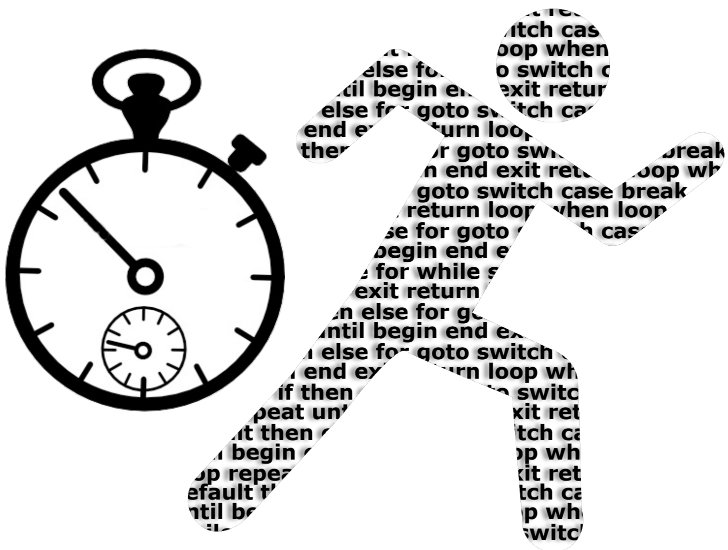
University of Guelph Prof. Pascal Matsakis

CIS2520

4. Analysis of Algorithms



Reading suggestion: Chapter 6 of the textbook

EXPERIMENTAL ANALYSIS

Theoretical Analysis

Examples

Reasonable vs. Unreasonable Algorithms

EXPERIMENTAL ANALYSIS: Principle (1/4) 4.3

1. Implement the algorithm

```
void Sort (List *L) {
       ......
}
```


EXPERIMENTAL ANALYSIS: Principle (2/4) 4.4

2. Use a function like `clock()` to measure the running time

```
#include <time.h>

float analyzeSort (List *L) {
       float sec;
       clock_t t1, t2;
       t1=clock();
       Sort(L);
       t2=clock();
       sec=(t2-t1)/(float)CLOCKS_PER_SEC;
       return sec; // Running time in seconds
}
```

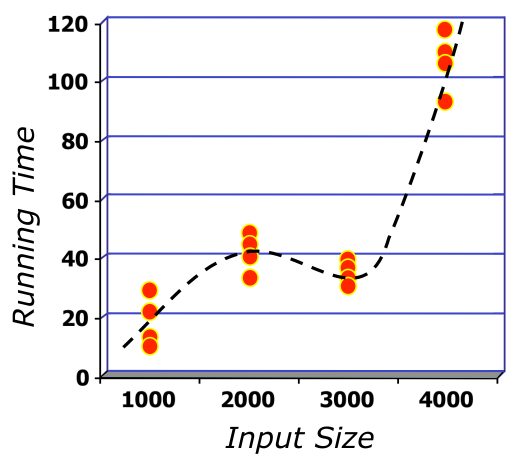
EXPERIMENTAL ANALYSIS: Principle (3/4) 4.5

3. Run the program with inputs of varying size and composition

```
int main (void) {
       List L1, L2, L3...;
       ...... // Initialize L1 to (a1,a2,...,an)
       runtime1=testSort(&L1);
       ...... // Initialize L2 to (b1,b2,...,bn)
       runtime2=testSort(&L2);
       ...... // Initialize L3 to (c1,c2,......,cm)
       runtime3=testSort(&L3);
       ......
}
```


EXPERIMENTAL ANALYSIS: Principle (4/4) 4.6

4. Plot the results



EXPERIMENTAL ANALYSIS: Limitations (1/3) 4.7

1. Implement the algorithm

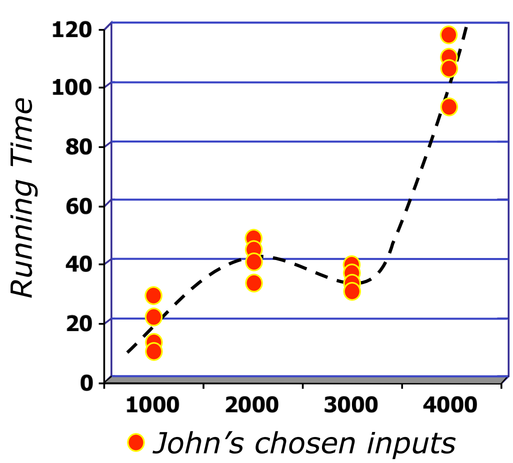
Might be difficult!
Which language?
How optimized?
Which compiler?

