Great Ideas in Theoretical CS

Lecture 9: Time Complexity

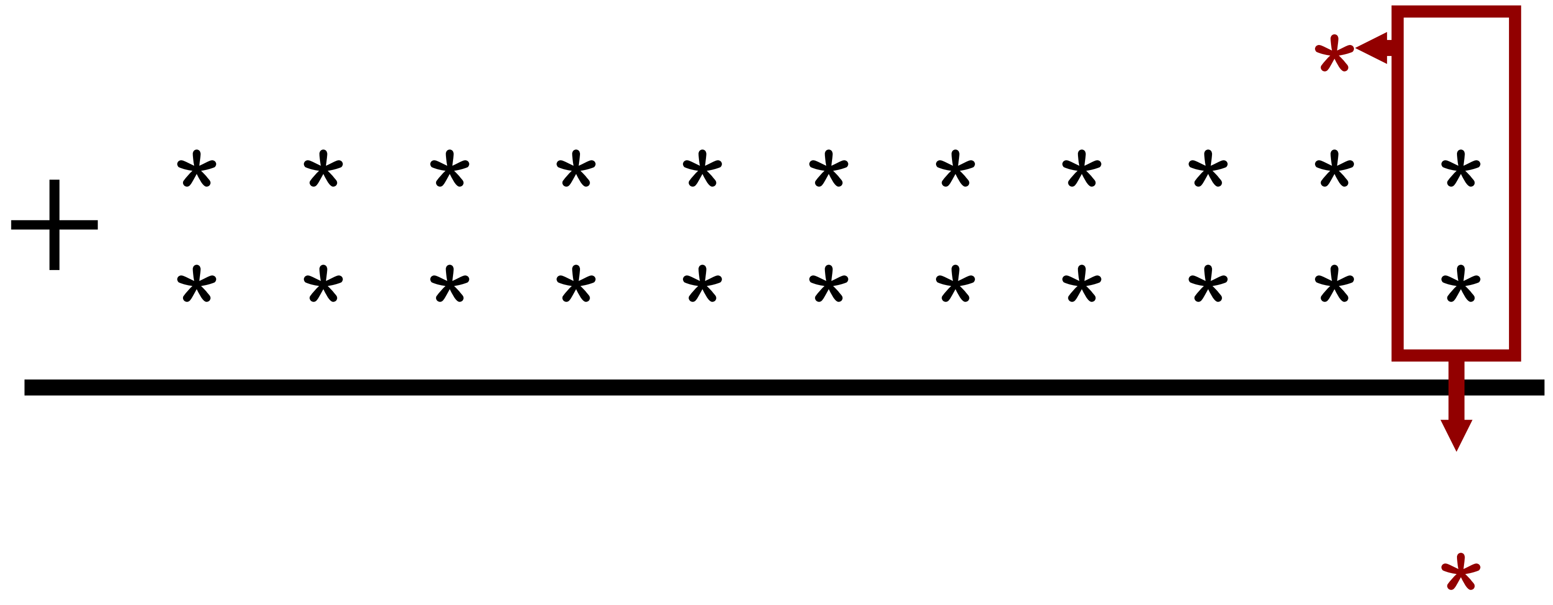
Anil Ada
Ariel Procaccia (this time)

The big O

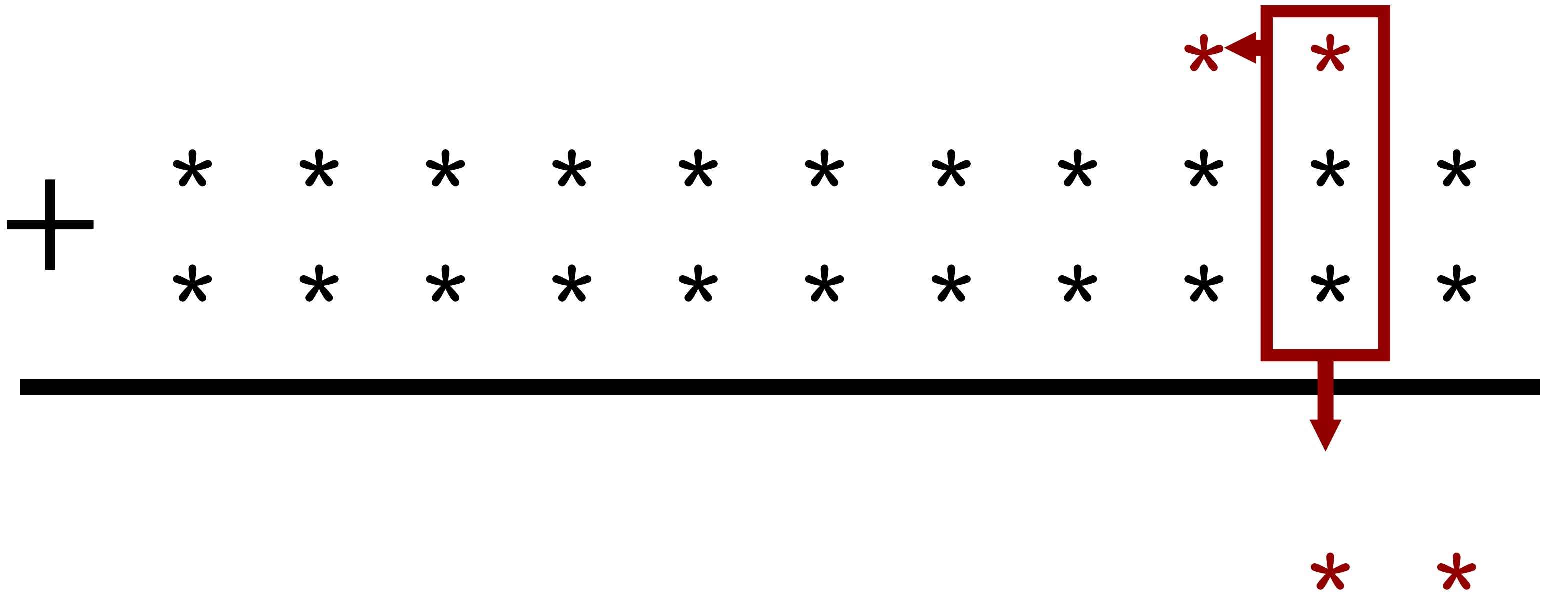
Adding two n-bit numbers

$$\begin{array}{c c c c c c c c c c c c} & * & * & * & * & * & * & * & * & * & * & * \\ + & * & * & * & * & * & * & * & * & * & * & * \\ \hline \end{array}$$

