

ALGORITHM ANALYSIS

REMEMBER

“An algorithm is a function that takes an input or set of inputs and produces an output or set of outputs”

And when we talk about Algorithms run by computers, AKA Software:

- Algorithms should be correct
- Algorithms should be **EFFICIENT**

SUBJECTS UNDER ANALYSIS



THE TIME FUNCTION: $T(n)$

```
def calculate(n):  
    x = 0  
    for i in range(n):  
        for j in range(n):  
            x += i * j  
    return x
```

THE TIME FUNCTION: $T(n)$

- Describes the running time of an algorithm based on the size of its input:
 - Running time, or number of operations executed
 - Input size, depends on the problem to solve
- Each sentence of our algorithm takes a constant time “ c ”
- Define “ n ” as the size of the input



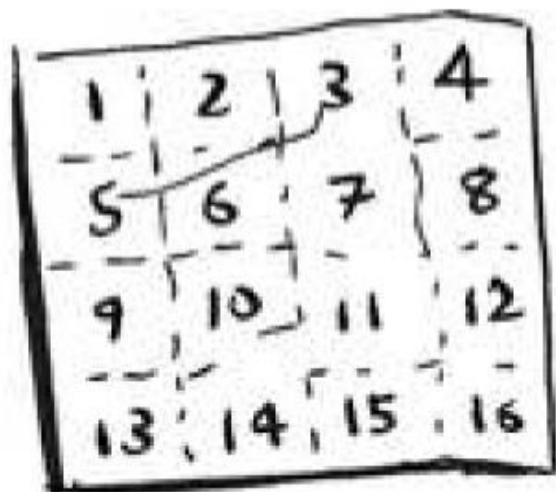
ASYMPTOTIC EQUIVALENCE

- Def: $f(n) \sim g(n)$
- $\lim_{n \rightarrow \infty} \left(\frac{f(n)}{g(n)} \right) = 1$

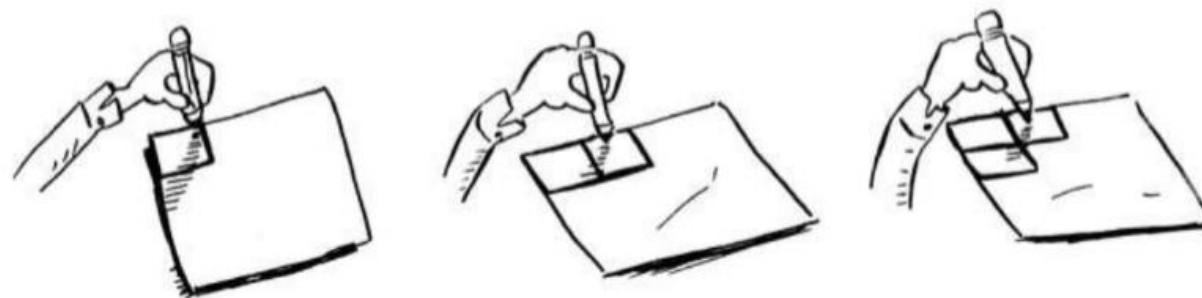
ORDERS OF GROWTH: BIG O NOTATION

- Big O notation describes how fast an algorithm is. More specifically, the rate at which the runtime grows with the size of the input.
- It **simplifies** the Time function, considering only the most significant terms.
- Algorithm speed is not measured in seconds, but in the growth of the number of operations.
- It's formal definition is used to represent worst case scenarios. For best case scenarios the formal notation is the Ω notation (omega). But for the sake of simplicity we will use always the Big O notation, indicating also the worst, best and average case scenarios.