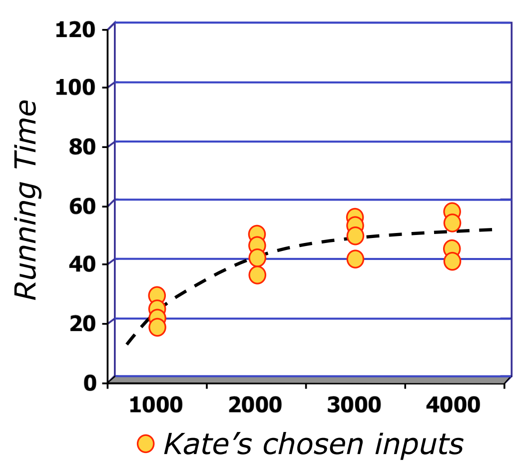
CIS2520 Analysis of Algorithms

EXPERIMENTAL ANALYSIS: Limitations (2/3) 4.8

3. Run the program with inputs of varying size and composition

Which hardware and software environments?
Which inputs?



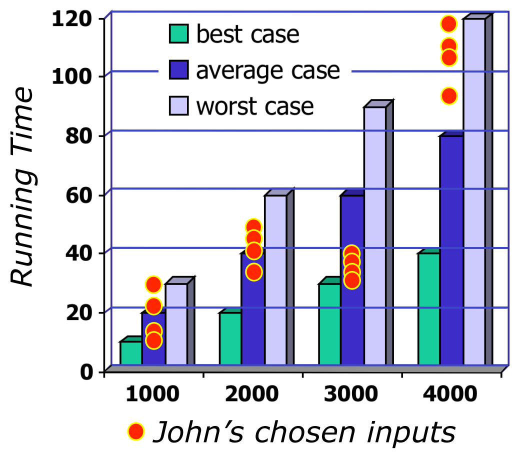


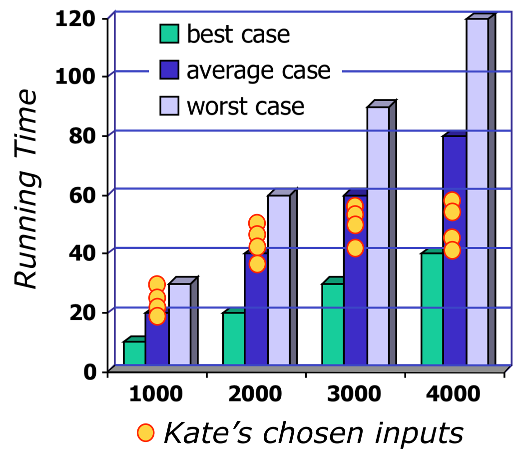
EXPERIMENTAL ANALYSIS: Limitations (3/3) 4.9

3. Run the program with inputs of varying size and composition

Which hardware and software environments?

Which inputs?





Reading suggestion: Chapter 6 of the textbook

Experimental Analysis

THEORETICAL ANALYSIS

Examples

Reasonable vs. Unreasonable Algorithms

THEORETICAL ANALYSIS: Goal 4.11

Evaluate the speed of an algorithm:

✧ without having to implement it
✧ while taking into account all possible inputs
✧ independently of the hardware and software environments

THEORETICAL ANALYSIS: O-Notation (1/6) 4.12

Let S be a subset of $\mathbb{Z}_+$.
S is a **neighborhood of infinity** iff: $\exists m \in \mathbb{Z}_+, m..+\infty \subseteq S$

In the next slides, unless otherwise specified,
a **function** is a function from $\mathbb{Z}_+$ to $\mathbb{R}_+$ defined on a neighborhood of ∞.

