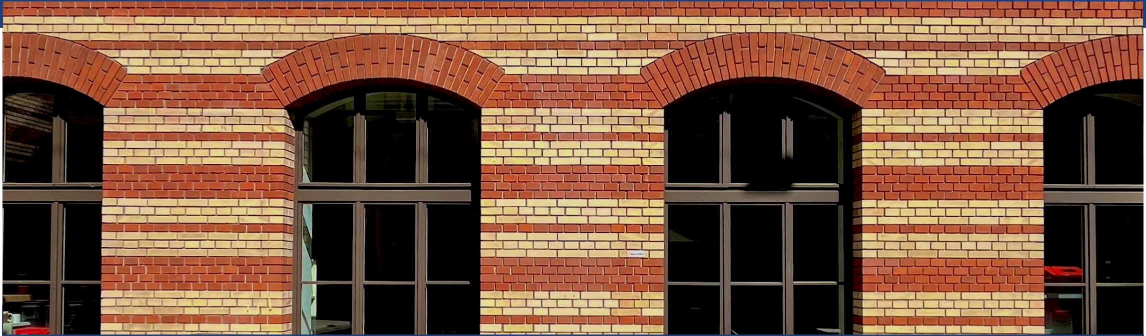


Datastructures and Algorithms

Recurrence Equations, Induction, Master Method, Runtime Analysis

Adel Gavranović — ETH Zürich — 2025

Overview

- Learning Objectives
- Landau Notation Quiz
- Analyse the running time of (recursive) Functions
- Solving Simple Recurrence Equations
- Sorting Algorithms
  - Quiz
  - Stable and In-Situ Sorting Algorithms
- In-Class Code-Examples
- Past Exam Questions
- Tips for **code** expert

`n.ethz.ch/~agavranovic`

Material

Webpage

Mail

1. Follow-up

Follow-up from last session

Slide 18 "Motivational Example"

Follow-up from last session

Slide 18 "Motivational Example"

- I have relayed the feedback regarding the missing definition for +

Slide 53 "Altklausur 2020: Aufgabe 2a) — Solution"

Follow-up from last session

Slide 18 "Motivational Example"

- I have relayed the feedback regarding the missing definition for +

Slide 53 "Altklausur 2020: Aufgabe 2a) — Solution"

- Due to time constraints, we're not going to go over this exam question again, but you're probably going to be able to solve it after this session

2. Feedback regarding **code** expert

General things regarding **code** expert

General things regarding **code** expert

- If you want feedback for Code, please make sure to mention it at the very top of the code with "`FEEDBACK PLEASE`" (or similar)

General things regarding code expert

- If you want feedback for Code, please make sure to mention it at the very top of the code with "`FEEDBACK PLEASE`" (or similar)
- I can't recommend this enough: Check out the reference solution each week and double check your understanding

General things regarding code expert

- If you want feedback for Code, please make sure to mention it at the very top of the code with "`FEEDBACK PLEASE`" (or similar)
- I can't recommend this enough: Check out the reference solution each week and double check your understanding
- If I ever seem needlessly strict (do tell me!), It's only because I really want you all to pass the exam (well)

Big-O-Notation

Specific things regarding code expert

Big-O-Notation

- You might've seen in the lectures: for Landau-notation it doesn't matter if you write $\log_2$ or any other base ($\log_b$) since they're asymptotically equivalent! (thus we usually just write $\log$ with no specified base)

Specific things regarding code expert

Big-O-Notation

- You might've seen in the lectures: for Landau-notation it doesn't matter if you write $\log_2$ or any other base ($\log_b$) since they're asymptotically equivalent! (thus we usually just write $\log$ with no specified base)

Asymptotic Growth

Specific things regarding code expert

Big-O-Notation

- You might've seen in the lectures: for Landau-notation it doesn't matter if you write $\log_2$ or any other base ($\log_b$) since they're asymptotically equivalent! (thus we usually just write $\log$ with no specified base)

Asymptotic Growth

- Ideally, you'd have a ranking on your cheat sheet (or know it by heart) and then you just apply some logic and analysis to determine a ranking for some given asymptotic complexities

Any questions regarding **code expert** on your part?

→ Prefix sum in 2D:

terrible description since

it only mentions 1 function!

(no word about the other)

3. Learning Objectives

Objectives

- ☐ Be able to solve "rank-by-complexity" tasks
- ☐ Be able to set up *recurrence equations* from Code Snippets
- ☐ Be able to solve *recurrence equations* and solution’s correctness

4. Summary

Getting on the same page

Landau Notation

- Give a correct definition of the set $\Theta(f)$ as compact as possible analogously to the definitions for sets $\mathcal{O}(f)$ and $\Omega(f)$.

Landau Notation

- Give a correct definition of the set $\Theta(f)$ as compact as possible analogously to the definitions for sets $\mathcal{O}(f)$ and $\Omega(f)$.
- $\Theta(f) = \{g : \mathbb{N} \to \mathbb{R} \mid \exists a > 0,\ b > 0,\ n_0 \in \mathbb{N} : \underline{a \cdot f(n)} \leq \underline{g(n)} \leq \cancel{b} \cdot f(n)\ \forall n \geq n_0\}$

$\underbrace{\frac{1}{a}}$

Landau Notation

- Give a correct definition of the set $\Theta(f)$ as compact as possible analogously to the definitions for sets $\mathcal{O}(f)$ and $\Omega(f)$.
- $\Theta(f) = \{g : \mathbb{N} \to \mathbb{R} \mid \exists a > 0,\ b > 0,\ n_0 \in \mathbb{N} : a \cdot f(n) \leq g(n) \leq b \cdot f(n)\ \forall n \geq n_0\}$
- $\Theta(f) = \{g : \mathbb{N} \to \mathbb{R} \mid \exists c > 0,\ n_0 \in \mathbb{N} : \frac{1}{c} \cdot f(n) \leq g(n) \leq c \cdot f(n)\ \forall n \geq n_0\}$

$\leftarrow$

Landau Notation

Prove or disprove the following statements, where $f, g : \mathbb{N} \to \mathbb{R}^+$.

