Lecture 1: Introduction and asymptotic complexity

BE5B33ALG — Algorithms

Ing. Robert Pěnička, Ph.D.
doc. RNDr. Daniel Průša, Ph.D.

Faculty of Electrical Engineering
Czech Technical University in Prague

Introduction

- **Course page**
  `https://cw.fel.cvut.cz/b251/courses/be5b33alg/start`
- **Credits**
  6 credits $=$ 10 hours per week! (in 14 weeks + 5-week exam period)
- **Goals**
  Teach efficient solutions of various problems arising from elementary computer science. The main topics include algorithmic complexity, sorting and searching algorithms,
- **Prerequisites**
  Expected capability of proficient programming in either C, C++, Java or Python. Integral and mandatory part are practical programming homeworks. Attendants are expected to be familiar with basic data structures such as arrays, lists, files atc.
- **Literature**
  Cormen TH, Leiserson CE, Rivest RL, Stein C. *Introduction to algorithms*. MIT press; 2022 Apr 5.

Introduction

- **Timetable**
  1.5h Lecture + 1.5h Practices https://intranet.fel.cvut.cz/en/education/rozvrhy-ng.B251/public/html/predmety/43/56/p4356206.html
- **Brute system**
  Upload system for homeworks and exams `https://cw.felk.cvut.cz/brute/`
- **Plan of the semester**

week #	Topic
1	Order of growth of functions, asymptotic complexity
2	Trees, binary trees, recursion
3	More recursion and backtrack examples
4	Graph, graph representation, basic graph processing
5	Queue, Stack, Breadth/Depth First Search
6	Array search, Binary search tree
7	AVL and B- trees
8	Sorting algorithms I
9	Sorting algorithms II
10	Dynamic programming I
11	Dynamic programming II
12	Complexity of recursive algorithms, Master theorem
13	Hashing I & II
14	Individual repetitions and exam preparation

Introduction

- **Points**
  - Points gained by solving **homeworks** (max 36 points).
  - Points gained by solving the **midterm test** (max 12 points).
    Acceptable minimum from homeworks and the test is 24 points (out of 48 points).
  - Points gained by solving the **programming part** (max 20 points).
    Acceptable minimum is 10 points. Each correctly processed input file of total 10 files is worth 2 points.
  - Points gained by solving the **theoretical part** (max 32 points).
    Acceptable minimum is 16 points. The amount of points gained is decided by the examiner.
- **Grading**

Points total	Grade
< 50	F
50 – 59.99	E
60 – 69.99	D
70 – 79.99	C
80 – 89.99	B
90 – 100	A

Problems and algorithms

- **Computational problem** P

  The task of transforming an input IN into an output data OUT such that some prescribed conditions are satisfied.

- **Algorithm A**

  - Computational process that reflects the progress of solution of problem P.
  - Exact and unambiguous description of the sequence of computational steps which transforms input data IN step by step into the output data OUT satisfying the conditions of problem P.

- **Problem instance I**

  Problem P taken together with one set of particular input data.

- **Correctness**

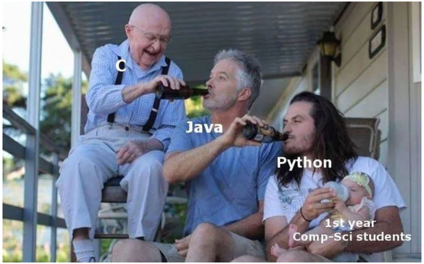
  Property of algorithm A that guarantees that for **each** problem instance I the algorithm produces **appropriate** output in **finite time**.

How to compare algorithms?

Empirically?

