INTRODUCTION TO ALGORITHMS: COMPUTATIONAL COMPLEXITY

BY VASYL NAKVASIUK, 2013

tech hangout by Innovecs

WHAT IS AN ALGORITHM?

WHAT IS AN ALGORITHM?

> *“ An algorithm is a procedure that takes any of the possible input instances and transforms it to the desired output. ”*

Important issues: correctness, elegance and efficiency.

EFFICIENCY

Is this really necessary?

CRITERIA OF EFFICIENCY:

- Time complexity
- Space complexity

Time complexity ≠ Space complexity ≠ Complexity of algorithm

HOW CAN WE MEASURE COMPLEXITY?

HOW CAN WE MEASURE COMPLEXITY?

EMPIRICAL ANALYSIS (BENCHMARKS)

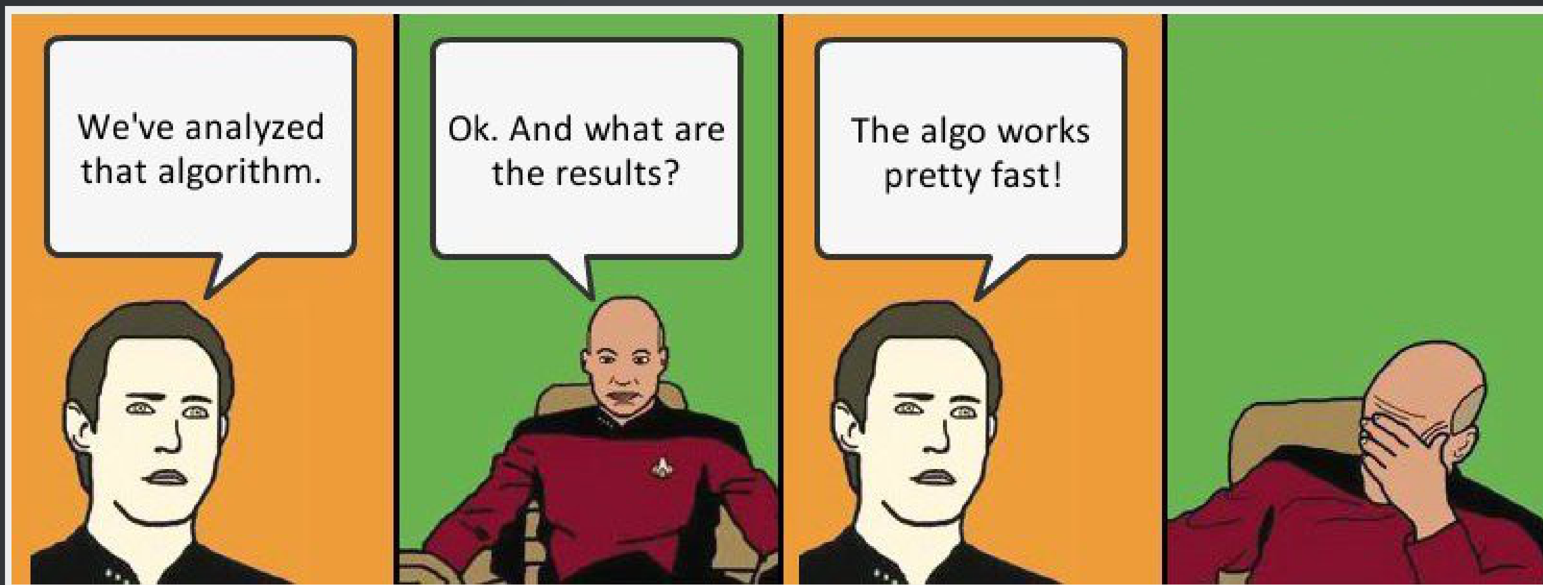
THEORETICAL ANALYSIS (ASYMPTOTIC ANALYSIS)

BENCHMARKS

EMPIRICAL ANALYSIS

BENCHMARKS

VERSION #1



WHAT MEANS “FAST”?

