ETH zürich

Exercise Session 03 – Recurrence, Sorting

Data Structures and Algorithms

These slides are based on those of the lecture, but were adapted and extended by the teaching assistant Adel Gavranović

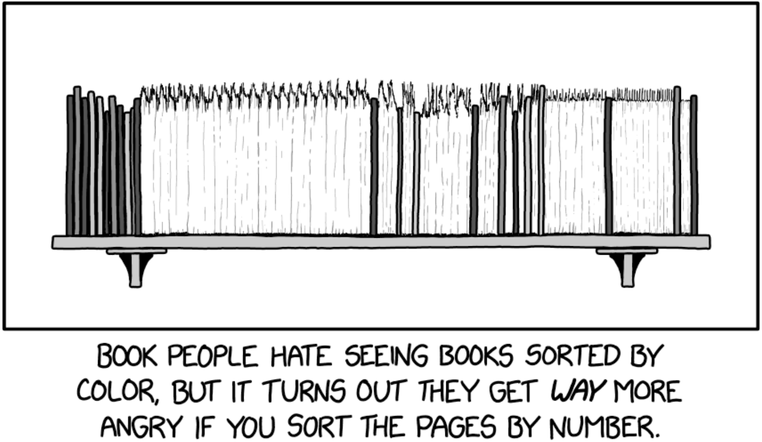
Today's Schedule

Intro
Follow-up
Feedback for **code** expert
Learning Objectives
Landau Notation
Landau Notation Quiz
Analyse the running time of (recursive) Functions
Solving Simple Recurrence Equations
Sorting Algorithms
In-Class Code-Examples
Outro

`n.ethz.ch/~agavranovic`

- Exercise Session Material
- Adel's Webpage
- Mail to Adel

Comic of the Week



▸ xkcd

1. Intro

Intro

- New room
- Please tell the others!

2. Follow-up

Follow-up from last exercise session

- None? Did I forget anything?

3. Feedback for **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 master 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

- Overall pretty bad, so we're gonna have a closer look today
- Was the task description not clear enough?
- 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

Questions regarding **code** expert from your side?

4. Learning Objectives

Learning 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

5. Landau Notation

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 $\Omega(f)$

$$\begin{aligned} \Theta(f) = & \quad \{g : \mathbb{N} \to \mathbb{R} \mid \exists a > 0,\ b > 0,\ n_0 \in \mathbb{N} : a \cdot f(n) \leq g(n) \leq b \cdot f(n)\ \forall n \geq n_0\} \\ = & \quad \{g : \mathbb{N} \to \mathbb{R} \mid \exists c > 0,\ n_0 \in \mathbb{N} : \tfrac{1}{c} \cdot f(n) \leq g(n) \leq c \cdot f(n)\ \forall n \geq n_0\} \end{aligned}$$

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 $f \in \mathcal{O}(g)$. Sort them

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

Sorted:

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

What I had on my Cheatsheet

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

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

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

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!)

6. Landau Notation Quiz

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

7. 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();
  }
}
```

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();
  }
}
```

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()`?

```
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()`?

```
// 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
void g(int n){
  if (n>0){
    f();
    g(n/2);
    f();
    g(n/2);
  }
}
```

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

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();
  }
}
```

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

How many calls to `f()`?

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

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

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:

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

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:

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

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

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

8. 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

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

$\Rightarrow$ 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

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

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

Examples

Naive Matrix Multiplication Divide & Conquer[1]

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

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

[1]Treated in the course later on

Examples

Strassens Matrix Multiplication Divide & Conquer[2]

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

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

[2]Treated in the course later on

Examples

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

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

Examples

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

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

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 ($\varepsilon > 0$):

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

Personal Approach to "Solving RecEqs"

"Plug and Chuck"-Approach

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

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

1. Loops: just multiply
2. If too hard: usually $\Theta(2^n)$
3. Just brute-force calculate $g(0), g(1), g(2), g(3), \ldots$ and try to identify trends
4. If necessary, simply set up and solve RecEqs
5. If asked provide proof (by induction)

9. 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

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

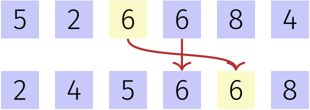
Quiz

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

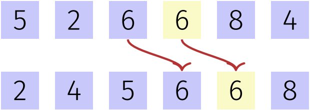
8	7	10	15	3	6	9	5	2	13
2	7	5	6	3	**8**	9	10	15	13
2	7	5	6	3	**8**	**9**	10	15	13
2	3	5	6	**7**	**8**	**9**	**10**	15	13

Stable and in-situ sorting algorithms

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



not stable



stable

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

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

10. In-Class Code-Examples

Implement (Binary) Search from Scratch
⟶ **code** expert
Use the result to implement binary insertion sort.
⟶ **code** expert

11. Outro

General Questions?

See you next time

Have a nice week!