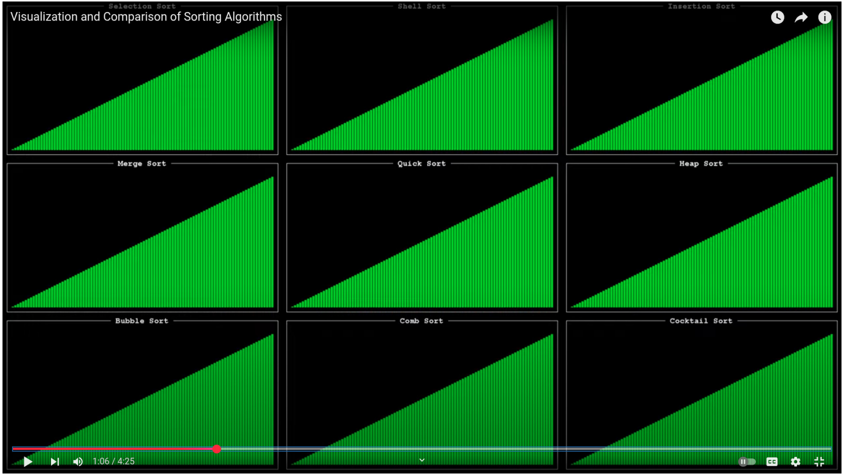
- Based on the algorithm we can write a program which implements it and run it on a computer on some set of instances
- We can compare the speed and memory demands of each implementation.
- However, is this a good way how to compare algorithms?

  - What if the computer is different? What if the OS is different?
  - What if the programming language is different? What if the compiler is different?
  - What if the instances are different?



- A more fair comparison of particular algorithms is needed for independent comparison...

How to compare algorithms?



https://www.youtube.com/watch?v=ZZuD6iUe3Pc

Computing the complexity

- **Elementary operation:** an operation that takes a constant amount of time, e.g., arithmetic operations, comparisons, assignments, memory moves.
- **Complexity:** the number of elementary operations performed by an algorithm (with respect to input size).
- **Worst case:** when all conditions are unfavorable w.r.t. the complexity (e.g., searching for an element in an unsorted list).
- **Best case:** when all conditions are favorable w.r.t. the complexity (e.g., searching for an element in a sorted list).

Computing the complexity - Examples

- Find minimum min and maximum max value in an array a.
- Standard variant iterate index i from 1 to $n-1$ by one.

$a =$

3	2	7	10	0	5	-10	4	6

- Result is $min = -10, max = 10$

```
min = a[0], max = a[0]
for i ← 1 to n − 1 do
    if a[i] < min then
        min = a[i]
    if a[i] > max then
        max = a[i]
```

Computing the complexity – example

- Find minimum min and maximum max value in an array a.
- Faster variant: iterate index i from 1 to $n-1$ by two.

$a =$									
	3	2	7	10	0	5	-10	4	6

$min = a[0], max = a[0]$
for i **to** $n-1$ *with* $i = i+2$ **do**
    if $a[i] < a[i+1]$ **then**
        if $a[i] < min$ **then**
            $min = a[i]$
        if $a[i+1] > max$ **then**
            $max = a[i+1]$
    else
        if $a[i] > max$ **then**
            $max = a[i]$
        if $a[i+1] < min$ **then**
            $min = a[i+1]$

Computing the complexity – example

- Assume N is the size of the input array a.
- **Complexity:** the number of elementary operations performed.
- Can be **simplified** to number of elementary operations on the data (e.g. on our array a).
- **Most commonly simplified to the number of comparisons made on the data.**

$\overbrace{min = a[0]}^{1}, \overbrace{max = a[0]}^{1}$

for $\overbrace{i \leftarrow 1}^{1}$ **to** $\overbrace{n - 1}^{N-1}$ **do**

    if $\overbrace{a[i] < min}^{N-1}$ **then**

        $\overbrace{min = a[i]}^{0 \ldots N-1}$

    if $\overbrace{a[i] > max}^{N-1}$ **then**

        $\overbrace{max = a[i]}^{0 \ldots N-1}$

$\overbrace{min = a[0]}^{1}, \overbrace{max = a[0]}^{1}$

for $\overbrace{i \leftarrow 1}^{1}$ **to** $\overbrace{n - 1 \textit{ with } i = i + 2}^{(N-1)/2}$ **do**

    if $a[i] < a[i+1]$ **then** $\} \ \frac{N-1}{2}$

        if $a[i] < min$ **then** $\} \ \frac{N-1}{2}$

            $min = a[i] \} \ 0 \ldots (N-1)/2$

        if $a[i+1] > max$ **then** $\} \ \frac{N-1}{2}$

            $max = a[i+1] \} \ 0 \ldots (N-1)/2$

    else

        if $a[i] > max$ **then** $\} \ \frac{N-1}{2}$

            $max = a[i] \} \ 0 \ldots (N-1)/2$

        if $a[i+1] < min$ **then** $\} \ \frac{N-1}{2}$

            $min = a[i+1] \} \ 0 \ldots (N-1)/2$

Computing the complexity – example

- Assume N is the size of the input array a.
- What is the number of elementary operations of the standard algorithm?
- What is the number of elementary operations on the data (array a?) of the standard algorithm?
- What is the number of comparisons made on the data (array a?) of both algorithms?