$1, \log(n), n, n\log(n), n^2, 2^n, n!, n^n$
$n^2-8n+15, 2\sqrt{n}+7\sin(n)-1, \sqrt{n}+0.5n-10$

Consider two functions f and g from $\mathbb{Z}_+$ to $\mathbb{R}_+$.
If f and g are defined on a neighborhood of ∞
then f+g and fg are defined on a neighborhood of ∞.

THEORETICAL ANALYSIS: O-Notation (2/6) 4.13

Consider two functions f and g.

1) Can we find a neighborhood of infinity S such that f≤g on S?

$1 \le \log(n) \le n \le n\log(n) \le n^2 \le 2^n \le n! \le n^n$
$n^2-8n+15 \le n^2$, $2\sqrt{n}+7\sin(n)-1 \le 3\sqrt{n}$, $\sqrt{n}+0.5n-10 \le n$

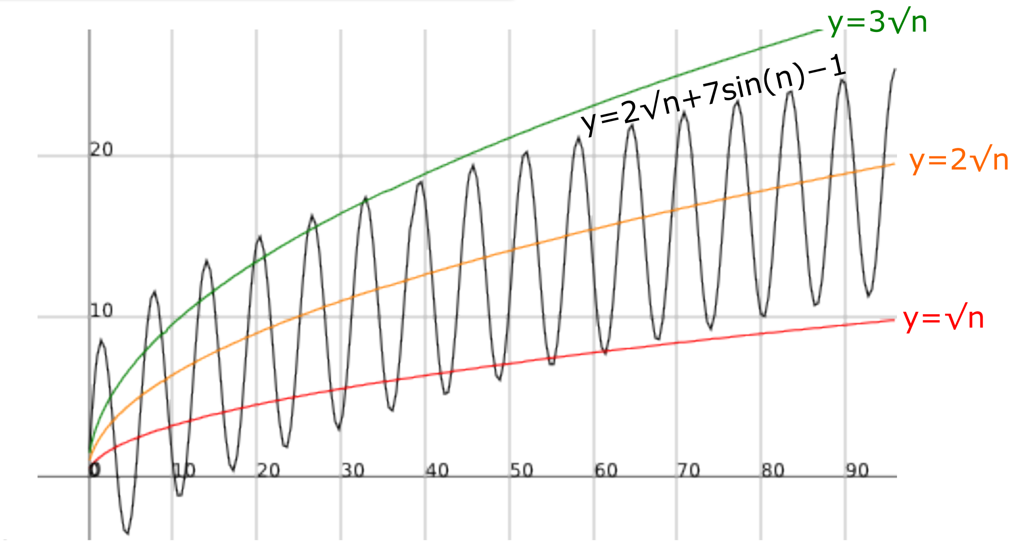
2) Can we find a positive real number λ and a neighborhood of infinity S such that f≤λg on S? If the answer is yes, we say that **f is O(g)**, or that f(n) is O(g(n)).

We say that **f is O(g)** iff: $\exists \lambda \in \mathbb{R}_+, \exists m \in \mathbb{Z}_+, \forall n \in m..+\infty, f(n) \le \lambda g(n)$

n^2 is $O(2^n)$, n is O(n!), n log(n) is O(n log(n))
$n^2-8n+15$ is $O(n^2)$, $2\sqrt{n}+7\sin(n)-1$ is $O(\sqrt{n})$

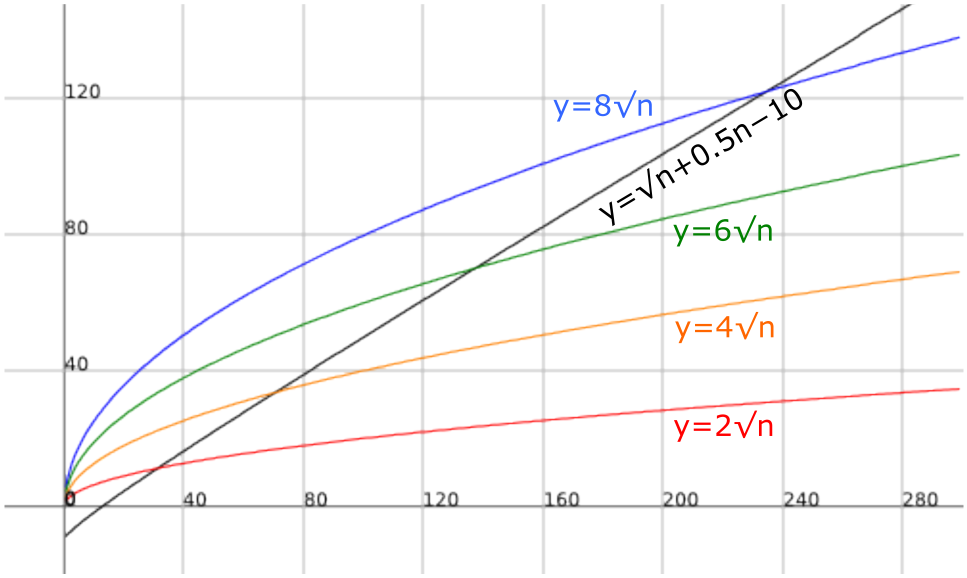
THEORETICAL ANALYSIS: O-Notation (3/6) 4.14

$2\sqrt{n}+7\sin(n)-1$ is $O(\sqrt{n})$



THEORETICAL ANALYSIS: O-Notation (4/6) 4.15

√n+0.5n−10 is not O(√n)



THEORETICAL ANALYSIS: O-Notation (5/6) 4.16

PROPERTIES

f is O(f)

If f≤g on a neighborhood of infinity then f is O(g)

If f is O(g) and g is O(h) then f is O(h)

Assume f_1 is $O(g_1)$ and f_2 is $O(g_2)$:

✧ f_1+f_2 is $O(g_1+g_2)$
✧ f_1f_2 is $O(g_1g_2)$

THEORETICAL ANALYSIS: O-Notation (6/6) 4.17

PROPERTIES

Consider a function f and a real number α:

✧ If $\alpha>0$ then α is O(1)
✧ If $\alpha>0$ then αf is O(f)
✧ If there exists a positive real number ε such that $\varepsilon+\alpha>0$ and $f\geq\varepsilon$ on a neighborhood of infinity then $f+\alpha$ is O(f)

Let $\alpha_0, \alpha_1, \ldots, \alpha_d$ be d+1 real numbers, with $\alpha_d>0$:
$\sum_{i=0}^{d} \alpha_i n^i$ is O(n^d)

THEORETICAL ANALYSIS: Pseudocode (1/2) 4.18

Algorithms often described as programs written in a **pseudocode**.
Pseudocode similar to C, C++, Java, Python, Pascal:

```
function Foo(A,k)
          for i=0 to k
                    for j=k downto 0
                              ......
          i=A.length-k
          while A[i]<0 or A[i]≥1
                    ......
          Reverse(A)
          b=IsPrime(k)
          ......
          if A[0]≠0 and b=true
          then   ......
          elseif ......
          else   ......
          ......
          return A
```

THEORETICAL ANALYSIS: Pseudocode (2/2) 4.19

Algorithms often described as programs written in a **pseudocode**.
Pseudocode similar to C, C++, Java, Python, Pascal. However:

- ✧ intended for human reading rather than machine reading
- ✧ English descriptions and mathematical notation are allowed
- ✧ details not essential for human understanding are ignored (e.g., variable declarations, data abstraction, error handling)

```
// note that there is no standard for pseudocode syntax
let h be the first prime greater than k
```

$$x=\sqrt{\frac{k+1}{h}}$$

```
min=+∞
```


THEORETICAL ANALYSIS: Running Time (1/2) 4.20

n.............................. input size of the algorithm

primitive operation....... indexing into an array, evaluating an expression, assigning a value to a variable, calling or returning from a function...