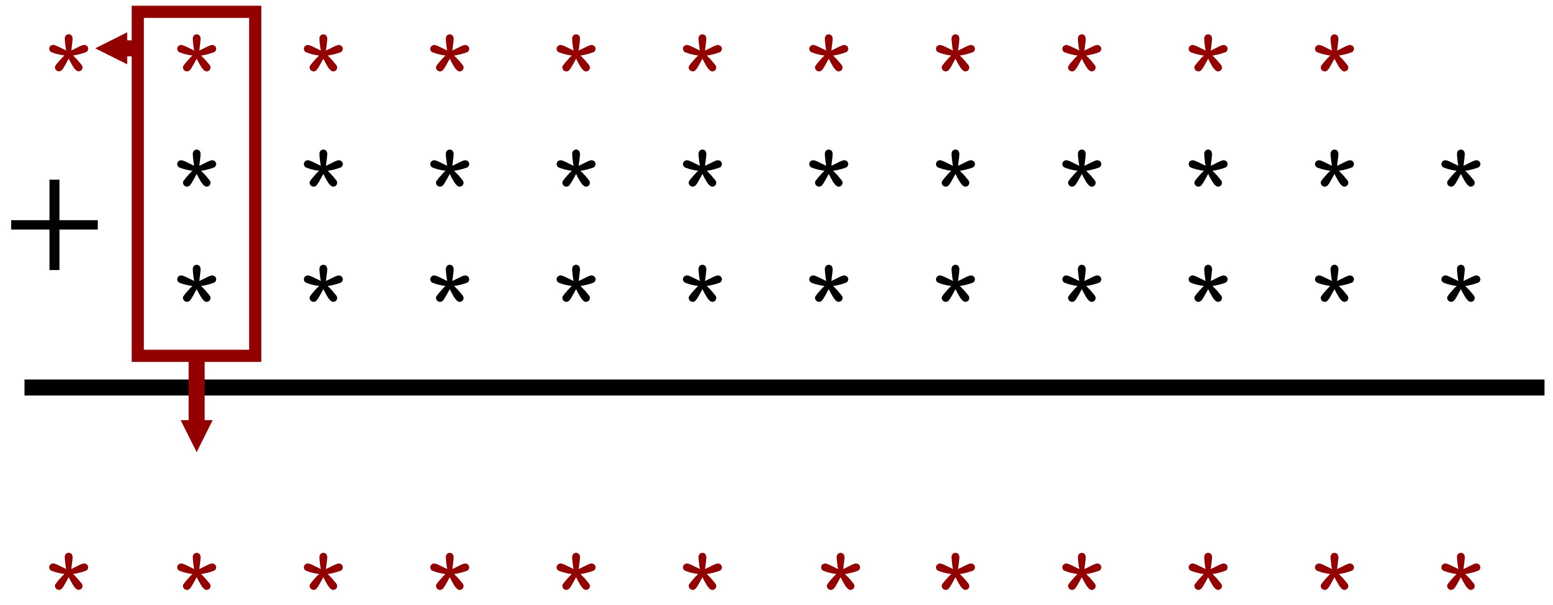
Adding two n-bit numbers



Adding two n-bit numbers



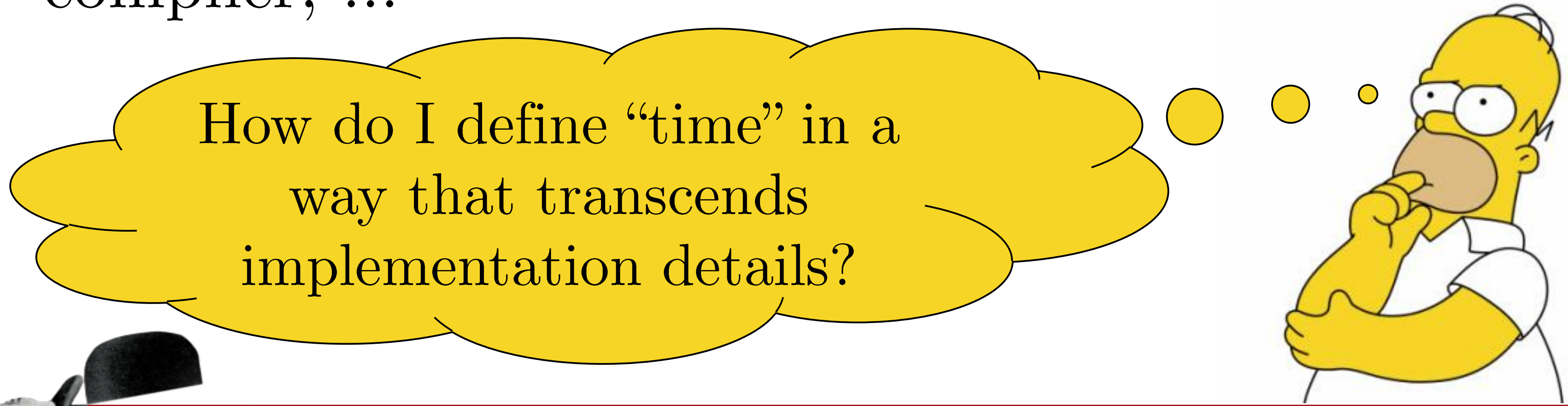
Adding two n-bit numbers



Grade school addition

Time complexity

- $T(n)$ = amount of time grade school addition takes to add two n-bit numbers
- What do we mean by "time"?
- Given algorithm will take different amounts of time on the same input depending on hardware, compiler, ...