$\overbrace{min = a[0]}^{1}, \overbrace{max = a[0]}^{1}$
for $\overbrace{i \leftarrow 1}^{1}$ **to** $\overbrace{n-1}^{N-1}$ **do**
  if $\overbrace{a[i] < min}^{N-1}$ **then**
    $\overbrace{min = a[i]}^{0\dots N-1}$
  if $\overbrace{a[i] > max}^{N-1}$ **then**
    $\overbrace{max = a[i]}^{0\dots N-1}$

$\overbrace{min = a[0]}^{1}, \overbrace{max = a[0]}^{1}$
for $\overbrace{i \leftarrow 1}^{1}$ **to** $\overbrace{n-1 \text{ with } i = i+2}^{(N-1)/2}$ **do**
  if $a[i] < a[i+1]$ **then** $\}\ \frac{N-1}{2}$
    if $a[i] < min$ **then** $\}\ \frac{N-1}{2}$
      $min = a[i]\}\ 0\dots(N-1)/2$
    if $a[i+1] > max$ **then** $\}\ \frac{N-1}{2}$
      $max = a[i+1]\}\ 0\dots(N-1)/2$
  else
    if $a[i] > max$ **then** $\}\ \frac{N-1}{2}$
      $max = a[i]\}\ 0\dots(N-1)/2$
    if $a[i+1] < min$ **then** $\}\ \frac{N-1}{2}$
      $min = a[i+1]\}\ 0\dots(N-1)/2$

Computing the complexity – example

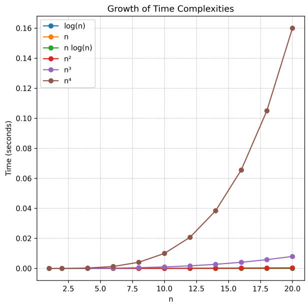
- Total operations: $3N$ best case and $5N-2$ worst case (includes also blue and green).
- Operations on data: $2N$ best case and $4N-2$ worst case (includes also green).
- Comparisons on data: always $2N-2$ for standard and always $3(N-1)/2$ for faster variant.

$\overbrace{min = a[0]}^{1}, \overbrace{max = a[0]}^{1}$
for $\overbrace{i \leftarrow 1}^{1}$ **to** $\overbrace{n-1}^{N-1}$ **do**
  if $\overbrace{a[i] < min}^{N-1}$ **then**
    $\overbrace{min = a[i]}^{0 \dots N-1}$
  if $\overbrace{a[i] > max}^{N-1}$ **then**
    $\overbrace{max = a[i]}^{0 \dots N-1}$

$\overbrace{min = a[0]}^{1}, \overbrace{max = a[0]}^{1}$
for $\overbrace{i \leftarrow 1}^{1}$ **to** $\overbrace{n - 1 \text{ with } i = i + 2}^{(N-1)/2}$ **do**
  if $a[i] < a[i+1]$ **then** $\} \ \frac{N-1}{2}$
    if $a[i] < min$ **then** $\} \ \frac{N-1}{2}$
      $min = a[i] \} \ 0 \dots (N-1)/2$
    if $a[i+1] > max$ **then** $\} \ \frac{N-1}{2}$
      $max = a[i+1] \} \ 0 \dots (N-1)/2$
  else
    if $a[i] > max$ **then** $\} \ \frac{N-1}{2}$
      $max = a[i] \} \ 0 \dots (N-1)/2$
    if $a[i+1] < min$ **then** $\} \ \frac{N-1}{2}$
      $min = a[i+1] \} \ 0 \dots (N-1)/2$

Rate of growth of functions

- n number of the data items (size of the data)
- Assumes one operation takes exactly 1 μ
- $T(n)$ time needed to process n data items