f(n)........................... worst-case running time, i.e., time *tmax* taken by slowest primitive operation × worst-case number of primitive operations

```
function ArrayMax(A)
        currentMax=A[0]              // 2 primitive operations
        for i=1 to A.length−1        // 2(n−1)+5
                if A[i]>currentMax   // 2(n−1)
                then currentMax=A[i] // 2(n−1)
        return currentMax            // 1
                                     // ---------------
                                     // f(n)=(6n+2)tmax
```

THEORETICAL ANALYSIS: Running Time (2/2) 4.21

If you know that (i) f is O(g) and f is O(h)
and (ii) g is O(h)
and (iii) h is not O(g), or
h is O(g) but g's expression is "simpler" than h's

say: "the algorithm runs in O(g) time"

```
function ArrayMax(A)
      currentMax=A[0]
      for i=1 to A.length−1
            if A[i]>currentMax
            then currentMax=A[i]
      return currentMax
      // This algorithm runs in O(n) time.
```

Reading suggestion: Chapter 6 of the textbook

Experimental Analysis
Theoretical Analysis
EXAMPLES
Reasonable vs. Unreasonable Algorithms

EXAMPLES: Constant O(1) 4.23

Pushing and popping (stacks),
enqueueing and dequeueing (queues)

EXAMPLES: Logarithmic O(log(n)) 4.24

Searching for an element in a sorted array
(binary search)

Look for 50 among 10 elements:	8	12	39	44	45	53	59	71	72	77
Compare 50 with middle element:	8	12	39	44	45	53	59	71	72	77
Look for 50 among 5 elements:	8	12	39	44	45	53	59	71	72	77
Compare 50 with middle element:	8	12	39	44	45	53	59	71	72	77
Look for 50 among 2 elements:	8	12	39	44	45	53	59	71	72	77
Compare 50 with middle element:	8	12	39	44	45	53	59	71	72	77
Look for 50 among 0 elements:	8	12	39	44	45	53	59	71	72	77

50 not found

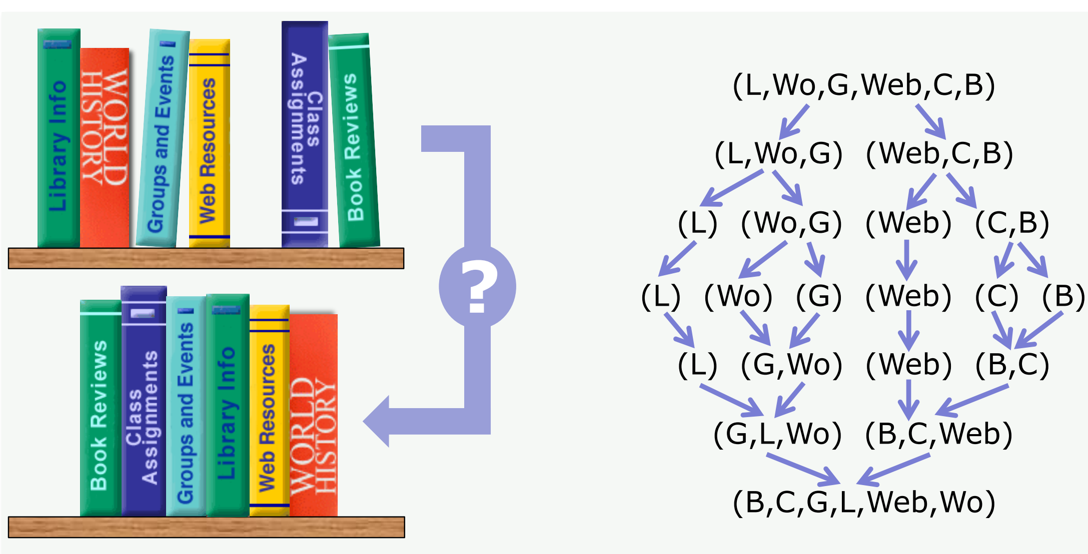
Reading suggestion: Chapter 6 of the textbook