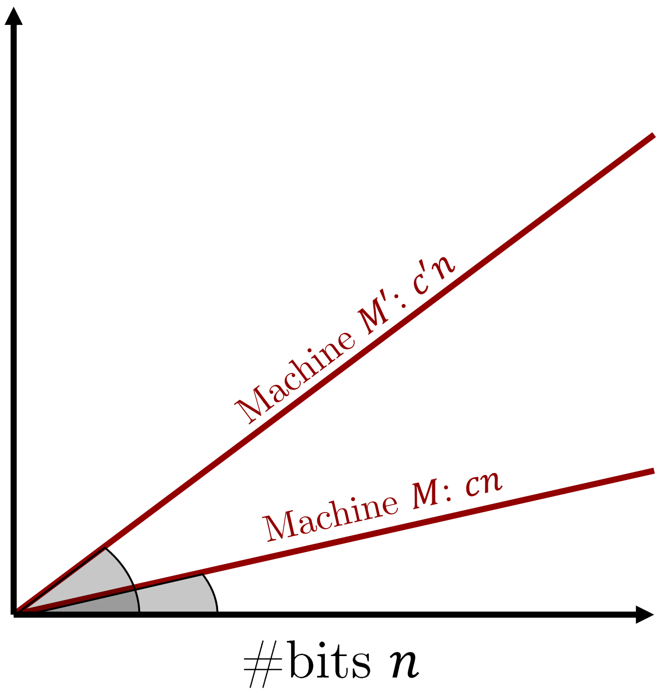

How do I define "time" in a way that transcends implementation details?

A great idea

- On any reasonable computer, adding 3 bits and writing down the 2 bit answer can be done in constant time
- For a computer M, let c be the time it takes to perform on M
- The total time to add two n-bit numbers using grade school addition on M is $c \cdot n$
- On M', the time to perform could be c'
- The total time on M' is $c'n$

A great idea

- The fact that we get a line is invariant under different implementations
- Different machines result in different slopes, but the running time grows linearly

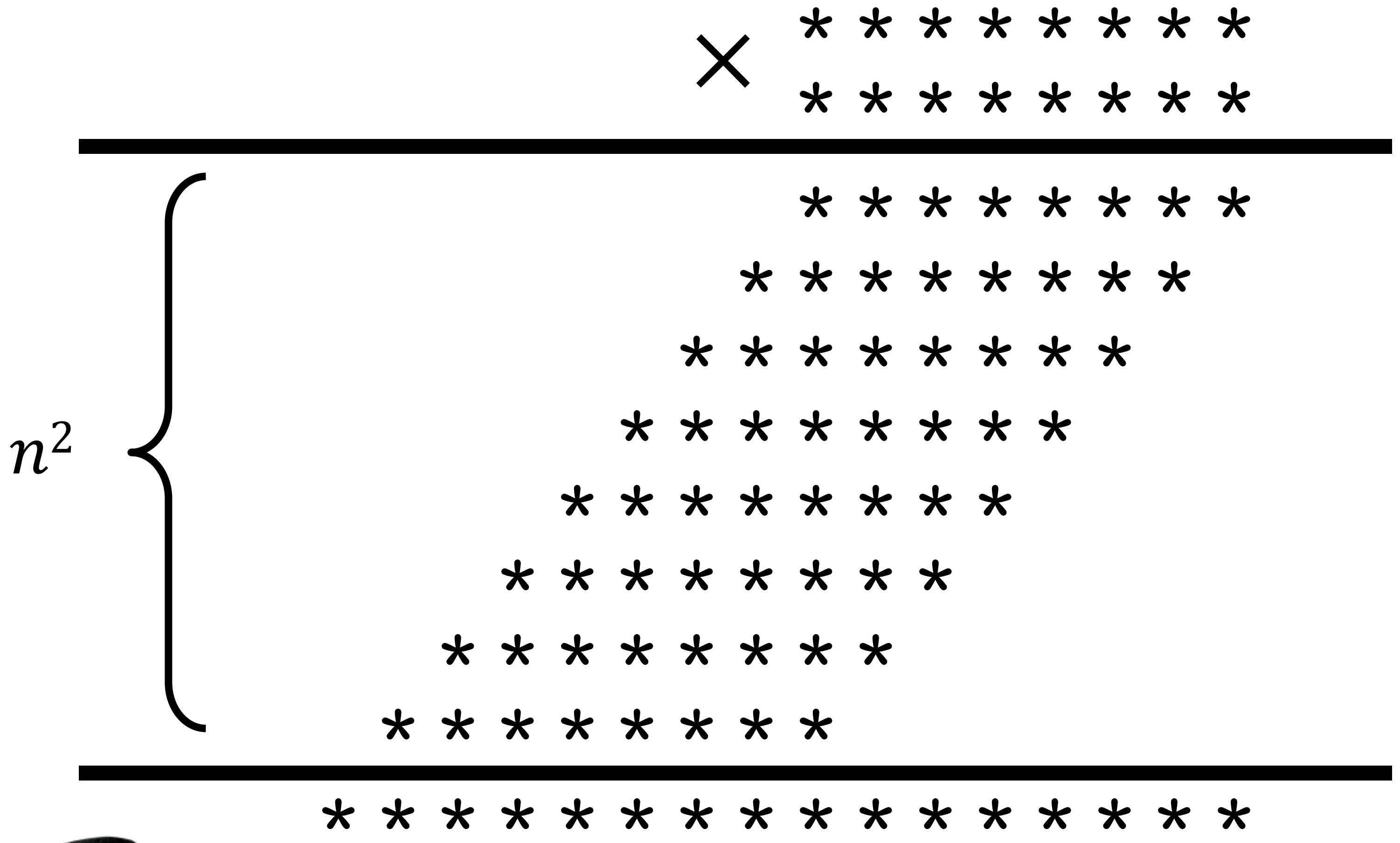


A great idea

- Conclusion: Grade school addition is a linear time algorithm

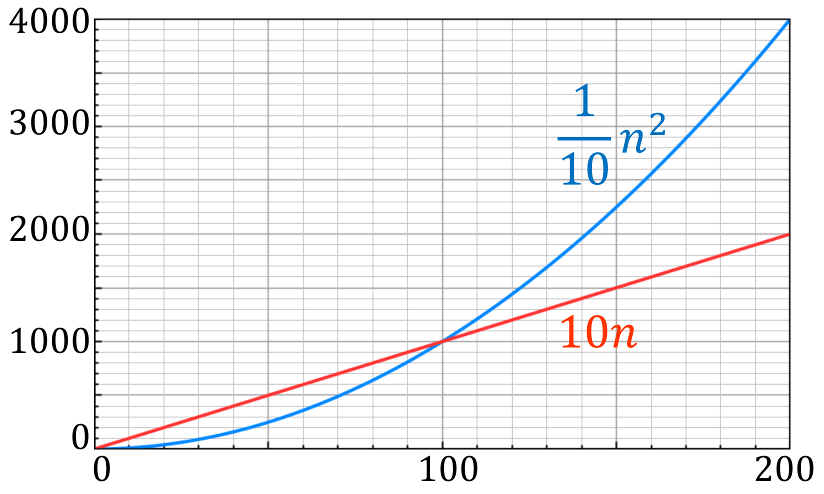
This process of abstracting away details and determining the rate of resource usage in terms of the problem size n is one of the fundamental ideas in computer science

Multiplying two n-bit numbers



Linear vs. quadratic

- Total time to multiply: cn^2
- Addition is linear time, multiplication is quadratic time
- Regardless of the constants, the quadratic curve will eventually dominate the linear curve



Nursery school addition

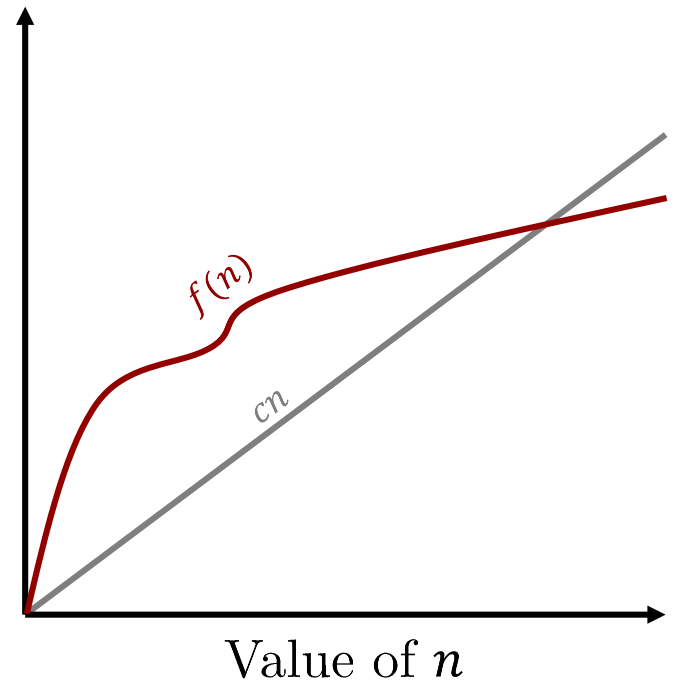
- To add two n-bit numbers a and b, start at a and increment (by 1) b times
- What is $T(n)$?
- If $b = 00\cdots0$, NSA takes almost no time
- Poll 1: If $b = 11\cdots1$, NSA takes time
  1. $c(\log n)^2$
  2. $cn\log n$
  3. cn^2
  4. $cn2^n$

Worst case time

Worst-case running time $T(n)$ of algorithm $A =$ the maximum over all feasible inputs x of size n of the running time of A on x

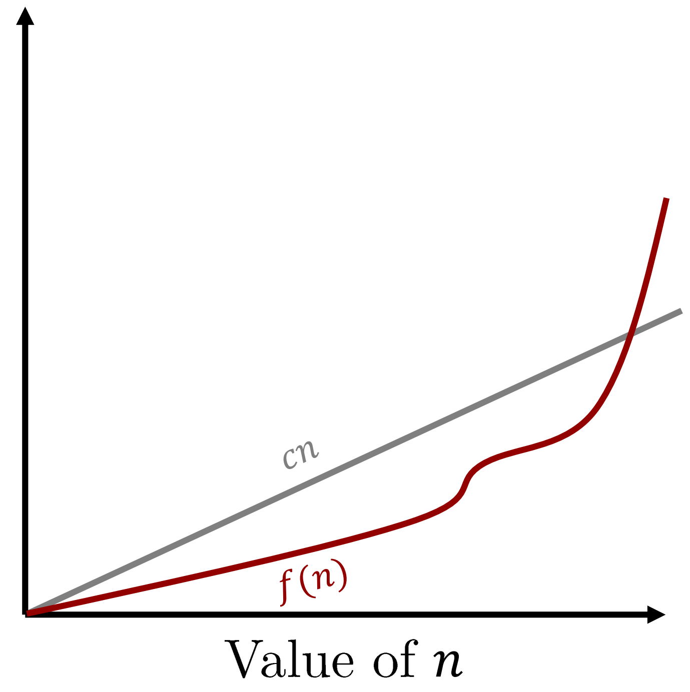
More formally: O

- For a function $f: \mathbb{N} \to \mathbb{N}$, $f(n) = O(n)$ if there exists a constant c such that for all sufficiently large n, $f(n) \leq cn$
- Informally: There is a line that can be drawn that stays above f from some point on