ORDERS OF GROWTH



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

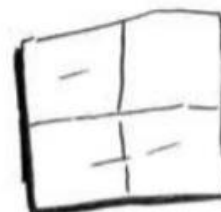


Drawing a grid
one box at a time

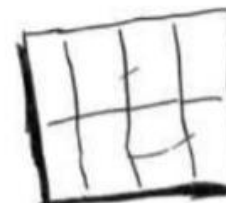
1 FOLD



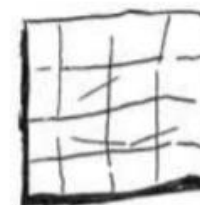
2 FOLDS



3 FOLDS



4 FOLDS



Drawing a grid
in four folds

BEWARE OF LOGARITHMS

- Remember: Logarithms are the **flip** of exponential operations

$$10^2 = 100 \leftrightarrow \log_{10} 100 = 2$$

$$10^3 = 1000 \leftrightarrow \log_{10} 1000 = 3$$

$$2^3 = 8 \leftrightarrow \log_2 8 = 3$$

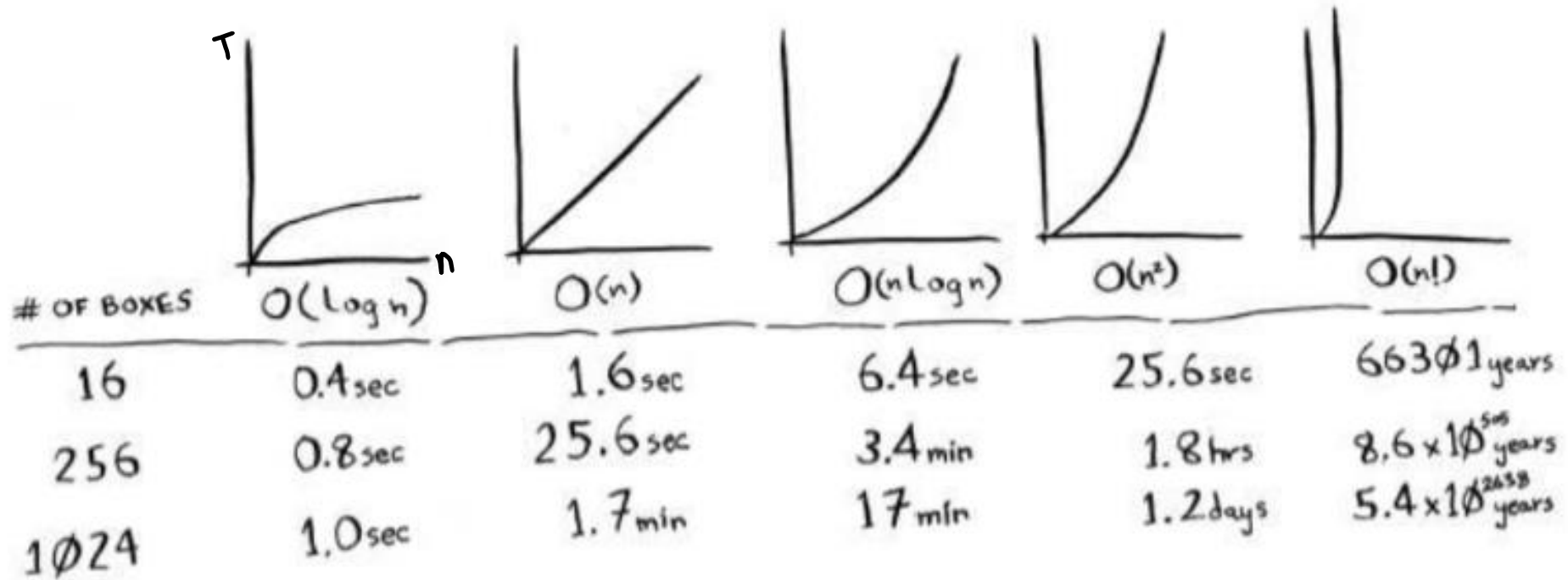
$$2^4 = 16 \leftrightarrow \log_2 16 = 4$$

$$2^5 = 32 \leftrightarrow \log_2 32 = 5$$

ORDER OF GROWTH: TYPICAL CASES



FAST



SLOW

QUICK EXERCISES

- What's the runtime of the following scenarios in terms of Big O?
 - You have a name and you want to find the person's phone number in the phone book
 - You have a phone number and you want to find the person's name in the phone book

PRIME CALCULATOR

- Code a function to calculate whether a number is prime or not.
- What's its time complexity?

EXERCISE: INVERSIONS

- Let $A[1..n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an ***inversion*** of A .
 - Code a function to list the number of inversions present in a given array **input**.
 - Test it using the array $[2, 3, 8, 6, 1]$, and verify your code lists five inversions.
 - What array with the elements from the set $\{1, 2, \dots, n\}$ has the most inversions?
 - Identify its time complexity: the best case and worst case scenarios in terms of Big O.

IMPROVE THE PRIME CALCULATOR

- The Sieve of Eratosthenes can provide better running times for our prime calculator algorithm.
Implement a new prime calculator and analyse its time complexity.

	2	3	4	5	6	7	8	9	10	Prime numbers
11	12	13	14	15	16	17	18	19	20	
21	22	23	24	25	26	27	28	29	30	
31	32	33	34	35	36	37	38	39	40	
41	42	43	44	45	46	47	48	49	50	
51	52	53	54	55	56	57	58	59	60	
61	62	63	64	65	66	67	68	69	70	
71	72	73	74	75	76	77	78	79	80	
81	82	83	84	85	86	87	88	89	90	
91	92	93	94	95	96	97	98	99	100	
101	102	103	104	105	106	107	108	109	110	
111	112	113	114	115	116	117	118	119	120	

Source: https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes