via induction on $k$. Assume that $n = 4^k$ for some $k \in \mathbb{N}$.

(c) Give an *asymptotic* upper bound ($\mathcal{O}$-Notation) for $T(n)$, assuming $n = 4^k$ for some $k \in \mathbb{N}$. Prove that the same bound applies to $T(n)$ for *all* $n \in \mathbb{N}$ (i.e., if $n$ is not necessarily a power of 4).

## Sample Solution

(a) We use repeated substitution to guess a tight (with the given knowledge) upper bound of $T(n)$.

$$\begin{aligned} T(n) &\le 4\cdot T(\tfrac{n}{4}) + \beta\cdot n \\ &\le 4\cdot(4\cdot T(\tfrac{n}{16}) + \beta\cdot\tfrac{n}{4}) + \beta\cdot n = 4^2\cdot T(\tfrac{n}{4^2}) + 2\beta n \\ &\le 4^2\cdot(4\cdot T(\tfrac{n}{4^3}) + \beta\cdot\tfrac{n}{4^2}) + 2\beta n = 4^3\cdot T(\tfrac{n}{4^3}) + 3\beta n \\ &\le \dots \overset{*}{\le} 4^k\cdot T(\tfrac{n}{4^k}) + k\beta n \overset{n=4^k}{=} n\cdot T(1) + \beta n\log_4 n \le \alpha n + \beta n \log_4 n \qquad \textit{*: 'educated guess'} \end{aligned}$$

(b) *Induction base:* Let $k = 0$, i.e. $n = 4^0 = 1$.

Then we have $T(4^0) = T(1) \le \alpha = \alpha\cdot 1 + \beta\cdot 1\cdot 0 = \alpha\cdot 1 + \beta\cdot 1\cdot\log_4 1$ ✓.

*Induction hypothesis:* Assume $T(4^k) \le \alpha 4^k + \beta 4^k k = \alpha n + \beta n\log_4 n$ for $k\in\mathbb{N}$ (where $n{=}4^k$).

*Induction step:* Using the induction hypothesis we prove the claim for $4n = 4^{k+1}$:

$$\begin{aligned} T(4n) = T(4^{k+1}) &\le 4\cdot T(\tfrac{4^{k+1}}{4}) + \beta\cdot 4^{k+1} = 4\cdot T(4^k) + \beta\cdot 4^{k+1} \\ &\overset{\text{IH}}{\le} 4(\alpha 4^k + \beta 4^k k) + \beta\cdot 4^{k+1} = \alpha 4^{k+1} + \beta 4^{k+1}(k+1) = \alpha 4n + \beta 4n\log_4(4n). \end{aligned}$$

(c) From part (a) and (b) we learned that $T(n) \le \alpha n + \beta n\log_4 n$, for $n = 4^k$, $k\in\mathbb{N}$. Hence $T(n)\in\mathcal{O}(n\log n)$. Now let $n\in\mathbb{N}$ arbitrary. We choose $k\in\mathbb{N}$ such that $4^{k-1}\le n\le 4^k$ ($4^k$ is the power of 4 bigger than $n$, which is closest to $n$). Since $T(n)$ is monotonously increasing, we have

$$\begin{aligned} T(n) \le T(4^k) &\overset{(a),(b)}{=} \alpha 4^k + \beta 4^k k = 4\alpha 4^{k-1} + 4\beta 4^{k-1} + 4\beta 4^{k-1}(k-1) \\ &\overset{4^{k-1}\le n}{\le} 4\big((\alpha+\beta)n + (\beta n\log_4 n)\big) \in \mathcal{O}(n\log n) \end{aligned}$$

*Note:* An even faster solution would be to cite the Master Theorem for recurrences which immediately implies $T(n)\in\mathcal{O}(n\log n)$ for arbitrary $n\in\mathbb{N}$ due to $T(n)\le 4\cdot T(n/4)+\beta\cdot n$.

## Exercise 4: Master Theorem for Recurrences (5 Points)

Use the *Master Theorem* for recurrences, to fill the following table. That is, in each cell write $\Theta\big(g(n)\big)$, such that $T(n)\in\Theta\big(g(n)\big)$ for the given parameters $a, b, f(n)$. Assume $T(1)\in\Theta(1)$. Additionally, in each cell note the case you used (1st, 2nd or 3rd). We filled out one cell as an example.