✓(a) $f \in \mathcal{O}(g)$ if and only if $g \in \Omega(f)$.

✓(e) $\log_a(n) \in \Theta(\log_b(n))$ for all constants $a, b \in \mathbb{N} \setminus \{1\}$ $\Theta(\log(n))$

✗(g) If $f_1, f_2 \in \mathcal{O}(g)$ and $f(n) := f_1(n) \cdot f_2(n)$, then $f \in \mathcal{O}(g)$.

$\mathcal{O}(n)$

$f_1 = n$
$f_2 = 3n$ $\rightarrow f = f_1 \cdot f_2 = 3n^2 \notin \mathcal{O}(g)$

$g(n) = n$
$f_1, f_2 = 1$
$f_2 = n \in \mathcal{O}(n)$

Landau Notation

Sorting functions: if function f is left to function g, then $f \in \mathcal{O}(g)$.

Reorder the following:
$n^5 + n,\ \log(n^4),\ \sqrt{n},\ \binom{n}{3},\ 2^{16},\ n^n,\ n!,\ \frac{2^n}{n^2},\ \log^8(n),\ n \log n$

Landau Notation

Sorting functions: if function f is left to function g, then $f \in \mathcal{O}(g)$.

Reorder the following:

$n^5 + n,\ \log(n^4),\ \sqrt{n},\ \binom{n}{3},\ 2^{16},\ n^n,\ n!,\ \frac{2^n}{n^2},\ \log^8(n),\ n \log n$

$n^{\frac{1}{2}}$

Solution: $2^{16},\ \log(n^4),\ \log^8(n),\ \sqrt{n},\ n\log n,\ \binom{n}{3},\ n^5 + n,\ \frac{2^n}{n^2},\ n!,\ n^n$.

$4\log(n) < (\log(n))^8$

$n \cdot \log(n)$

What I had on my Cheatsheet

[1] there's a mistake here!

What I had on my Cheatsheet

for $c \in \mathbb{R}^+$:[1]

$c = \frac{1}{2}$

$n^{\frac{1}{2}} = \sqrt{n}$

$$c,\ \log\log n,\ \log^c n,\ \sqrt{n},\ n,\ n\log n,\ n^c,\ c^n,\ n!,\ n^n$$

$$\binom{n}{k} = \frac{n!}{k!\cdot(n-k)!} \in \Theta(n^k),\quad \log(n!) \in \Theta(n\log n)$$

$\Theta(\log n^n$

[1] there's a mistake here!

My personal approach to solving them

1. Have the "ranking" on my cheatsheet

My personal approach to solving them

1. Have the "ranking" on my cheatsheet
2. Move all entries with exponents dependend on n to the right

My personal approach to solving them

1. Have the "ranking" on my cheatsheet
2. Move all entries with exponents dependend on n to the right
3. Constants (no matter how large) all the way to the left

My personal approach to solving them

1. Have the "ranking" on my cheatsheet
2. Move all entries with exponents dependend on n to the right
3. Constants (no matter how large) all the way to the left
4. All "obviously $\log$"-things rather to the left

My personal approach to solving them

1. Have the "ranking" on my cheatsheet
2. Move all entries with exponents dependend on n to the right
3. Constants (no matter how large) all the way to the left
4. All "obviously $\log$"-things rather to the left
5. Resolve/rewrite binomial stuff to polynomials

My personal approach to solving them

1. Have the "ranking" on my cheatsheet
2. Move all entries with exponents dependend on n to the right
3. Constants (no matter how large) all the way to the left
4. All "obviously $\log$"-things rather to the left
5. Resolve/rewrite binomial stuff to polynomials
6. Do not forget that $\sqrt{n} = n^{\frac{1}{2}}$

My personal approach to solving them

1. Have the "ranking" on my cheatsheet
2. Move all entries with exponents dependend on n to the right
3. Constants (no matter how large) all the way to the left
4. All "obviously $\log$"-things rather to the left
5. Resolve/rewrite binomial stuff to polynomials
6. Do not forget that $\sqrt{n} = n^{\frac{1}{2}}$
7. All obvious polynomial-in-n things rather to the right

My personal approach to solving them

1. Have the "ranking" on my cheatsheet
2. Move all entries with exponents dependend on n to the right
3. Constants (no matter how large) all the way to the left
4. All "obviously $\log$"-things rather to the left
5. Resolve/rewrite binomial stuff to polynomials
6. Do not forget that $\sqrt{n} = n^{\frac{1}{2}}$
7. All obvious polynomial-in-n things rather to the right
8. Where it’s not obvious:
    - Switch on your brain and make comparisons

My personal approach to solving them

1. Have the "ranking" on my cheatsheet
2. Move all entries with exponents dependend on n to the right
3. Constants (no matter how large) all the way to the left
4. All "obviously $\log$"-things rather to the left
5. Resolve/rewrite binomial stuff to polynomials
6. Do not forget that $\sqrt{n} = n^{\frac{1}{2}}$
7. All obvious polynomial-in-n things rather to the right
8. Where it's not obvious:
   - Switch on your brain and make comparisons
   - (Analysis I was actually useful!)

5. Landau Notation Quiz

Landau Notation Quiz

Is $f \in \mathcal{O}(n^2)$, if $f(n) = \ldots$?

$\leftarrow$

Landau Notation Quiz

Is $f \in \mathcal{O}(n^2)$, if $f(n) = \dots$?

- n

Landau Notation Quiz

Is $f \in \mathcal{O}(n^2)$, if $f(n) = \ldots$?

- n ✓
- $n^2 + 1$

Landau Notation Quiz

Is $f \in \mathcal{O}(n^2)$, if $f(n) = \ldots$?

- n ✓
- $n^2 + 1$ ✓
- $\log^4(n^2)$

$\leftarrow$

Landau Notation Quiz

Is $f \in \mathcal{O}(n^2)$, if $f(n) = \ldots$?