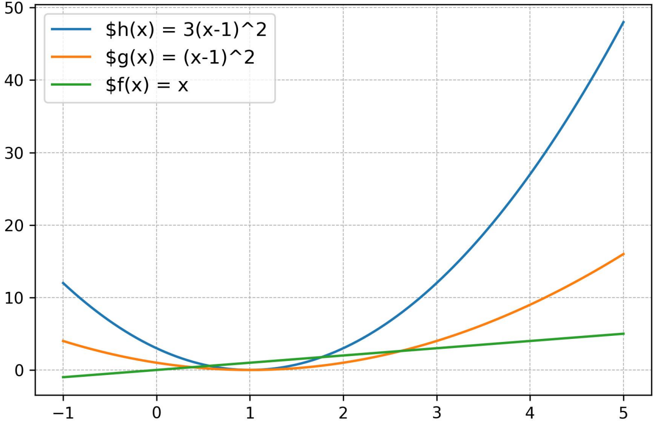
$n/T(n)$	20	40	60	80	100
$\log(n)$	4.3 μs	5.3 μs	5.9 μs	6.3 μs	6.6 μs
n	20 μs	40 μs	60 μs	80 μs	0.1 ms
$n\log(n)$	86 μs	0.2 ms	0.35 ms	0.5 ms	0.7 ms
n^2	0.4 ms	1.6 ms	3.6 ms	6.4 ms	10 ms
n^3	8 ms	64 ms	0.22 sec	0.5 sec	1 sec
n^4	0.16 sec	2.56 sec	13 sec	41 sec	100 sec
2^n	1 sec	12.7 days	36600 yrs	10^{11} yrs	10^{16} yrs
$n!$	77100 yrs	10^{34} yrs	10^{68} yrs	10^{105} yrs	10^{144} yrs

Asymptotic estimations – Upper asymptotic estimate O

- Upper asymptotic estimate (big-O (omicron) notation)
  $f(n) \in O(g(n))$
- **Meaning:**
  function f is asymptotically upper-bounded (from above) by a function g disregarding the additive and multiplicative constants.
- **Definition:**
  $(\exists c > 0)(\exists n_0)(\forall n > n_0) : \boldsymbol{f(n) \leq c \cdot g(n)}$
  where: $c \in \mathbb{R}^{>0}$ $n_o, n \in \mathbb{N}$ $f, g \in \mathbb{N} \rightarrow \mathbb{R}^{\geq 0}$

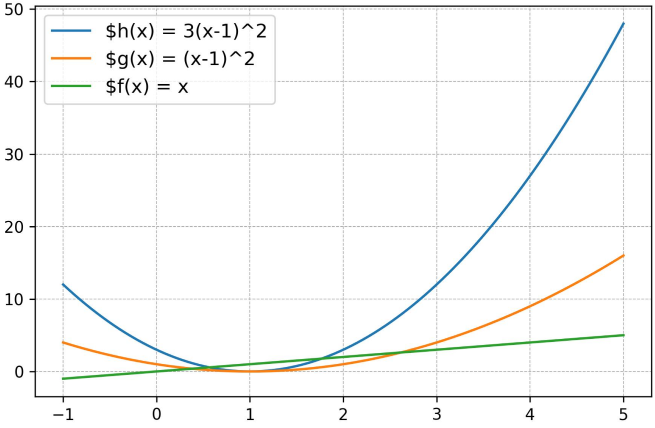
Asymptotic estimations – examples

- Is it true that $f(n) \in O(g(n))$ and that $h(n) \in O(g(n))$?
- What are the values of c and n_0? ($\boldsymbol{f(n) \in O}$ **if** $\boldsymbol{f(n) \leq c \cdot g(n)}$)



Asymptotic estimations – examples

- Is it true that $f(n) \in O(g(n))$ YES and that $h(n) \in O(g(n))$ YES?
- What are the values of c and n_0? ($\boldsymbol{f(n) \in O}$ if $\boldsymbol{f(n) \leq c \cdot g(n)}$)