***Note**: You loose $\frac{1}{2}$ points if the complexity class is wrong and another $\frac{1}{2}$ if the case is wrong.*

| $T(n){=}aT(\frac{n}{b}){+}f(n)$ | $a=1, b=2$ | $a=3, b=2$ | $a=b=4$ |
|---|---|---|---|
| $f(n)=1$ | | | |
| $f(n)=n^2$ | $\Theta(n^2)$, 2nd | | |
| $f(n)=n\log n$ | | | |

## Sample Solution

| $T(n){=}aT(\frac{n}{b}){+}f(n)$ | $a=1, b=2$ | $a=3, b=2$ | $a=b=4$ |
|---|---|---|---|
| $f(n)=1$ | $\Theta(\log n)$, 3rd | $\Theta\big(n^{\log_2 3}\big)$, 1st | $\Theta(n)$, 1st |
| $f(n)=n^2$ | $\Theta(n^2)$, 2nd | $\Theta(n^2)$, 2nd | $\Theta(n^2)$, 2nd |
| $f(n)=n\log n$ | $\Theta(n\log n)$, 2nd | $\Theta\big(n^{\log_2 3}\big)$, 1st | $\Theta(n\log^2 n)$, 3rd |

## Exercise 5: Almost Closest Pairs (4+7 Points)

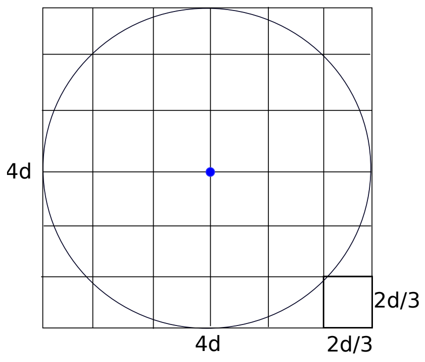
In the lecture, we discussed an $\mathcal{O}(n \log n)$-time divide-and-conquer algorithm to determine the closest pair of points among a set of $n$ points on the real plane $\mathbb{R}^2$. Assume that we are not only interested in the closest pair of points, but in all pairs of points that are at distance at most twice the distance between the closest two points.

(a) How many such pairs of points can there be? It suffices to give your answer using the $\mathcal{O}$-notation.

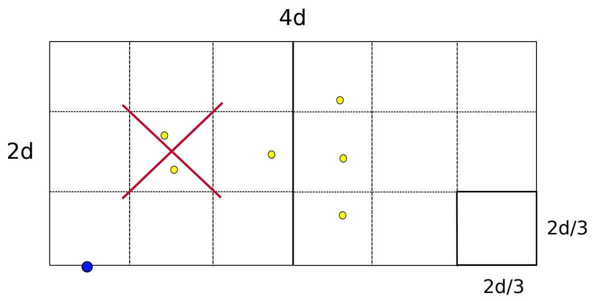
(b) Devise an algorithm that outputs a list with all pairs of points at distance at most twice the distance between the closest two points. Describe what you have to change compared to the closest pair algorithm of the lecture and analyze the running time of your algorithm.

## Sample Solution

The recursive algorithm for finding the closest pair of points, that was presented in the lecture, recursively divided the set of points on the plane into two sets and found the minimal distance as $\min\{d_\ell, d_r, d_{\ell r}\}$, where $d_\ell$ and $d_r$ are the minimal distances among the pairs of points in both sets and $d_{\ell r}$ is the minimal distance between pairs of points that lie in different sets. It was shown that finding $d_{\ell r}$ has linear complexity and the overall running time of the algorithm is $\mathcal{O}(n \log n)$.



(a) Positioning of points that are at distance at most $2d$ and at least at distance $d$ around the actual point we are checking.



(b) There is at most one point in each small square. We need to look only at most at 17 points.

Figure 1: Each small square can contain at most one point inside.

(a) Let us assume that the closest pair of points is already known and the distance between them is $d$. For each point $p$ we evaluate how many points there can be in distance $2d$. Figure 1a shows how the points at distance at most $2d$ from the center point can be covered using 36 squares with side length $2d/3$. As $2\sqrt{2}d/3 < d$ each such square can contain at most one point (including points on the boundary) and since the center point is part of 4 squares, the figure shows that a point can have at most 32 other points at distance at most $2d$. This way we count each pair twice and thus the number of pairs of points at distance at most $2d$ is at most $\frac{32}{2}n \in \mathcal{O}(n)$.

(b) Now, let us modify the divide-and-conquer algorithm from the lecture, in order to solve our task.

As in the closest pair algorithm of the lecture, after sorting points by their $x$-coordinate, we divide the set of points into a left subset $S_l$ and a right subset $S_r$ of equal size and we recursively find the smallest distance $d_\ell$ ($d_r$) in the left (right) half, as well as the list $L_\ell$ ($L_r$) that contains all pairs of points in $S_\ell$ ($S_r$) that are in distance at most $2d_\ell$ ($2d_r$) of each other.