EXAMPLES: Linear O(n) 4.25

Finding the maximum value in an array

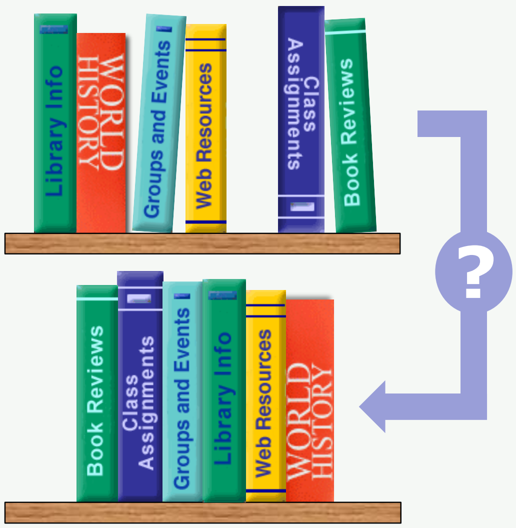
EXAMPLES: Linearithmic O(n log(n)) 4.26

Sorting a list
(mergesort)



EXAMPLES: Quadratic $O(n^2)$ 4.27

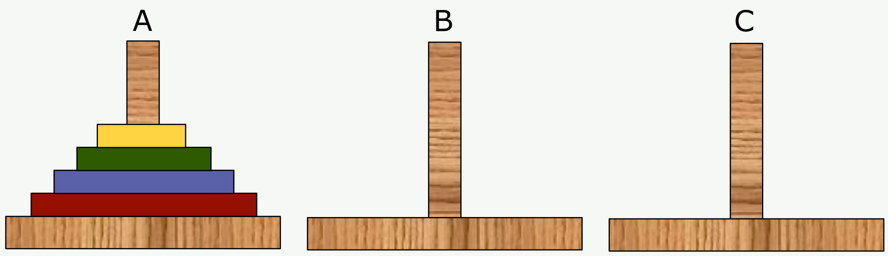
Sorting a list
(bubblesort)



(L,Wo,G,Web,C,B)
(L,**Wo**,G,Web,C,B)
(L,G,**Wo**,Web,C,B)
(L,G,Web,**Wo**,C,B)
(L,G,Web,C,**Wo**,B)
(L,G,Web,C,B,**Wo**)
(G,L,**Web**,C,B,Wo)
(G,L,**Web**,C,B,Wo)
(G,L,C,**Web**,B,Wo)
......
(B,C,G,L,Web,Wo)

EXAMPLES: Exponential $O(2^n)$ 4.28

Solving the Towers of Hanoi puzzle



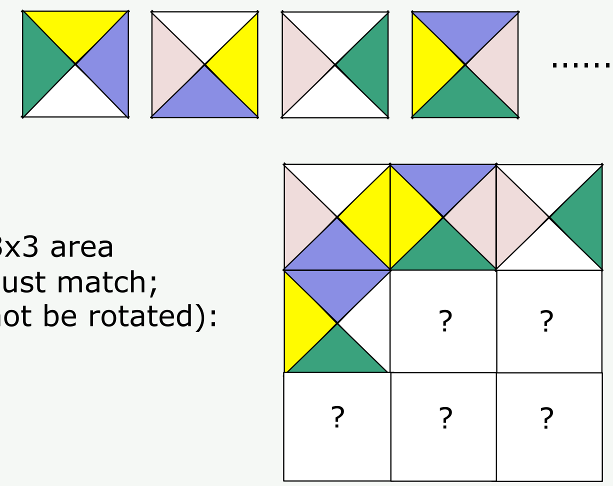
3 pegs and 4 disks = 15 moves

EXAMPLES: Factorial O(n!) 4.29

Solving the Bounded Tiling problem

9 tiles:

Cover a 3x3 area
(edges must match;
tiles cannot be rotated):



EXAMPLES: Polynomial vs. Exponential 4.30

Polynomial	Constant	O(1)	Pushing, popping, enqueuing, dequeuing
	Logarithmic	O(log(n))	Binary search
	Linear	O(n)	Finding the maximum value in an array
	Linearithmic	O(n log(n))	Mergesort
	Quadratic	O(n^2)	Bubblesort
Exponential	Exponential	O(2^n)	Solving the Towers of Hanoi puzzle
	Factorial	O(n!)	Solving the Bounded Tiling problem

