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"

- 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

- 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)

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?

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 $\Omega(f)$.
- $\Theta(f) = \{g : \mathbb{N} \to \mathbb{R} \mid \exists a > 0,\ b > 0,\ n_0 \in \mathbb{N} : a \cdot f(n) \leq g(n) \leq b \cdot f(n)\ \forall n \geq n_0\}$
- $\Theta(f) = \{g : \mathbb{N} \to \mathbb{R} \mid \exists c > 0,\ n_0 \in \mathbb{N} : \frac{1}{c} \cdot f(n) \leq g(n) \leq c \cdot f(n)\ \forall n \geq n_0\}$

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 $a, b \in \mathbb{N} \setminus \{1\}$

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

Landau Notation

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

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

What I had on my Cheatsheet

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

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

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

[1]there's a mistake here!

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 $f(n) = \ldots$?

- n
- $n^2 + 1$
- $\log^4(n^2)$
- $n \log(n^2)$
- n^π
- $n \cdot 2^{16}$
- $n^2 \cdot 2^{16}$
- 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

Analysis

How many calls to `f()`?

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

How many calls to `f()`?

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

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);
  }
}
```

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()`?

```
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();
}
```

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

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

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

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

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

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

Examples

Naive Matrix Multiplication Divide & Conquer[2]

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

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

[2]Treated in the course later on

Examples

Strassens Matrix Multiplication Divide & Conquer[3]

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

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

[3]Treated in the course later on

Examples

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

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

Examples

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

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

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 ($\varepsilon > 0$):

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

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

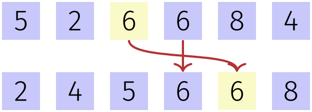
Quiz

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

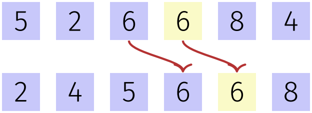
8	7	10	15	3	6	9	5	2	13
2	7	5	6	3	8	**9**	10	15	13

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"

- 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

Tips for **code** expert Exercise 2

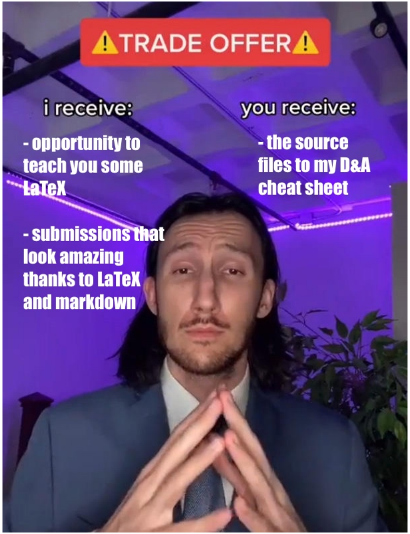
Task "Karatsuba Ofman"

- 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⚠
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!