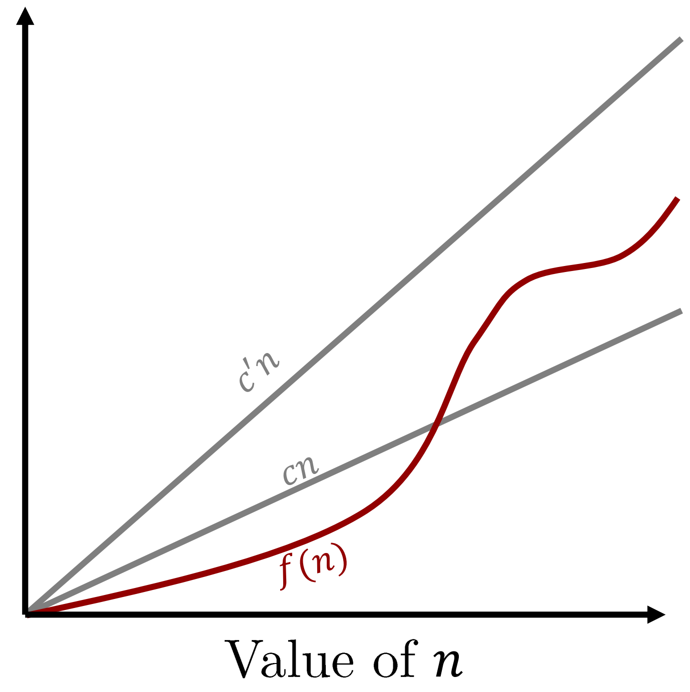
More formally: Ω

- For a function $f\colon \mathbb{N} \rightarrow \mathbb{N}$, $f(n) = \Omega(n)$ if there exists a constant c such that for all sufficiently large n, $f(n) \geq cn$
- Informally: There is a line that can be drawn that stays below f from some point on



More formally: Θ

- For a function $f: \mathbb{N} \to \mathbb{N}$, $f(n) = \Theta(n)$ if $f(n) = O(n)$ and $f(n) = \Omega(n)$
- Informally: f can be sandwiched between two lines from some point on



More formally and generally

- $f(n) = O(g(n))$ if there exists a constant c such that for all sufficiently large n, $f(n) \leq c \cdot g(n)$
- $f(n) = \Omega(g(n))$ if there exists a constant c such that for all sufficiently large n, $f(n) \geq c \cdot g(n)$
- $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

Exercises

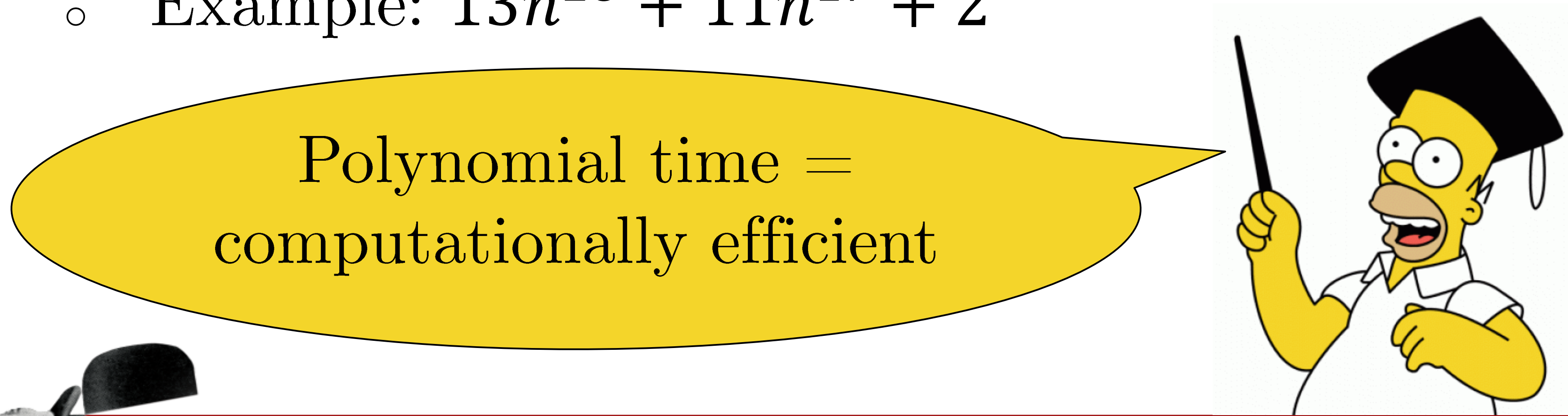
- $n^4 + 3n + 22 = O(n^4)$?
- $n^4 + 3n + 22 = \Omega(n^4 \log n)$?
- Poll 2: Which of the following statements is true:
  1. $\ln n = O(\log_2 n)$
  2. $\ln n = \Omega(\log_2 n)$
  3. Both
  4. Neither

Exercises

- Poll 3: $\log(n!) = ?$
  1. $\Theta(n)$
  2. $\Theta(n \log n)$
  3. $\Theta(n^2)$
  4. $\Theta(2^n)$
- Poll 4: Which of the following statements is true:
  1. $f = O(g)$ and $g = O(h) \Rightarrow f = O(h)$
  2. $f = O(h)$ and $g = O(h) \Rightarrow f = O(g)$
  3. Both
  4. Neither