For the merging step we find the smallest distance $d_{\ell r}$ between points that are on different sides of the bisection line. Additionally we compile a list $L_{\ell r}$ of pairs of points at distance at most

$2\min\{d_\ell, d_r\}$ such that the points lie on different sides. We compute $d = \min\{d_\ell, d_r, d_{\ell r}\}$, concatenate the three lists and remove all pairs $(p, q)$ for which $\text{dist}(p, q) > 2d$ from the combined list. Finally, we return $d$ and the list with all pairs of points at distance at most $2d$.

The following pseudocode provides an overview over the steps involved in the augmented algorithm.

**Algorithm 1** DoubleMinDist($S$) ▷ *returns* $(d, L, Y)$

```
▷ Note 1: We require $S$ to be sorted by x-coordinate to be able to split it in $\mathcal{O}(n)$.[a]
▷ Note 2: $d = \min_{u,v \in S} d(u,v)$, $L = \{(u,v) \mid u,v \in S, d(u,v) \le 2d\}$, $Y =$ "S sorted by y-coordinate"

if $|S| < 4$ then                                                    ▷ Base case, $\mathcal{O}(1)$
    Use the naive Algorithm (check all pairs of nodes) to calculate $(d, L, Y)$
    return $(d, L, Y)$

Divide $S$ equally into $S_\ell$ and $S_r$ along a vertical bisection line $b$    ▷ "Divide", $\mathcal{O}(n)$

$(d_\ell, L_\ell, Y_\ell) \leftarrow$ DoubleMinDist($S_\ell$)                         ▷ "Conquer", $2 \cdot T(n/2)$
$(d_r, L_r, Y_r) \leftarrow$ DoubleMinDist($S_r$)
                                                                     ▷ "Merge", $\mathcal{O}(n)$
$d \leftarrow \min(d_\ell, d_r)$                      ▷ Calculate current minimum distance, $\mathcal{O}(1)$
$Y \leftarrow$ MergeLists($Y_\ell, Y_r$)          ▷ Merge sorted lists into one, $\mathcal{O}(n)$ (cf. Mergesort)
$(d_{\ell r}, L_{\ell r}) \leftarrow$ DoubleMinDistMerge($Y, d, b$)                     ▷ Pseudocode below
    ▷ Determines $d_{\ell r} = \min_{u \in Y_\ell, v \in Y_r} d(u,v)$ and $L_{\ell r} = \{(u,v) \mid u \in Y_\ell, v \in Y_r, d(u,v) \le 2d\}$ in $\mathcal{O}(n)$
$d \leftarrow \min(d, d_{\ell r})$                       ▷ Calculate new minimum distance, $\mathcal{O}(1)$
$L \leftarrow L_\ell \cup L_r \cup L_{\ell r}$                                  ▷ Concatenate lists, $\mathcal{O}(n)$
$L \leftarrow L \setminus \{(u,v) \mid \text{dist}(u,v) > 2d\}$    ▷ Remove pairs violating new minimum distance 2d, $\mathcal{O}(n)$
return $(d, L, Y)$
```

[a] Alternatively, we can use the algorithm "median of medians" to find a decent pivot (i.e. a point close to the median) in an unsorted list $S$ in $\mathcal{O}(n)$ along which we can split it in two (again unsorted) lists $S_\ell$, $S_r$ in $\mathcal{O}(n)$, while retaining the claimed runtime.

Let us take a closer look at the merge step DoubleMinDistMerge($Y_\ell, Y_r$) of the algorithm (pseudocode is provided further below). Our goal is to compute $d_{\ell r}$ and $L_{\ell r}$ in $\mathcal{O}(n)$. Just as in the original algorithm we apply a procedure commonly called 'sweep-line'-approach, which iterates the points (in this context also called events) in $Y = Y_\ell \cup Y_r$ in ascending order by y-coordinate. By doing so we determine $d_{\ell r}$ by 'looking ahead' in $Y$, to find those pairs in question. Additionally we keep track of all pairs from opposite parts of the bisection line which have distance $\le 2d$.

The idea behind the correctness of algorithm DoubleMinDistMerge is that for each point $e$ within distance $2d$ of the bisection line $b$, we only have to look at points in $Y$ whose difference with respect to the $y$-coordinate is at most $2d$. This means when we iterate $Y$ to find a point close to $e$, then as soon as we find a point farther than $2d$ from $e$ in terms of $y$-coordinate (or if we are at the end of $Y$) we can stop looking ahead. Moreover, we only look at "partners" $p$ for $e$ with a bigger $y$-coordinate, because the other pairs were already found during previous "events".