Asymptotic estimations – Lower asymptotic estimate Ω

- Lower asymptotic estimate (big-Omega notation)
  $f(n) \in \Omega(g(n))$
- **Meaning:**
  function f is asymptotically lower-bounded (bounded from below) by a function g disregarding the additive and multiplicative constants.
- **Definition:**
  $(\exists c > 0)(\exists n_0)(\forall n > n_0) : \boldsymbol{c \cdot g(n) \leq f(n)}$
  where: $c \in \mathbb{R}^{>0}$ $n_o, n \in \mathbb{N}$ $f, g \in \mathbb{N} \to \mathbb{R}^{\geq 0}$

Asymptotic estimations for functions of more variables

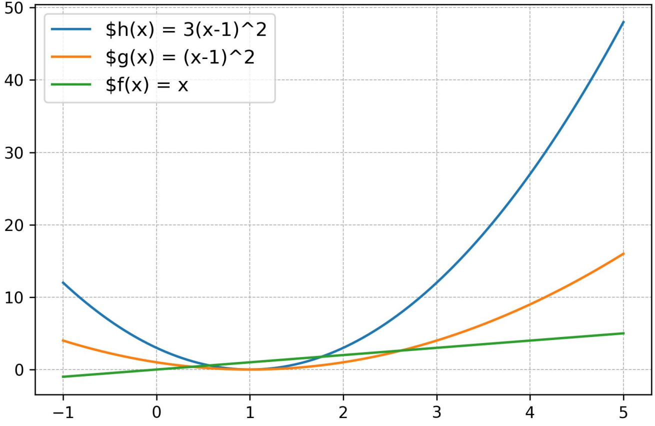
- Upper asymptotic estimate (big-O (omicron) notation)
  $f(n_1, \cdots, n_k) \in O(g(n_1, \cdots, n_k))$
- **Definition:**
  $(\exists c > 0)(\exists n_0)(\forall n_1 > n_0) \cdots (\forall n_k > n_0) : \boldsymbol{f(n_1, \cdots, n_k) \leq c \cdot g(n_1, \cdots, n_k)}$
  where: $c \in \mathbb{R}^{>0}$ $n_o, n_1, \cdots, n_k \in \mathbb{N}$ $f, g \in \mathbb{N} \to \mathbb{R}^{\geq 0}$
- Lower asymptotic estimate (big-Omega notation) can be defined similarly for functions of more variables as the for the single-variable functions.

Asymptotic estimations – Optimal asymptotic estimate Θ

- Optimal asymptotic estimate (capital Theta notation)
  $f(n) \in \Theta(g(n))$
- **Meaning:**
  function f is asymptotically bounded from above and from below by a function g disregarding the additive and multiplicative constants.
- **Definition:**
  $\Theta(g(n)) := O(g(n)) \cap \Omega(g(n))$
  i.e. $(\exists c_1, c_2 > 0)(\exists n_0)(\forall n > n_0) : \boldsymbol{c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)}$
  where: $c_1, c_2 \in \mathbb{R}^{>0}$ $n_o, n \in \mathbb{N}$ $f, g \in \mathbb{N} \rightarrow \mathbb{R}^{\geq 0}$

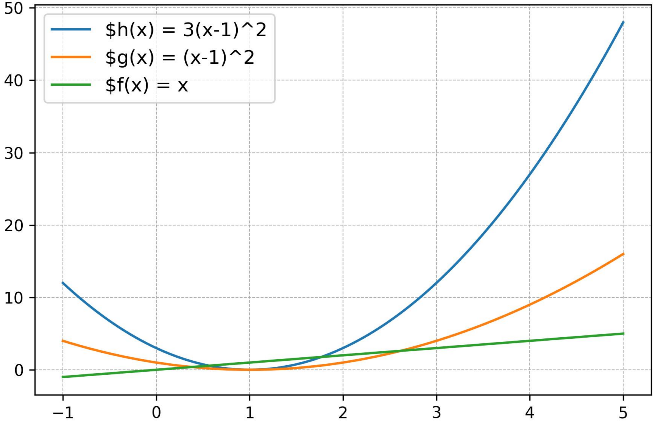
Asymptotic estimations – examples

- Is it true that $f(n) \in \Theta(g(n))$ and that $h(n) \in \Theta(g(n))$?
- What are the values of c_1, c_2 and n_0? ($\boldsymbol{\Theta(g(n)) : c_1 \cdot g(n) \leq h(n) \leq c_2 \cdot g(n))}$



Asymptotic estimations – examples

- Is it true that $f(n) \in \Theta(g(n))$ NO and that $h(n) \in \Theta(g(n))$ YES ?
- What are the values of c_1, c_2 and n_0? ($\boldsymbol{\Theta(g(n)) : c_1 \cdot g(n) \leq h(n) \leq c_2 \cdot g(n)}$)



Asymptotic estimations – examples

- Example: What is the asymptotic complexity of finding maximum value in 2D array with $M \times N$ numbers?