Names for growth rates

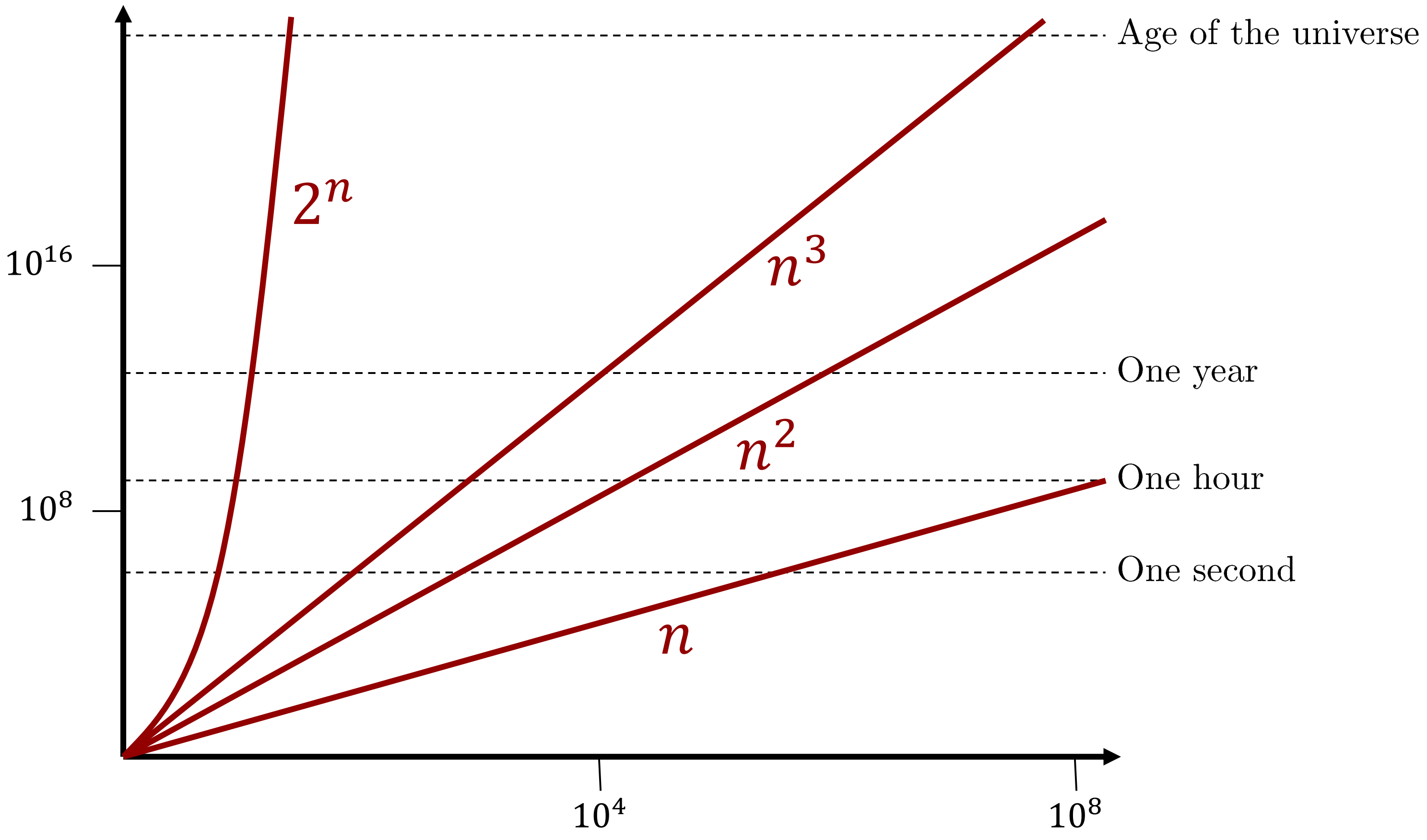
- Linear time: $T(n) = O(n)$
- Quadratic time: $T(n) = O(n^2)$
- Polynomial time: there exists $k \in \mathbb{N}$ such that $T(n) = O(n^k)$
    - Example: $13n^{28} + 11n^{17} + 2$



Names of growth rates

- Exponential time: there exists $k \in \mathbb{N}$ such that $T(n) = O(k^n)$
  - Example: $T(n) = n2^n = O(3^n)$
- Logarithmic time: $T(n) = O(\log n)$
  - A logarithmic-time algorithm can't read all of its input
  - The running time of binary search is logarithmic