BENCHMARKS

VERSION #2

```
import time

start = time.time() # Return the time in seconds since the epoch.
my_algo(some_input)
end = time.time()

print(end - start)
```

```
0.048032498359680176
```

BENCHMARKS

VERSION #3

```
import timeit

timeit.timeit('my_algo(some_input)', number=1000)
```

```
1000 loops, best of 3: 50.3 ms per loop
```

BENCHMARKS

VERSION #4

```
import timeit

inputs = [1000, 10000, 500000, 1000000]

for input in inputs:
    timeit.timeit('my_algo(input)', number=1000)
```

```
list of 1000 items:
1000 loops, best of 3: 50.3 ms per loop

list of 10000 items:
1000 loops, best of 3: 104.7 ms per loop

list of 500000 items:
1000 loops, best of 3: 459.1 ms per loop

list of 1000000 items:
1000 loops, best of 3: 3.12 s per loop
```

BENCHMARKS

VERSION #5

```
# Intel Core i7-3970X @ 3.50GHz, RAM 8 Gb, Ubuntu 12.10 x64, Python 3.

import timeit

inputs = [1000, 10000, 500000, 1000000]

for input in inputs:
    timeit.timeit('my_algo(input)', number=1000)
```

```
list of 1000 items:
1000 loops, best of 3: 50.3 ms per loop

list of 10000 items:
1000 loops, best of 3: 104.7 ms per loop

list of 500000 items:
1000 loops, best of 3: 459.1 ms per loop

list of 1000000 items:
1000 loops, best of 3: 3.12 s per loop
```

EXPERIMENTAL STUDIES HAVE SEVERAL LIMITATIONS:

- It is necessary to implement and test the algorithm in order to determine its running time.
- Experiments can be done only on a limited set of inputs, and may not be indicative of the running time on other inputs not included in the experiment.
- In order to compare two algorithms, the same hardware and software environments should be used.

ASYMPTOTIC ANALYSIS

THEORETICAL ANALYSIS

ASYMPTOTIC ANALYSIS

EFFICIENCY AS A FUNCTION OF INPUT SIZE

$T(n)$ – running time as a function of n, where n – size of input.

$n \to \infty$

Random-Access Machine (RAM)

BEST, WORST, AND AVERAGE-CASE COMPLEXITY

LINEAR SEARCH

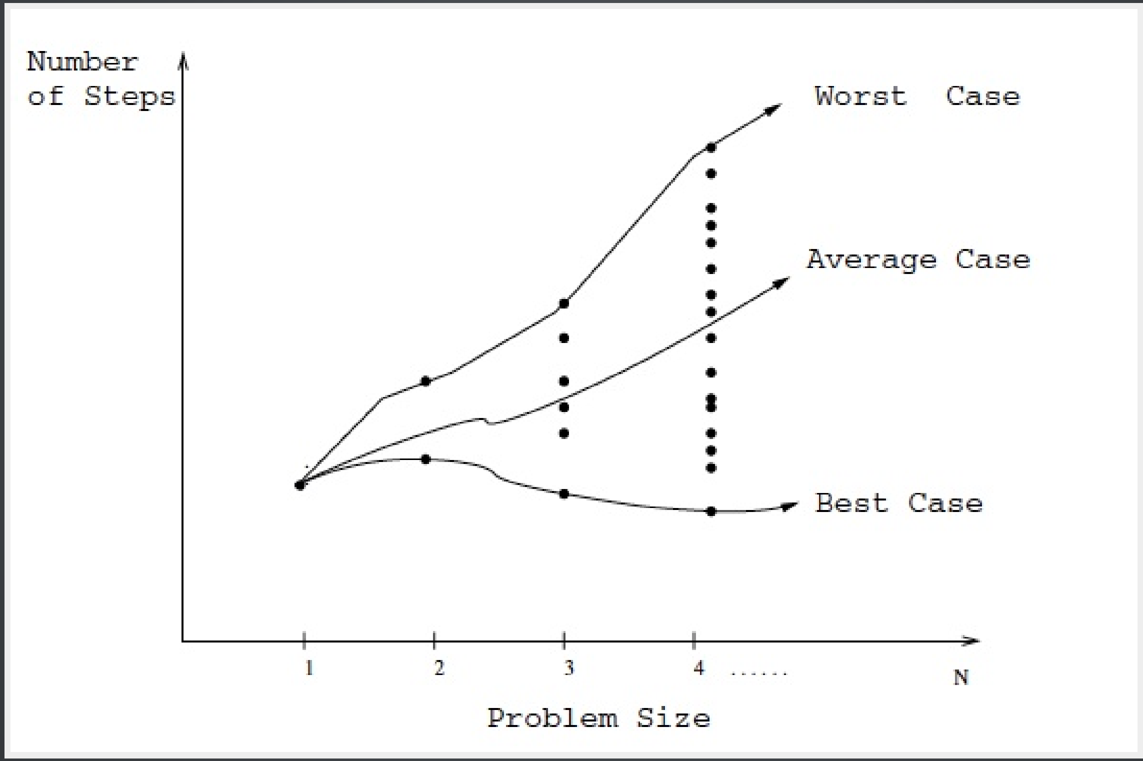
```
def linear_search(my_item, items):
    for position, item in enumerate(items):
        if my_item == item:
            return position
```

$T(n) = n$?