Which are upper bounds?

- $O((M+N)^2)$
- $O(max(M,N)^2)$
- $O(N^2)$
- $O(M \cdot N)$

Which are lower bounds?

- $\Omega(1)$
- $\Omega(M)$
- $\Omega(M \cdot N)$

Which is the optimal asymptotic complexity of finding maximum value in 2D array with $M \times N$ numbers?

Asymptotic estimations – examples

- Example: What is the asymptotic complexity of finding maximum value in 2D array with $M \times N$ numbers?

Which are upper bounds?

- $O((M+N)^2)$ YES
- $O(max(M,N)^2)$ YES
- $O(N^2)$ NO
- $O(M \cdot N)$ YES

Which are lower bounds?

- $\Omega(1)$ YES
- $\Omega(M)$ YES
- $\Omega(M \cdot N)$ YES

Which is the optimal asymptotic complexity of finding maximum value in 2D array with $M \times N$ numbers? $\Theta(M \cdot N)$

Asymptotic estimations – typical classes

An algorithm with complexity $f(n)$ is said to be:

- - **logarithmic**, if $f(n) \in \Theta(\log(n))$
  - **linear**, if $f(n) \in \Theta(n)$
  - **quadratic**, if $f(n) \in \Theta(n^2)$
  - **cubic**, if $f(n) \in \Theta(n^3)$
  - **polynomial**, if $f(n) \in \Theta(n^k)$ for some $k \in \mathbb{N}$
  - **exponential**, if $f(n) \in \Theta(2^n)$ for some $k \in \mathbb{N}$
- Note that logarithmic complexities do not require to include the base of the logarithm, as for any base a, b it holds that $\log_a(n) = \Theta(\log_b(n))$.

Asymptotic estimations – typical classes

An algorithm with complexity $f(n)$ is said to be:

- Why exactly this holds $\log_a(n) = \Theta(\log_b(n))$?
- Using formula for change of base of logarithm:

$$\log_a(n) = \frac{\log_b(n)}{\log_b(a)} = \underbrace{\frac{1}{\log_b(a)}}_{\text{constant}} \cdot \log_b(n)$$

- We can show that there exists c_1, c_2 such that for all $n > n_0$ it holds that

$$c_1 \cdot \log_b(n) \leq \log_a(n) \leq c_2 \cdot \log_b(n)$$

where $c_1 \leq \frac{1}{\log_b(a)} \leq c_2$ and $n_0 = 1$.

Asymptotic estimations – properties

- Properties of the asymptotic complexity:

$$
\begin{aligned}
n^m &\in O(n^{m'}) \quad \text{if } m \leq m' \\
f(n) &\in O(f(n)) \\
c \cdot O(f(n)) &= O(c \cdot f(n)) = O(f(n)) \\
O(O(f(n))) &= O(f(n)) \\
O(f(n)) + O(g(n)) &= O(\max\{f(n), g(n)\}) \\
O(f(n)) \cdot O(g(n)) &= O(f(n) \cdot g(n)) \\
O(f(n) \cdot g(n)) &= f(n) \cdot O(g(n))
\end{aligned}
$$

- The class of the complexity of a polynomial is given by the term with the highest exponent: $\sum_{i=0}^{k} a_i \cdot n^{k-i} \in \sum_{i=0}^{k} O(n^k) = k \cdot O(n^k) = O(k \cdot n^k) = O(n^k)$