Experimental Analysis
Theoretical Analysis
Examples

REASONABLE VS. UNREASONABLE ALGORITHMS

(UN?)REASONABLE: $O(n^2)$ vs. $O(n \log(n))$ 4.32

Bubblesort: $O(n^2)$

250,000,000 items to sort
1,000,000,000 operations/second

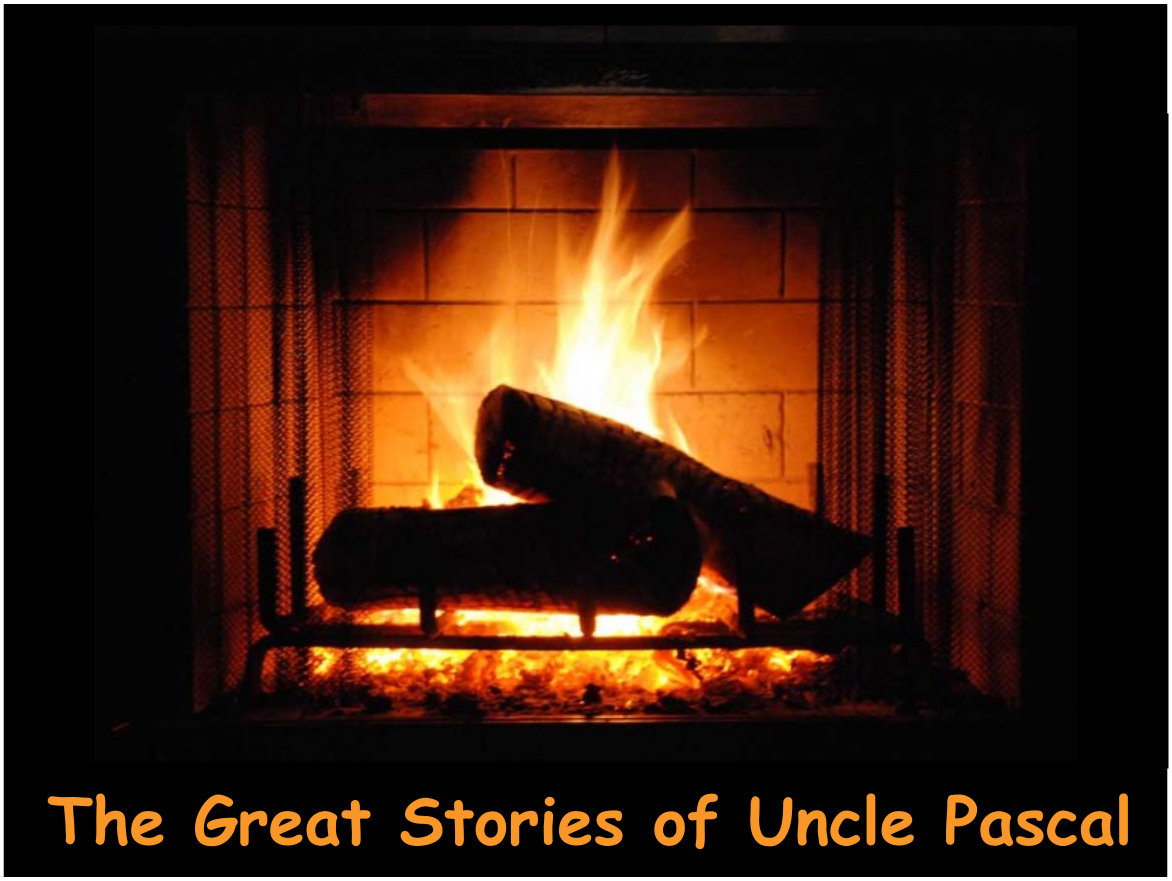
≈ **2 years**

Mergesort: $O(n \log(n))$

250,000,000 items to sort
1,000,000,000 operations/second

≈ **5 seconds**

Reading suggestion: Chapter 6 of the textbook



(UN?)REASONABLE: $O(2^n)$ vs. $O(n^2)$ 4.34

The king and the peasant: $O(2^n)$

64 squares of a chessboard
1,000,000,000 pieces of grain/second

≈ **1000 years**

Some other story: $O(n^2)$

64 squares of a chessboard
1,000,000,000 pieces of grain/second

≈ **0.0001 second**

(UN?)REASONABLE: O(n!) vs. O(2^n) 4.35

Bounded Tiling problem: O(n!)