- n ✓
- $n^2 + 1$ ✓
- $\log^4(n^2)$ ✓
- $n \log(n^2)$ $= \cancel{2} n \log(n) \overset{<}{\in} n \cdot n \quad n^2$

Landau Notation Quiz

Is $f \in \mathcal{O}(n^2)$, if $f(n) = \ldots$?

- n ✓
- $n^2 + 1$ ✓
- $\log^4(n^2)$ ✓
- $n \log(n^2)$ ✓
- n^π

Landau Notation Quiz

Is $f \in \mathcal{O}(n^2)$, if $f(n) = \ldots$?

- n ✓
- $n^2 + 1$ ✓
- $\log^4(n^2)$ ✓
- $n \log(n^2)$ ✓
- n^π ✗ $(\pi \approx 3.14 > 2)$

←

Landau Notation Quiz

Is $f \in \mathcal{O}(n^2)$, if $f(n) = \ldots$?

- n ✓
- $n^2 + 1$ ✓
- $\log^4(n^2)$ ✓
- $n \log(n^2)$ ✓
- n^π ✗ $(\pi \approx 3.14 > 2)$
- $n \cdot 2^{16}$

Landau Notation Quiz

Is $f \in \mathcal{O}(n^2)$, if $f(n) = \ldots$?

- n ✓
- $n^2 + 1$ ✓
- $\log^4(n^2)$ ✓
- $n \log(n^2)$ ✓
- n^π ✗ $(\pi \approx 3.14 > 2)$
- $n \cdot 2^{16}$ ✓
- $n^2 \cdot 2^{16}$

Landau Notation Quiz

Is $f \in \mathcal{O}(n^2)$, if $f(n) = \ldots$?

- n ✓
- $n^2 + 1$ ✓
- $\log^4(n^2)$ ✓
- $n \log(n^2)$ ✓
- n^π ✗ $(\pi \approx 3.14 > 2)$
- $n \cdot 2^{16}$ ✓
- $n^2 \cdot 2^{16}$ ✓
- 2^n

Landau Notation Quiz

Is $f \in \mathcal{O}(n^2)$, if $f(n) = \ldots$?

- n ✓
- $n^2 + 1$ ✓
- $\log^4(n^2)$ ✓
- $n \log(n^2)$ ✓
- n^π ✗ $(\pi \approx 3.14 > 2)$
- $n \cdot 2^{16}$ ✓
- $n^2 \cdot 2^{16}$ ✓
- 2^n ✗

Landau Notation Quiz

Is $f \in \mathcal{O}(n^2)$, if $f(n) = \ldots ?$

- n ✓
- $n^2 + 1$ ✓
- $\log^4(n^2)$ ✓
- $n \log(n^2)$ ✓
- n^π ✗ $(\pi \approx 3.14 > 2)$
- $n \cdot 2^{16}$ ✓
- $n^2 \cdot 2^{16}$ ✓
- 2^n ✗

Is $g \in \Omega(2n)$, if $g(n) = \ldots ?$

$\leftarrow$

Landau Notation Quiz

Is $f \in \mathcal{O}(n^2)$, if $f(n) = \ldots$?

- n ✓
- $n^2 + 1$ ✓
- $\log^4(n^2)$ ✓
- $n \log(n^2)$ ✓
- n^π ✗ $(\pi \approx 3.14 > 2)$
- $n \cdot 2^{16}$ ✓
- $n^2 \cdot 2^{16}$ ✓
- 2^n ✗

Is $g \in \Omega(2n)$, if $g(n) = \ldots$?

- 1

Landau Notation Quiz

Is $f \in \mathcal{O}(n^2)$, if $f(n) = \ldots$?

- n ✓
- $n^2 + 1$ ✓
- $\log^4(n^2)$ ✓
- $n \log(n^2)$ ✓
- n^π ✗ $(\pi \approx 3.14 > 2)$
- $n \cdot 2^{16}$ ✓
- $n^2 \cdot 2^{16}$ ✓
- 2^n ✗

Is $g \in \Omega(2n)$, if $g(n) = \ldots$?

- 1 ✗

Landau Notation Quiz

Is $f \in \mathcal{O}(n^2)$, if $f(n) = \dots$?

- n ✓
- $n^2 + 1$ ✓
- $\log^4(n^2)$ ✓
- $n \log(n^2)$ ✓
- n^π ✗ $(\pi \approx 3.14 > 2)$
- $n \cdot 2^{16}$ ✓
- $n^2 \cdot 2^{16}$ ✓
- 2^n ✗

Is $g \in \Omega(2n)$, if $g(n) = \dots$?

- 1 ✗
- n

←

Landau Notation Quiz

Is $f \in \mathcal{O}(n^2)$, if $f(n) = \ldots$?

- n ✓
- $n^2 + 1$ ✓
- $\log^4(n^2)$ ✓
- $n \log(n^2)$ ✓
- n^π ✗ $(\pi \approx 3.14 > 2)$
- $n \cdot 2^{16}$ ✓
- $n^2 \cdot 2^{16}$ ✓
- 2^n ✗

Is $g \in \Omega(2n)$, if $g(n) = \ldots$?

- 1 ✗
- n ✓
- $\pi \cdot n$

Landau Notation Quiz

Is $f \in \mathcal{O}(n^2)$, if $f(n) = \ldots$?

- n ✓
- $n^2 + 1$ ✓
- $\log^4(n^2)$ ✓
- $n \log(n^2)$ ✓
- n^π ✗ $(\pi \approx 3.14 > 2)$
- $n \cdot 2^{16}$ ✓
- $n^2 \cdot 2^{16}$ ✓
- 2^n ✗

Is $g \in \Omega(2n)$, if $g(n) = \ldots$?

- 1 ✗
- n ✓
- $\pi \cdot n$ ✓
- $\pi^{42} \cdot n$

←

Landau Notation Quiz

Is $f \in \mathcal{O}(n^2)$, if $f(n) = \ldots$?

- n ✓
- $n^2 + 1$ ✓
- $\log^4(n^2)$ ✓
- $n \log(n^2)$ ✓
- n^π ✗ $(\pi \approx 3.14 > 2)$
- $n \cdot 2^{16}$ ✓
- $n^2 \cdot 2^{16}$ ✓
- 2^n ✗

Is $g \in \Omega(2n)$, if $g(n) = \ldots$?