Properties of asymptotic estimations

- Theorem: If functions $f(n), g(n)$ are always positive, then for the limit at infinity:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = 0 \;\;\Rightarrow\;\; f(n) \in O(g(n)),\text{ but not } f(n) \in \Theta(g(n))$$
$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = a, \quad 0 < a < \infty \;\;\Rightarrow\;\; f(n) \in \Theta(g(n))$$
$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \infty \;\;\Rightarrow\;\; g(n) \in O(f(n)),\text{ but not } g(n) \in \Theta(f(n))$$

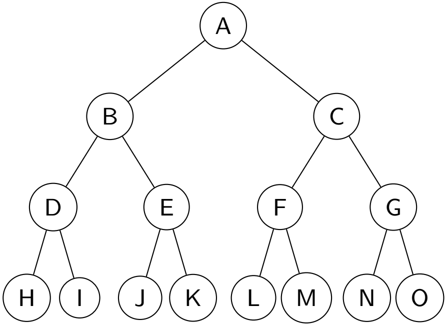
- Corollary: Let $i \in \mathbb{N}$ be a fixed integer. Then

$$(\log n)^i \in O(n)$$

You may use L'Hôpital's rule to prove the corollary.

Why complexity is important?

- What is better complexity, $log(n)$, n or n^2 ?
- what is the $log(n)$ complexity associated with and why? what about the n and n^2?

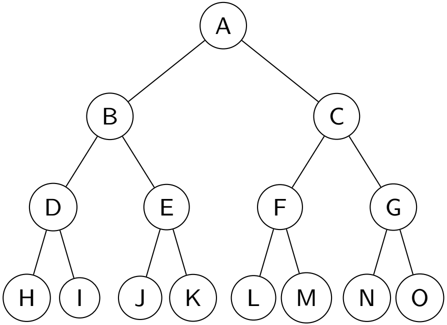


A	B	C	D	E	F
0	1	2	3	4	5

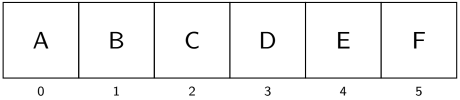
	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

Why complexity is important?

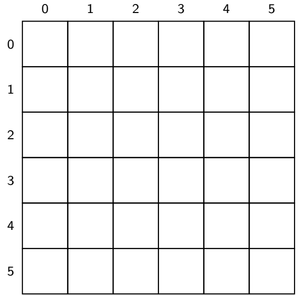
- What is better complexity, $log(n)$, n or n^2 ?
- what is the $log(n)$ complexity associated with and why? what about the n and n^2?



Tree: often logarithmic (search), or linear (traversal)



1D Array: linear



2D Array: quadratic (full scan), linear (row/col scan)

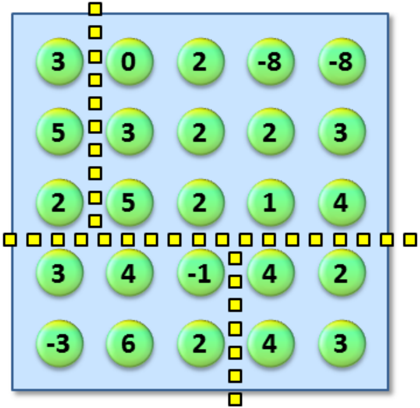
Homework: Orchard Division

Problem setup:

- Orchard is an $N \times N$ grid of trees.
- Each tree has an integer **quality index**.
- Divide into **4 rectangular parts** using:
  - One horizontal cut (East–West).
  - One vertical cut in the North.
  - One vertical cut in the South.

Goal: Minimize the difference between the **max** and **min** part quality. (Part quality = sum of tree values)

Input: N, then N rows of N integers. **Output:** Minimum possible difference.



Example division of the orchard.

Maybe some hint?

Prefix sums

Let:

- $A \ldots$ array of numbers indexed from 1 to N
- $P \ldots$ array of prefix sums (same length as A), indexed from 0 to N

Definition:

$$P[i] = A[1] + A[2] + \cdots + A[i], \quad P[0] = 0$$

Example:

$$A = \{1, -2, 4, 5, -1, -5, 2, 7\}$$

$$P = \{0, 1, -1, 3, 8, 7, 2, 4, 11\}$$

Any subarray sum can be computed in constant time:

$$A[i] + A[i+1] + \cdots + A[j] = P[j] - P[i-1]$$

See more: `https://en.wikipedia.org/wiki/Prefix_sum`