Just as important as correctness is the running time of DoubleMinDistMerge, which we claim is $\mathcal{O}(n)$. This is not so clear, since at each event $e$, we may have to iterate over many points in $Y$, possible forcing us to iterate to the very end of $Y$ thus giving us a runtime of $\mathcal{O}(n)$ just to handle one event. If this could happen at every event $e$, we could have a runtime of $\mathcal{O}(n^2)$! However we argue that on average this does not happen too often, which means that we can bound the total number of "look aheads" separately.

We do this by bounding the number of times a point $p$ in $Y$ is being looked at by a prior event $e$. We realize that any point $p$ in $Y$ can have at most 9 points $e$ on the other side of $b$ which are within distance $2d$ of $b$ and have a $y$-coordinate which is at most $2d$ *smaller* than that of $p$. Figure 1b illustrates this by dividing a rectangle of size $2d \times 4d$ centered at $b$ into squares $\frac{2d}{3} \times \frac{2d}{3}$ (the idea is similar to (a)). This means that each point $p$ in $Y$ gets looked up at most 9 times, since $p$ can not get looked up by events outside a 2d strip around $b$ because those were discarded in the first place.

Therefore the $n$ points in $Y$ are being "touched" at most $9n = \mathcal{O}(n)$ times in total during the look ahead process of *all* events. Besides that, all other operations which we execute during an event require only constant time, thus adding only an additional $\mathcal{O}(n)$, which means that the whole merging process can be done in $\mathcal{O}(n)$. Putting all together we get the following recurrence relation for DoubleMinDist

$$T(n) \in \mathcal{O}(1), \text{ for } n < 4, \text{ and } T(n) \in 2 \cdot T(\frac{n}{2}) + \mathcal{O}(n), n \geq 4$$

which solves to $\mathcal{O}(n \log n)$ with the Master Theorem.

*Remark: The analysis becomes significantly easier if we change the algorithm to prefilter $Y$ so it only contains points within the $2d$ strip around the bisection line $b$ (in time $\mathcal{O}(n)$). Specifically, we can get rid of the amortization argument.*

**Algorithm 2** DoubleMinDistMerge($Y, d, b$) ▷ *returns* $(d_{\ell r}, L_{\ell r})$

```
▷ Note 1: $d = \min_{(u,v) \in Y_\ell^2 \cup Y_r^2} d(u,v)$, $b$ = bisection line (basically a x-coordinate)
▷ Note 2: $d_{\ell r} = \min_{u \in Y_\ell, v \in Y_r} d(u,v)$ and $L_{\ell r} = \{(u,v) \mid u \in Y_\ell, v \in Y_r, d(u,v) \leq 2d\}$
▷ Note 3: Assume that $Y = Y_\ell \cup Y_r$ supports classical list operations and its elements represent points. Additionally $Y$ is sorted by y-coordinate with the 'lowest' point at the head of the list

$d_{\ell r} \leftarrow d, L_{\ell r} \leftarrow \emptyset$                                    ▷ Initialization

while $|Y| > 1$ do
    $e \leftarrow Y$.PopHead      ▷ Select point (event) which is within a $2d$ strip around bisection line $b$
    while $e$.Point farther than $2d$ from bisection line $b$ do
        if $|Y| = 1$ then return $(d_{\ell r}, L_{\ell r})$
        $e \leftarrow Y$.PopHead

    $p \leftarrow Y$.Head                        ▷ Define pointer that iterates through list-elements of $Y$
    while $p$.Point.YCoord − $e$.Point.YCoord $\leq 2d$ and $p$.HasNext do
        if $p$.Point on same side of $b$ as $e$.Point or $p$.Point farther than $2d$ from $b$ then
            $p \leftarrow p$.Next                     ▷ Move pointer to next list element and continue
        else if $d(p.\text{Point}, e.\text{Point}) \leq 2d$ then
            $d_{\ell r} \leftarrow \min(d_{\ell r}, d(p.\text{Point}, e.\text{Point}))$                ▷ Update $d_{\ell r}$
            $L_{\ell r} \leftarrow L_{\ell r} \cup \{(p.\text{Point}, e.\text{Point})\}$      ▷ Add new pair of points $(p.\text{Point}, e.\text{Point})$
return $(d_{\ell r}, L_{\ell r})$
```

*Remark:* PopHead *returns and removes the head of a list.* Head *returns the head of a list (without removing it).* Next *returns the successor of a list element.* HasNext *returns true if the list element has a successor, else false.* Point *returns the point a list element represents.* YCoord *returns the y-coordinate of a point.*