- 1 ✗
- n ✓
- $\pi \cdot n$ ✓
- $\pi^{42} \cdot n$ ✓
- $\log(n)$

Landau Notation Quiz

Is $f \in \mathcal{O}(n^2)$, if $f(n) = \ldots$?

- n ✓
- $n^2 + 1$ ✓
- $\log^4(n^2)$ ✓
- $n \log(n^2)$ ✓
- n^π ✗ $(\pi \approx 3.14 > 2)$
- $n \cdot 2^{16}$ ✓
- $n^2 \cdot 2^{16}$ ✓
- 2^n ✗

Is $g \in \Omega(2n)$, if $g(n) = \ldots$?

- 1 ✗
- n ✓
- $\pi \cdot n$ ✓
- $\pi^{42} \cdot n$ ✓
- $\log(n)$ ✗
- $\sqrt{n}$

Landau Notation Quiz

Is $f \in \mathcal{O}(n^2)$, if $f(n) = \ldots$?

- n ✓
- $n^2 + 1$ ✓
- $\log^4(n^2)$ ✓
- $n \log(n^2)$ ✓
- n^π ✗ $(\pi \approx 3.14 > 2)$
- $n \cdot 2^{16}$ ✓
- $n^2 \cdot 2^{16}$ ✓
- 2^n ✗

Is $g \in \Omega(2n)$, if $g(n) = \ldots$?

- 1 ✗
- n ✓
- $\pi \cdot n$ ✓
- $\pi^{42} \cdot n$ ✓
- $\log(n)$ ✗
- $\sqrt{n}$ ✗

6. Analyse the running time of (recursive) Functions

$g(n)$

How many calls to `f()`?

```
for(unsigned i = 1; i <= n/3; i += 3)
  for(unsigned j = 1; j <= i; ++j)
    f();
```

$g(n)$

void g(int n){

,

}

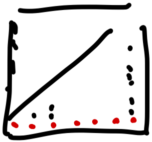
$\frac{n}{3} \cdot n \cdot 1 = \cdots n^2$

Analysis

How many calls to `f()`?

```
for(unsigned i = 1; i <= n/3; i += 3)
  for(unsigned j = 1; j <= i; ++j)
    f();
```

$\frac{1}{2}\left(\frac{1}{9}n^2\right) +$



$\frac{n}{9}$

The code fragment implies $\Theta(n^2)$ calls to `f()`: the outer loop is executed $n/9$ times and the inner loop contains i calls to `f()`

How many calls to f()?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

$\mathcal{O}(n)$

$\mathcal{O}(1)$

$\mathcal{O}(\log(n))$

"$\mathcal{O}(n) \cdot \left(\mathcal{O}(1) + \mathcal{O}(\log(n))\right)$"

$\rightarrow \mathcal{O}(n \log(n))$

How many calls to `f()`?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

We can ignore the first inner loop because it contains only a constant number of calls to `f()`

←

How many calls to `f()`?

```
for(unsigned i = 0; i < n; ++i) {
  for(unsigned j = 100; j*j >= 1; --j)
    f();
  for(unsigned k = 1; k <= n; k *= 2)
    f();
}
```

We can ignore the first inner loop because it contains only a constant number of calls to `f()`

The second inner loop contains $\lfloor \log_2(n) \rfloor + 1$ calls to `f()`. Summing up yields $\Theta(n \log(n))$ calls.

How many calls to `f()`?

$T(n)$: # calls to f()

```
void g(unsigned n) {
  if (n>0){
    g(n-1);
    f();
  }
}
```

$$T(n) = T(n-1) + 1$$

$$= T(n-2) + 1 + 1$$

sub.

$$= T(n-2) + 2$$

clean up.

$$\vdots$$

$$= T(0) + \dots$$

$$= 0 + n$$

$$\in \Theta(n)$$

n

How many calls to `f()`?

```
void g(unsigned n) {
  if (n>0){
    g(n-1);
    f();
  }
}
```

$$M(n) = M(n-1) + 1 = M(n-2) + 2 = \ldots = M(0) + n = n \in \Theta(n)$$

How many calls to `f()`?

$$\frac{2}{2} = 1$$

```
// pre: n is a power of 2
//      n = 2^k
void g(int n){
  if (n>0){
    g(n/2);
    f();
  }
}
```

n=0

$\log_2(n) = k$

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

$$= T\left(\frac{\frac{n}{2}}{2}\right) + 1 + 1$$

$$= T\left(\frac{n}{4}\right) + 2$$

$$= T(0) + \boxed{k+1} \quad \| \; k = \log_2 n$$

$\log(n) + 1 = k+1$

$$= 0 + \log_2(n) + 1$$

$$\in \Theta(\log(n))$$

How many calls to `f()`?

```
// pre: n is a power of 2
//      n = 2^k
void g(int n){
  if (n>0){
    g(n/2);
    f()
  }
}
```

$$M(n) = 1 + M(n/2) = 1 + 1 + M(n/4) = k + M(n/2^k) \in \Theta(\log n)$$

How many calls to `f()`?

```
// pre: n is a power of 2 (n = 2^k)
void g(int n){
  if (n>0){
    f();
    g(n/2);
    f();
    g(n/2);
  }
}
```

rec. eq. → Master Method

proof by Ind.

$$M(2^k) = \begin{cases} 2 + 2M(2^{k-1}) & k \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$M(2^k) =$

"closed form"

Lo proof

How many calls to `f()`?

```
// pre: n is a power of 2 (n = 2^k)
void g(int n){
  if (n>0){
    f();
    g(n/2);
    f();
    g(n/2);
  }
}
```

$$M(2^k) = \begin{cases} 2 + 2M(2^{k-1}) & k \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{aligned} M(n) &= 2M\left(\frac{n}{2}\right) + 2 = 4M\left(\frac{n}{4}\right) + 4 + 2 = 8M\left(\frac{n}{8}\right) + 8 + 4 + 2 \\ &= 2(n + n/2 + n/4 + \ldots + 1) \in \Theta(n) \end{aligned}$$