$T(n) = 1/2 \cdot n$?

$T(n) = 1$?

BEST, WORST, AND AVERAGE-CASE COMPLEXITY



BEST, WORST, AND AVERAGE-CASE COMPLEXITY

LINEAR SEARCH

```
def linear_search(my_item, items):
    for position, item in enumerate(items):
        if my_item == item:
            return position
```

Worst case: $T(n) = n$

Average case: $T(n) = 1/2 \cdot n$

Best case: $T(n) = 1$

$T(n) = O(n)$

HOW CAN WE COMPARE TWO FUNCTIONS?

WE CAN USE ASYMPTOTIC NOTATION

ASYMPTOTIC NOTATION

THE BIG OH NOTATION

ASYMPTOTIC UPPER BOUND

$O(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \le f(n) \le c \cdot g(n)$ for all $n \ge n_0\}$

$$T(n) \in O(g(n))$$

or

$$T(n) = O(g(n))$$

Ω-NOTATION

ASYMPTOTIC LOWER BOUND

$\Omega(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \le c \cdot g(n) \le f(n)$ for all $n \ge n_0\}$

$T(n) \in \Omega(g(n))$

or

$T(n) = \Omega(g(n))$

Θ-NOTATION
ASYMPTOTIC TIGHT BOUND

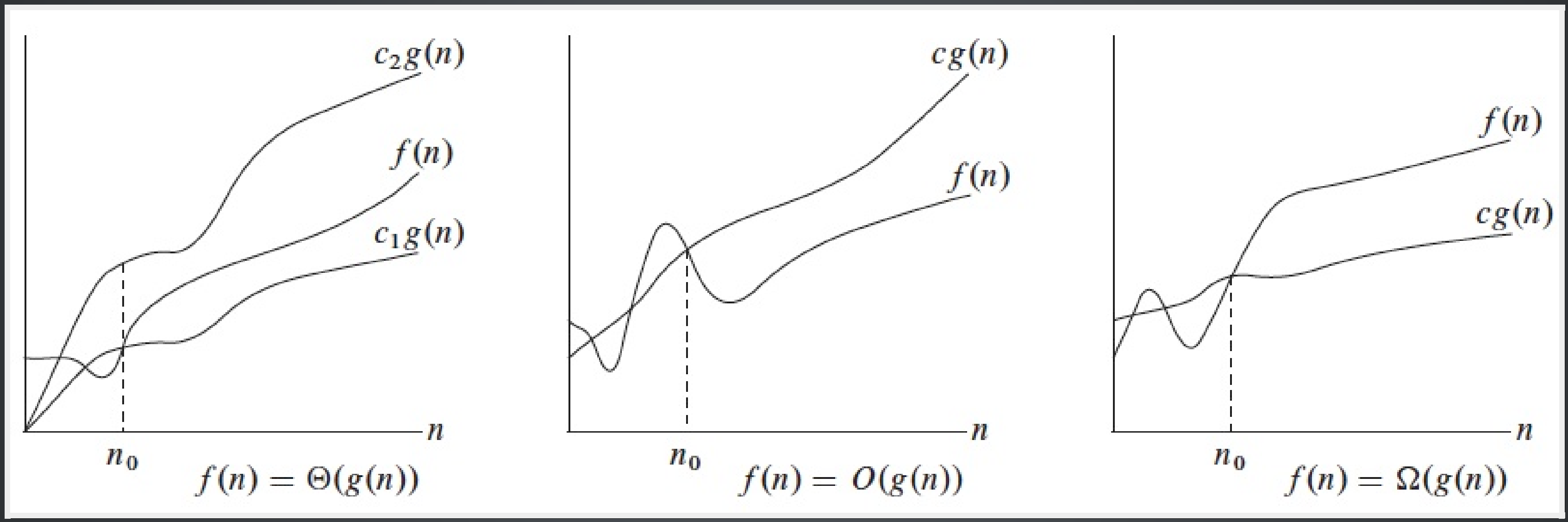
$\Theta(g(n))$ = {$f(n)$: there exist positive constants c_1, c_2 and n_0 such that $0 \le c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n)$ for all $n \ge n_0$}