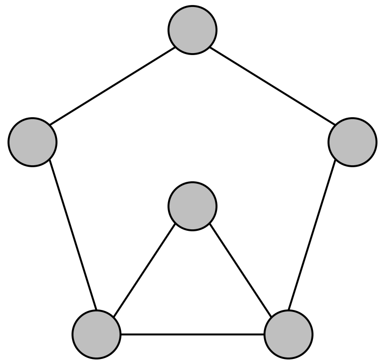
Limits of the possible



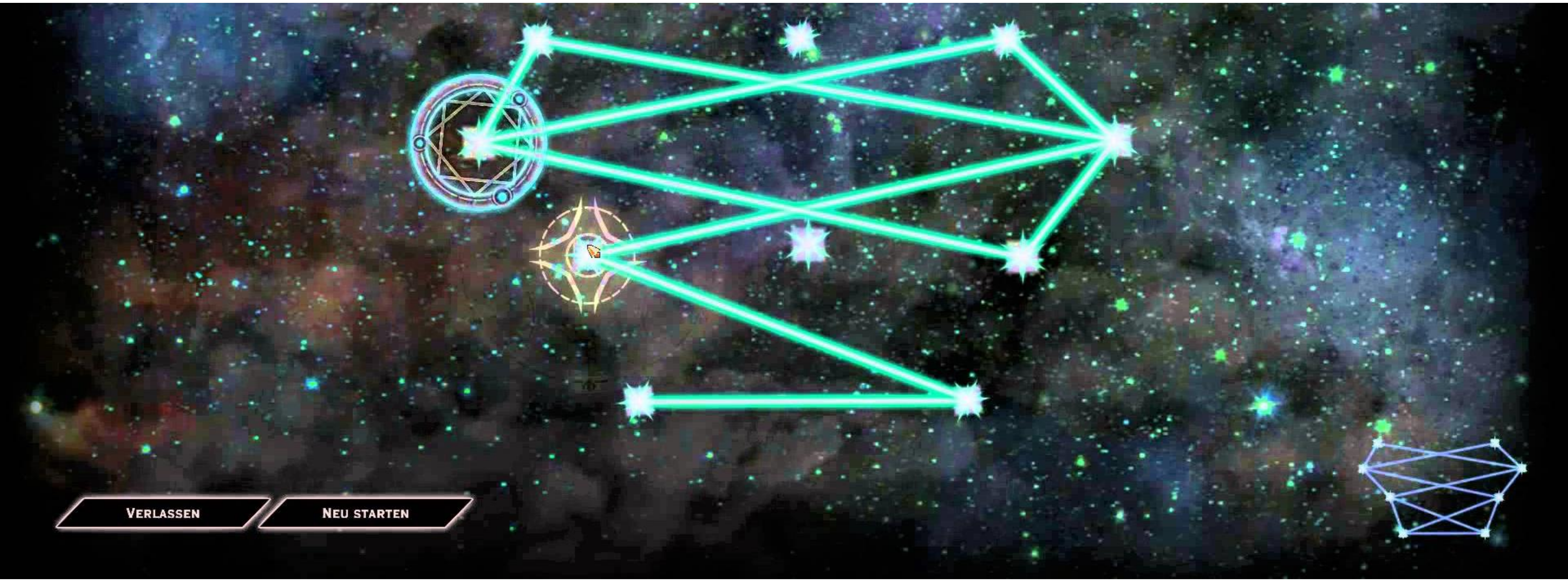
Log-log plot with 1 step = 1μs

Two similar problems

- Eulerian-Cycle:
  - Instance: A connected graph
  - Input size: Number of vertices
  - Question: Is there a tour visiting each edge exactly once?
- Algorithm: The answer is "yes" if and only if each vertex has even degree; complexity $O(n^2)$
- Theorem (Euler): The algorithm correctly solves Eulerian-Cycle



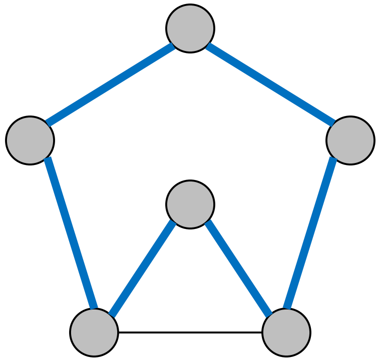
Application: Dragon Age



This is nothing but the EULERIAN-PATH problem!

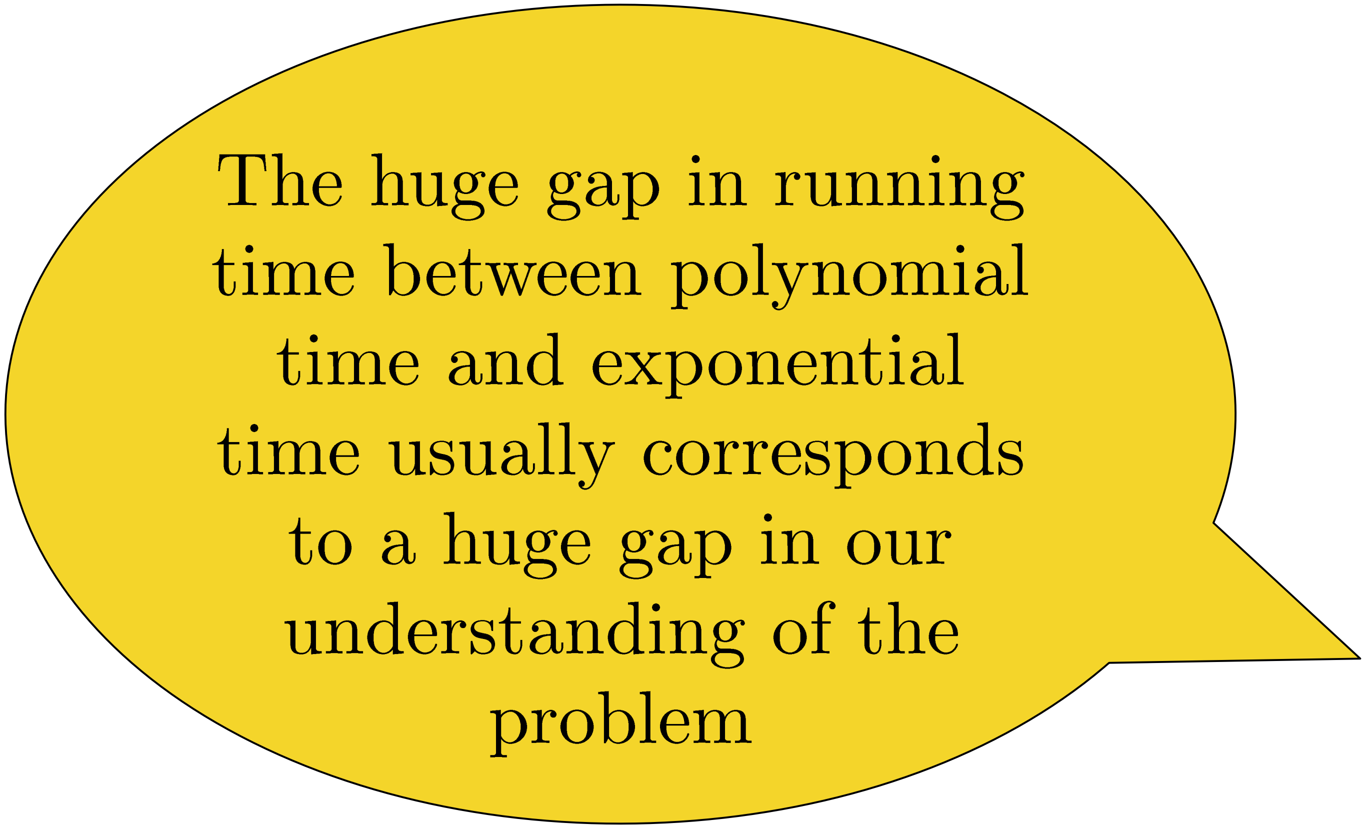
Two similar problems

- Hamiltonian-Cycle:
  - Instance: A connected graph
  - Input size: Number of vertices
  - Question: Is there a tour visiting each vertex exactly once?