How many calls to `f()`?

```
// pre: n is a power of 2
//      n = 2^k
void g(int n){
  if (n>0){
    g(n/2);
    g(n/2);
  }
  for (int i = 0; i < n; ++i){
    f();
  }
}
```

How many calls to `f()`?

```
// pre: n is a power of 2
//      n = 2^k
void g(int n){
  if (n>0){
    g(n/2);
    g(n/2);
  }
  for (int i = 0; i < n; ++i){
    f();
  }
}
```

$$M(n) = 2M(n/2) + n = 4M(n/4) + n + 2n/2 = \ldots = (k+1)n \in \Theta(n \log n)$$

How many calls to f()?

```
void g(unsigned n) {
  for (unsigned i = 0; i<n ; ++i) {
    g(i)
  }
  f();
}
```

How many calls to `f()`?

```
void g(unsigned n) {
  for (unsigned i = 0; i<n ; ++i) {
    g(i)
  }
  f();
}
```

$\boxed{T(0) = 1}$

How many calls to `f()`?

```
void g(unsigned n) {
  for (unsigned i = 0; i<n ; ++i) {
    g(i)
  }
  f();
}
```

$$T(0) = 1$$
$$T(n) = 1 + \sum_{i=0}^{n-1} T(i)$$

←

How many calls to `f()`?

```
void g(unsigned n) {
  for (unsigned i = 0; i<n ; ++i) {
    g(i)
  }
  f();
}
```

$T(0) = 1$

$T(n) = 1 + \sum_{i=0}^{n-1} T(i)$

n	0	1	2	3	4
$T(n)$	1	2	4	8	16

←

How many calls to `f()`?

```
void g(unsigned n) {
  for (unsigned i = 0; i<n ; ++i) {
    g(i)
  }
  f();
}
```

$T(0) = 1$

$T(n) = 1 + \sum_{i=0}^{n-1} T(i)$

n	0	1	2	3	4
$T(n)$	1	2	4	8	16

Hypothesis: $T(n) = 2^n$.

←

Induction

Hypothesis: $T(n) = 2^n$.
Induction step:

$$
\begin{aligned}
T(n) &= 1 + \sum_{i=0}^{n-1} 2^i \\
&= 1 + 2^n - 1 = 2^n
\end{aligned}
$$

Induction

Hypothesis: $T(n) = 2^n$.
Induction step:

$$\begin{aligned} T(n) &= 1 + \sum_{i=0}^{n-1} 2^i \\ &= 1 + 2^n - 1 = 2^n \end{aligned}$$

How many calls to `f()`?

```
void g(unsigned n) {
  for (unsigned i = 0; i<n ; ++i) {
    g(i)
  }
  f();
}
```

You can also see it directly:

$$T(n) = 1 + \sum_{i=0}^{n-1} T(i)$$

$$\Rightarrow T(n-1) = 1 + \sum_{i=0}^{n-2} T(i)$$

$$\Rightarrow T(n) = T(n-1) + T(n-1) = 2T(n-1)$$

7. Solving Simple Recurrence Equations

Recurrence Equation

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + \frac{n}{2} + 1, & n > 1 \\ 3 & n = 1 \end{cases}$$

Specify a closed (non-recursive), simple formula for $T(n)$ and prove it using mathematical induction. Assume that n is a power of 2.

Recurrence Equation

$$
\begin{aligned}
T(2^k) &= 2T(2^{k-1}) + 2^k/2 + 1 \\
&= 2(2(T(2^{k-2}) + 2^{k-1}/2 + 1) + 2^k/2 + 1 = \ldots \\
&= 2^k T(2^{k-k}) + \underbrace{2^k/2 + \ldots + 2^k/2}_{k} + 1 + 2 + \ldots + 2^{k-1} \\
&= 3n + \frac{n}{2}\log_2 n + n - 1
\end{aligned}
$$

$\Rightarrow$ Assumption $T(n) = 4n + \frac{n}{2}\log_2 n - 1$

$\leftarrow$

Induction

1. Hypothesis $T(n) = f(n) := 4n + \frac{n}{2}\log_2 n - 1$
2. Base Case $T(1) = 3 = f(1) = 4 - 1$.
3. Step $T(n) = f(n) \longrightarrow T(2 \cdot n) = f(2n)$ ($n = 2^k$ for some $k \in \mathbb{N}$):

$$\begin{aligned}
T(2n) &= 2T(n) + n + 1 \\
&\overset{i.h.}{=} 2(4n + \frac{n}{2}\log_2 n - 1) + n + 1 \\
&= 8n + n\log_2 n - 2 + n + 1 \\
&= 8n + n\log_2 n + n\log_2 2 - 1 \\
&= 8n + n\log_2 2n - 1 \\
&= f(2n).
\end{aligned}$$

Master Method

$$T(n) = \begin{cases} aT(\frac{n}{b}) + f(n) & n > 1 \\ f(1) & n = 1 \end{cases} \qquad (a, b \in \mathbb{N}^+)$$

1. $f(n) = \mathcal{O}(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0 \implies T(n) \in \Theta(n^{\log_b a})$
2. $f(n) = \Theta(n^{\log_b a}) \implies T(n) \in \Theta(n^{\log_b a} \log n)$
3. $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(\frac{n}{b}) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large $n \implies T(n) \in \Theta(f(n))$

Examples

Maximum Subarray / Mergesort

$a = 2$ $f = c \cdot n$

$$T(n) = 2T(n/2) + \Theta(n)$$

$b = 2$

Examples

Maximum Subarray / Mergesort

$$T(n) = 2T(n/2) + \Theta(n)$$

$$a = 2, b = 2, \; f(n) = cn = cn^1 = cn^{\log_2 2} \overset{[2]}{\Longrightarrow} T(n) = \Theta(n \log n)$$

Examples

Naive Matrix Multiplication Divide & Conquer[2]

$$T(n) = 8T(n/2) + \Theta(n^2)$$

[2] Treated in the course later on

Examples

Naive Matrix Multiplication Divide & Conquer[2]

$$T(n) = 8T(n/2) + \Theta(n^2)$$

$$a = 8, b = 2,\ f(n) = cn^2 \in \mathcal{O}(n^{\log_2 8 - 1}) \xRightarrow{[1]} T(n) \in \Theta(n^3)$$

[2]Treated in the course later on

Examples

Strassens Matrix Multiplication Divide & Conquer[3]

$$T(n) = 7T(n/2) + \Theta(n^2)$$

[3]Treated in the course later on

Examples

Strassens Matrix Multiplication Divide & Conquer[3]

$$T(n) = 7T(n/2) + \Theta(n^2)$$

$$a = 7, b = 2,\ f(n) = cn^2 \in \mathcal{O}(n^{\log_2 7 - \epsilon}) \overset{[1]}{\Longrightarrow} T(n) \in \Theta(n^{\log_2 7}) \approx \Theta(n^{2.8})$$

[3]Treated in the course later on

Examples

$$T(n) = 2T(n/4) + \Theta(n)$$

Examples

$$T(n) = 2T(n/4) + \Theta(n)$$

$$a = 2, b = 4,\ f(n) = cn \in \Omega(n^{\log_4 2 + 0.5}),\ 2f(n/4) = c\frac{n}{2} \le \frac{c}{2}n^1 \xrightarrow{[3]} T(n) \in \Theta(n)$$

$\leftarrow$

Examples

$a = 2$

$$T(n) = 2T(n/4) + \underbrace{\Theta(n^2)}_{f(n)}$$

$b = 4$

Examples

Snippet

↓

Rec. Eq.

reform.

$$\boxed{T(n) = 2T(n/4) + \Theta(n^2)}$$

$a = 2, b = 4,\ f(n) = cn^2 \in \Omega(n^{\log_4 2+1.5}),\ 2f(n/4) = \frac{n^2}{8} \leq \frac{1}{8}n^2 \xrightarrow{[3]}$
$T(n) \in \Theta(n^2)$

M.M.

What I had on my Cheatsheet

What I had on my Cheatsheet

Equation must be convertible into form

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n), \quad (a \geq 1, b > 1)$$

$\leftarrow$

What I had on my Cheatsheet

Equation must be convertible into form

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n), \quad (a \geq 1, b > 1)$$

where:

a : Number of Subproblems
$1/b$: Division Quotient
$f(n)$: Div- and Summing Costs

$\leftarrow$

What I had on my Cheatsheet

Equation must be convertible into form

$$\boxed{T(n) = a \cdot T\left(\frac{n}{b}\right) + \boxed{f(n)}, \quad (a \geq 1, b > 1)}$$

where:

a : Number of Subproblems
$1/b$: Division Quotient
$f(n)$: Div- and Summing Costs

Then we can proceed:

1. Convert the Recurrence Equation into the form above

What I had on my Cheatsheet

Equation must be convertible into form

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n), \quad (a \geq 1, b > 1)$$

where:

a : Number of Subproblems
$1/b$: Division Quotient
$f(n)$: Div- and Summing Costs

Then we can proceed:

1. Convert the Recurrence Equation into the form above
2. Calculate $\boxed{K} := \boxed{\log_b a}$

What I had on my Cheatsheet

Equation must be convertible into form

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n), \quad (a \geq 1, b > 1)$$

where:

a : Number of Subproblems
$1/b$: Division Quotient
$f(n)$: Div- and Summing Costs

Then we can proceed:

1. Convert the Recurrence Equation into the form above
2. Calculate $K := \log_b a$
3. Make case distinction $(\varepsilon > 0)$:

←

What I had on my Cheatsheet

Equation must be convertible into form

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n), \quad (a \geq 1, b > 1)$$

where:

a	: Number of Subproblems
$1/b$	: Division Quotient
$f(n)$	: Div- and Summing Costs

Then we can proceed:

1. Convert the Recurrence Equation into the form above
2. Calculate $K := \log_b a$
3. Make case distinction ($\varepsilon > 0$):

$K = 2$

$f(n) = n^2$

$$f \in \begin{cases} \mathcal{O}\left(n^{K-\varepsilon}\right) & \Longrightarrow \boxed{T(n) \in \Theta\left(n^K\right)} \\ \Theta\left(n^K\right) & \Longrightarrow \boxed{T(n)} \in \Theta\left(n^K \log(n)\right) \\ \Omega\left(n^{K+\varepsilon}\right) & \wedge\, af\left(\frac{n}{b}\right) \leq cf(n),\ 0 < c < 1 \\ & \Longrightarrow \boxed{T(n)} \in \Theta(f(n)) \end{cases}$$

Personal Approach to "Solving RecEqs"

"Plug and Chuck"-Approach

1. Expand few times
2. Notice patterns (careful with multiplications on $T(n)$)
3. Write down explicitly
4. Formulate explicit formula $f(n)$
5. Prove via induction

Personal Approach to "Calls of `f()`"

1. Loops: just multiply outer runtime with inner to get whole runtime (works recursively)
2. Just brute-force calculate $g(0), g(1), g(2), g(3), \ldots$ and try to identify trends
3. If too hard: consider $\Theta(2^n)$
4. If necessary/possible, simply set up and solve RecEqs via Master Method
5. If asked provide proof (by induction)

8. Sorting Algorithms

Quiz

Consider the following three sequences of snap-shots (steps) of the algorithms (a) Insertion Sort, (b) Selection Sort and (c) Bubblesort. Below each sequence provide the corresponding algorithm name.

5	4	1	3	2
1	4	5	3	2
1	2	5	3	4
1	2	3	5	4
1	2	3	4	5

5	4	1	3	2
4	1	3	2	5
1	3	2	4	5
1	2	3	4	5

5	4	1	3	2
4	5	1	3	2
1	4	5	3	2
1	3	4	5	2
1	2	3	4	5

Quiz

Consider the following three sequences of snap-shots (steps) of the algorithms (a) Insertion Sort, (b) Selection Sort and (c) Bubblesort. Below each sequence provide the corresponding algorithm name.