25 tiles (on a 5x5 area)
1,000,000,000 permutations/second

≈ **500,000,000 years**

Towers of Hanoi puzzle: O(2^n)

25 disks (and 3 pegs)
1,000,000,000 moves/second

≈ **0.03 second**

(UN?)REASONABLE: Fastest Computer? 4.36

Bounded Tiling problem: O(n!)

25 tiles (on a 5x5 area)
8,000,000,000,000,000 permutations/second

≈ **60 years**

(UN?)REASONABLE: (In?)Tractable

reasonable — **Polynomial**

Constant	$O(1)$
Logarithmic	$O(\log(n))$
Linear	$O(n)$
Linearithmic	$O(n \log(n))$
Quadratic	$O(n^2)$

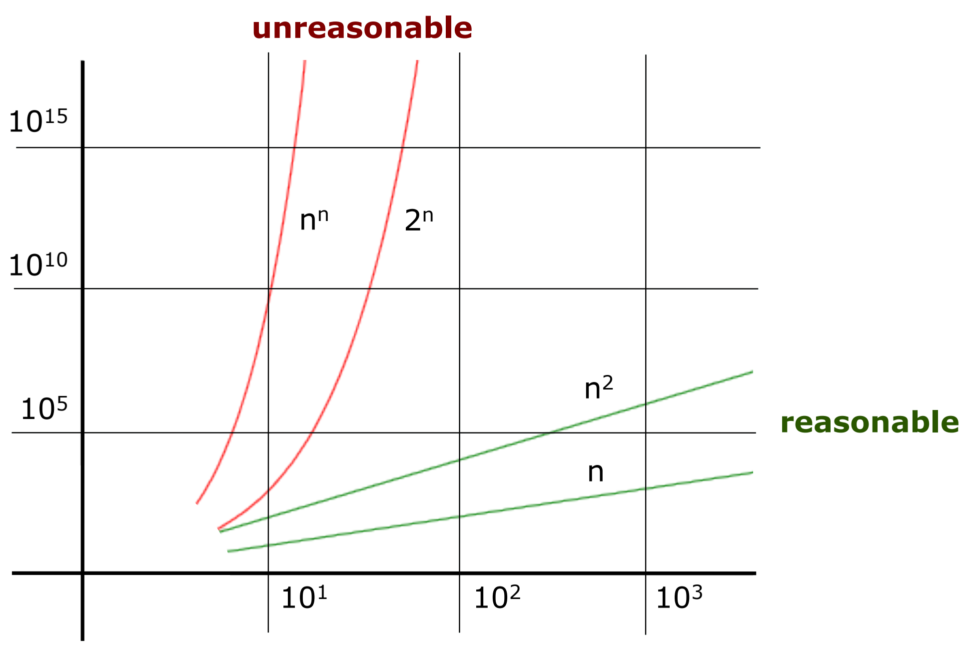
Tractable problem: can be solved by a polynomial-time algorithm

unreasonable — **Exponential**

Exponential	$O(2^n)$
Factorial	$O(n!)$

Intractable problem: cannot be solved by a polynomial-time algorithm

(UN?)REASONABLE: In Pictures



(UN?)REASONABLE: Unsolvable? 4.39

Is an intractable problem solvable?

- ✧ Optimize the exponential-time algorithm
- ✧ Incorporate heuristics
- ✧ Solve a simpler version of the problem
- ✧ Use a polynomial-time probabilistic algorithm
  (answer is the right one only with a certain probability)
- ✧ Use a polynomial-time approximation algorithm
  (solution found might not be the best)

Reading suggestion: Chapter 6 of the textbook

Experimental Analysis
Theoretical Analysis
Examples
Reasonable vs. Unreasonable Algorithms

THE END