- Complexity:
  - Brute force algorithm: $n!$
  - 1970: 2^n
  - 2010: 1.657^n

Polynomial time


The huge gap in running time between polynomial time and exponential time usually corresponds to a huge gap in our understanding of the problem

Representation

- The way a problem is represented can have a huge impact on its complexity
- KNAPSACK:
  - Instance: m items $1, \dots, m$ with values $v_1, \dots, v_m$ and weights $w_1, \dots, w_m$, capacity B, value V
  - Input size: We'll talk about this later
  - Question: Is there a subset of items S such that $\sum_{i \in S} w_i \leq B$ and $\sum_{i \in S} v_i \geq V$

Representation

- Dynamic programming algorithm for Knapsack:
    - $m \times B$ matrix A
    - $A(i,j) = \max\{A(i-1,j), A(i-1,j-w_i)+v_i\}$

Item	value	weight
1	4	5
2	3	2
3	5	2

$\boxed{B = 5}$

	1	2	3	4	5
1	0	0	0	0	4
2	0	3	3	3	4
3	0	5	5	8	8

Capacity allowed

Items allowed

Representation

- Running time of the dynamic programming algorithm: $\Theta(mB)$
- Binary representation for Knapsack:
  - Input size: $n \approx 2m \cdot \max\{\log B, \log V\}$
  - Exponential running time!
- Unary representation for Knapsack:
  - Input size: $n \approx 2m \cdot \max\{B, V\}$
  - Linear running time!

Cool growth rates: 2stack

- $\mathrm{2STACK}(0) = 1$
- $\mathrm{2STACK}(n) = 2^{\mathrm{2STACK}(n-1)}$
- Examples:
  - $\mathrm{2STACK}(1) = 2$
  - $\mathrm{2STACK}(2) = 4$
  - $\mathrm{2STACK}(3) = 16$
  - $\mathrm{2STACK}(4) = 65536$
  - $\mathrm{2STACK}(5) =$ yikes!

$$2^{2^{2^{2^{2^{2^{2^{2^{2}}}}}}}}$$

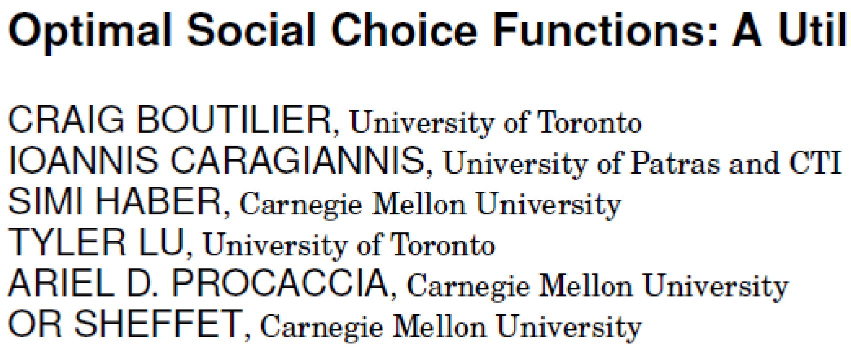
Cool growth rates: log*

- $\log^*(n) = \#$times you have to apply the log function to n to make it ≤ 1
- Examples:

$\text{2STACK}(1) = 2$	$\log^*(2) = 1$
$\text{2STACK}(2) = 4$	$\log^*(4) = 2$
$\text{2STACK}(3) = 16$	$\log^*(16) = 3$
$\text{2STACK}(4) = 65536$	$\log^*(65536) = 4$
$\text{2STACK}(5) = \text{yikes!}$	$\log^*(\text{yikes!}) = 5$

Cool growth rates: log*

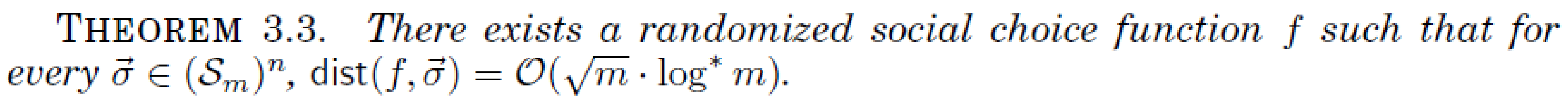
- There's no way **log*** is actually useful, right?
- Multiplication takes $O(n \log n \, 2^{\log^* n})$



Optimal Social Choice Functions: A Utilitarian View

CRAIG BOUTILIER, University of Toronto
IOANNIS CARAGIANNIS, University of Patras and CTI
SIMI HABER, Carnegie Mellon University
TYLER LU, University of Toronto
ARIEL D. PROCACCIA, Carnegie Mellon University
OR SHEFFET, Carnegie Mellon University

(2015)



THEOREM 3.3. *There exists a randomized social choice function f such that for every $\vec{\sigma} \in (\mathcal{S}_m)^n$*, $\mathrm{dist}(f, \vec{\sigma}) = \mathcal{O}(\sqrt{m} \cdot \log^* m)$.

Summary

- Terminology:
  - Big O notation
  - Names for growth rates
- Principles:
  - Why polynomial time?
  - Representation matters