5	4	1	3	2
1	4	5	3	2
1	2	5	3	4
1	2	3	5	4
1	2	3	4	5

selection

5	4	1	3	2
4	1	3	2	5
1	3	2	4	5
1	2	3	4	5

5	4	1	3	2
4	5	1	3	2
1	4	5	3	2
1	3	4	5	2
1	2	3	4	5

Quiz

Consider the following three sequences of snap-shots (steps) of the algorithms (a) Insertion Sort, (b) Selection Sort and (c) Bubblesort. Below each sequence provide the corresponding algorithm name.

5	4	1	3	2
1	4	5	3	2
1	2	5	3	4
1	2	3	5	4
1	2	3	4	5

selection

5	4	1	3	2
4	1	3	2	5
1	3	2	4	5
1	2	3	4	5

bubblesort

5	4	1	3	2
4	5	1	3	2
1	4	5	3	2
1	3	4	5	2
1	2	3	4	5

Quiz

Consider the following three sequences of snap-shots (steps) of the algorithms (a) Insertion Sort, (b) Selection Sort and (c) Bubblesort. Below each sequence provide the corresponding algorithm name.

5	4	1	3	2
1	4	5	3	2
1	2	5	3	4
1	2	3	5	4
1	2	3	4	5

selection

5	4	1	3	2
4	1	3	2	5
1	3	2	4	5
1	2	3	4	5

bubblesort

5	4	1	3	2
4	5	1	3	2
1	4	5	3	2
1	3	4	5	2
1	2	3	4	5

insertion

Quiz

Execute two further iterations of the algorithm Quicksort on the following array.
The first element of the (sub-)array serves as the pivot.

8	7	10	15	3	6	9	5	2	13
2	7	5	6	3	8	**9**	10	15	13

Quiz

Execute two further iterations of the algorithm Quicksort on the following array.
The first element of the (sub-)array serves as the pivot.

8	7	10	15	3	6	9	5	2	13
2	7	5	6	3	8	**9**	10	15	13
2	**7**	5	6	3					

Quiz

Execute two further iterations of the algorithm Quicksort on the following array.
The first element of the (sub-)array serves as the pivot.

8	7	10	15	3	6	9	5	2	13
2	7	5	6	3	8	**9**	10	15	13
2	**7**	5	6	3	8				

Quiz

Execute two further iterations of the algorithm Quicksort on the following array.
The first element of the (sub-)array serves as the pivot.

8	7	10	15	3	6	9	5	2	13
2	7	5	6	3	8	**9**	10	15	13
2	**7**	5	6	3	8	9	**10**	15	13

Quiz

Execute two further iterations of the algorithm Quicksort on the following array.
The first element of the (sub-)array serves as the pivot.

8	7	10	15	3	6	9	5	2	13
2	7	5	6	3	8	**9**	10	15	13
2	**7**	5	6	3	8	9	**10**	15	13
2	3	5	6	7					

Quiz

Execute two further iterations of the algorithm Quicksort on the following array.
The first element of the (sub-)array serves as the pivot.

8	7	10	15	3	6	9	5	2	13
2	7	5	6	3	8	**9**	10	15	13
2	**7**	5	6	3	8	9	**10**	15	13
2	3	5	6	7	8				

Quiz

Execute two further iterations of the algorithm Quicksort on the following array. The first element of the (sub-)array serves as the pivot.

8	7	10	15	3	6	9	5	2	13
2	7	5	6	3	8	**9**	10	15	13
2	**7**	5	6	3	8	9	**10**	15	13
2	3	5	6	7	8	9			

Quiz

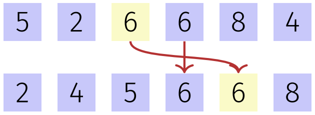
Execute two further iterations of the algorithm Quicksort on the following array.
The first element of the (sub-)array serves as the pivot.

8	7	10	15	3	6	9	5	2	13
2	7	5	6	3	8	**9**	10	15	13
2	**7**	5	6	3	8	9	**10**	15	13
2	3	5	6	7	8	9	10	15	13

←

Stable and in-situ sorting algorithms

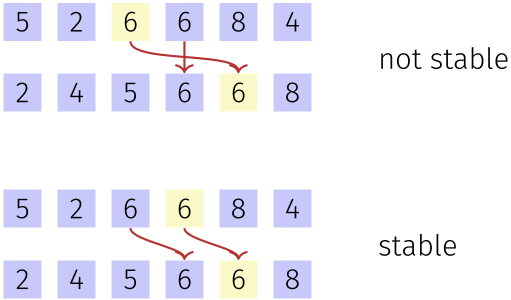
- Stable sorting algorithms don't change the relative position of two equal elements.



not stable

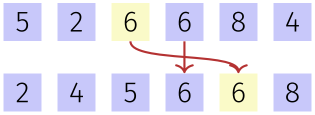
Stable and in-situ sorting algorithms

- Stable sorting algorithms don't change the relative position of two equal elements.

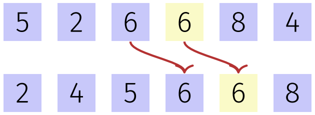


Stable and in-situ sorting algorithms

- Stable sorting algorithms don’t change the relative position of two equal elements.



not stable



stable

- In-situ algorithms require only a constant amount of additional memory.

Discussion: Which of the sorting algorithms are stable? Which are in-situ? (How) can we make them stable / in-situ?

9. In-Class Code-Examples

Implement (Binary) Search from Scratch
$\longrightarrow$ CodeExpert
Use the result to implement binary insertion sort.
$\longrightarrow$ CodeExpert

10. Past Exam Questions

Past Exam 2020: Task 2b)

(b) Gegeben sei die folgende Rekursionsgleichung:

Consider the following recursion equation:

$$T(n) = \begin{cases} 2T(\frac{n}{4}) + 1, & n > 1 \\ 1 & n = 1 \end{cases}$$

Geben Sie eine geschlossene (nicht rekursive), einfache Formel für $T(n)$ an und beweisen Sie diese mittels vollständiger Induktion. Gehen Sie davon aus, dass n eine Potenz von 4 ist.

Hinweis:

Für $q \neq 1$ gilt $\sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1}$.

Specify a closed (non-recursive), simple formula for $T(n)$ and prove it using mathematical induction. Assume that n is a power of 4.

Hint:

For $q \neq 1$ it holds that $\sum_{i=0}^{k} q^i = \frac{q^{k+1}-1}{q-1}$.

Past Exam 2020: Task 2b) — Solution

$$
\begin{aligned}
T(4^k) &= 2T(4^{k-1}) + 1 \\
&= 2(2(T(4^{k-2}) + 1) + 1 = \dots \\
&= 2^k T(4^{k-k}) + 2^{k-1} + 2^{k-2} + \dots + 2 + 1 \\
&= \sum_{j=0}^{k} 2^j = 2^{k+1} - 1 \\
&= 2 \cdot 2^{\log_4 n} - 1 = 2 \cdot n^{\log_4 2} - 1 = 2\sqrt{n} - 1.
\end{aligned}
$$

`Assumption:` $T(n) = 2\sqrt{n} - 1$
`Induktion:`

1. `Hypothesis` $T(n) = f(n) := 2\sqrt{n} - 1$
2. `Base Case` $T(1) = 1 = f(1) = 2\sqrt{1} - 1 = 1.$
3. `Step` $T(n) = f(n) \longrightarrow T(4 \cdot n) = f(4 \cdot n)$ $(n = 4^k$ `for some` $k \in \mathbb{N})$`:`

$$
\begin{aligned}
T(4n) &= 2T(n) + 1 \\
&\overset{i.h.}{=} 2(2 \cdot \sqrt{n} - 1) + 1 \\
&= 2\sqrt{4n} - 2 + 1 \\
&= 2\sqrt{4n} - 1 \\
&= f(4n).
\end{aligned}
$$

←

Past Exam 2020: Task 2e)

Gegeben sei die folgende Rekursionsgleichung:

Consider the following recursion equation:

$$T(n) = \begin{cases} 2T(n/4) + \log_4 n, & n > 1 \\ 0 & n \leq 1 \end{cases}$$

Schreiben Sie eine Funktion g, die bei Aufruf von $g(n)$ genau $T(n)$ Aufrufe von f erzeugt. Nehmen Sie an, dass $n = 4^k$ für ein $k \geq 0$.

Write a function g that when called as $g(n)$ will produce $T(n)$ calls to f. Assume that $n = 4^k$ for some $k \geq 0$.

```
// pre: n = 4^k for some k >= 0
void g(int n){

    }
}
```

Past Exam 2020: Task 2e) — Solution

```
if (n > 1){

    g(n/4); g(n/4);

    while(n > 1){
        f();
        n /= 4;
    }

// } the other brace was closed
//   in the exercise description
```

11. Tips for **code** expert

Tips for **code** expert Exercise 2

Task "Prefix Sum in 2D"

[4] There's an implementation in the code examples on **code** expert

Tips for **code** expert Exercise 2

Task "Prefix Sum in 2D"

- Study the Prefix Sum in 1D[4] well and go from there
- Make sketches!

[4]There's an implementation in the code examples on **code** expert

Tips for code expert Exercise 2

Task "Prefix Sum in 2D"

- Study the Prefix Sum in 1D[4] well and go from there
- Make sketches!

Task "Sliding Window"

[4] There's an implementation in the code examples on **code** expert

Tips for **code** expert Exercise 2

Task "Prefix Sum in 2D"

- Study the Prefix Sum in 1D[4] well and go from there
- Make sketches!

Task "Sliding Window"

- Sketches!

[4]There's an implementation in the code examples on **code** expert

Tips for **code** expert Exercise 2

Task "Prefix Sum in 2D"

- Study the Prefix Sum in 1D[4] well and go from there
- Make sketches!

Task "Sliding Window"

- Sketches!

Task "Proofs by Induction"

[4] There's an implementation in the code examples on **code** expert

Tips for **code** expert Exercise 2

Task "Prefix Sum in 2D"

- Study the Prefix Sum in 1D[4] well and go from there
- Make sketches!

Task "Sliding Window"

- Sketches!

Task "Proofs by Induction"

- The binomial formula will be useful for the second one
- Please format it well or just scan a PDF and upload it

[4]There's an implementation in the code examples on **code** expert

Task "Karatsuba Ofman"

Tips for **code** expert Exercise 2

Task "Karatsuba Ofman"

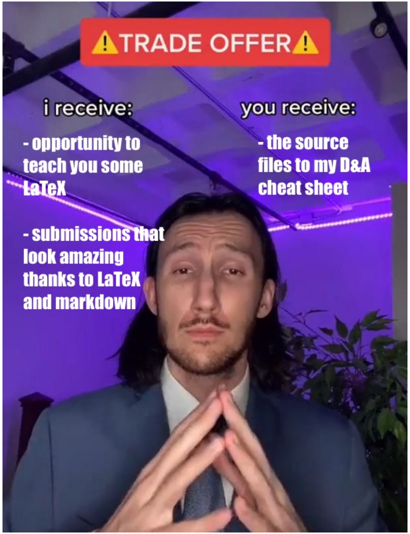
- Translate "3.3.2 Divide And Conquer" from the script into code
- Main struggle: generalizing to non-"power of 2" cases
- Study the definition of `.part(lo, hi)` method
- Make sure to have both "subnumbers" be of equal length
- There might be issues with a silly off-by-one error due to how `n/2` gets calculated — be aware of that
- Naming variable sensibly might prevent you from making silly mistakes

12. Outro

General Questions?

TRADE OFFER

TRADE OFFER


⚠TRADE OFFER⚠
i receive:
- opportunity to teach you some LaTeX
- submissions that look amazing thanks to LaTeX and markdown
you receive:
- the source files to my D&A cheat sheet

←

See you next time!

Have a nice week!

↳ lots of follow up next time!