$T(n) \in \Theta(g(n))$

or

$T(n) = \Theta(g(n))$

GRAPHIC EXAMPLES OF THE Θ, O AND Ω NOTATIONS



EXAMPLES

$3 \cdot n^2 - 100 \cdot n + 6 = O(n^2)$,
because we can choose $c = 3$ and
$3 \cdot n^2 > 3 \cdot n^2 - 100 \cdot n + 6$

$100 \cdot n^2 - 70 \cdot n - 1 = O(n^2)$,
because we can choose $c = 100$ and
$100 \cdot n^2 > 100 \cdot n^2 - 70 \cdot n - 1$

$3 \cdot n^2 - 100 \cdot n + 6 \approx 100 \cdot n^2 - 70 \cdot n - 1$

LINEAR SEARCH

LINEAR SEARCH (VILLARRIBA VERSION):

$$T(n) = O(n)$$

LINEAR SEARCH (VILLABAJO VERSION)

```
def linear_search(my_item, items):
    for position, item in enumerate(items):
        print('poition - {0}, item - {0}'.format(position, item))
        print('Compare two items.')
        if my_item == item:
            print('Yeah!!!')
            print('The end!')
            return position
```

$$T(n) = O(3 \cdot n + 2) = O(n)$$

Speed of "Villarriba version" ≈ Speed of "Villabajo version"

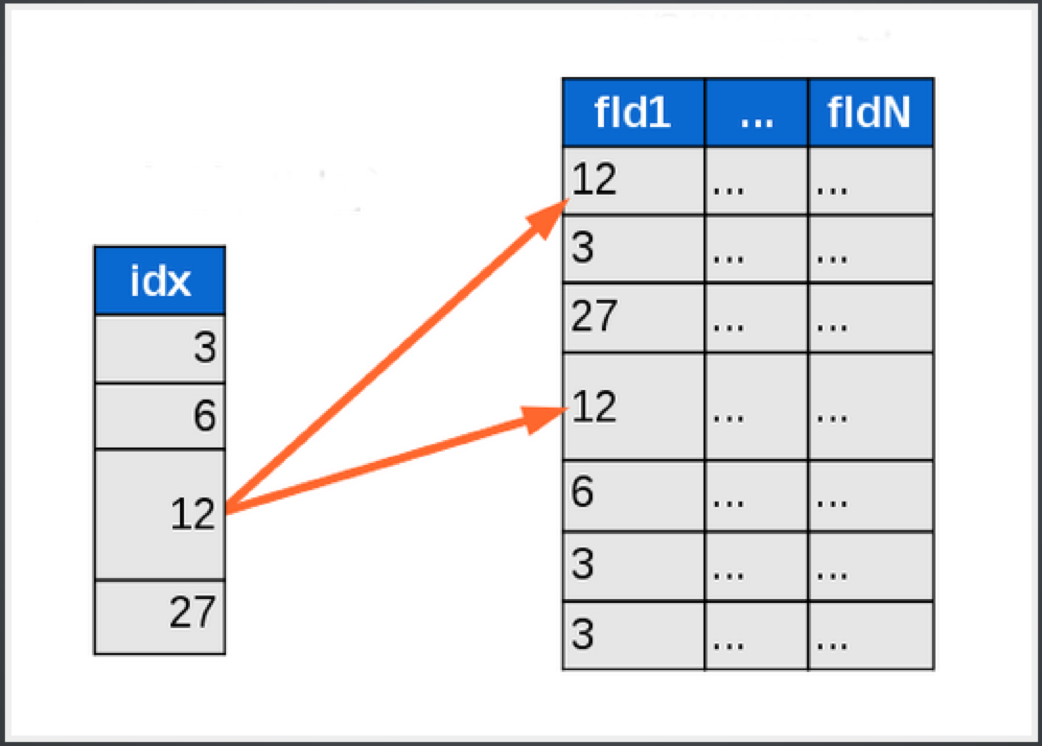
BINARY SEARCH

```
def binary_search(seq, t):
    min = 0; max = len(seq) - 1
    while 1:
        if max < min:
            return -1
        m = (min + max) / 2
        if seq[m] < t:
            min = m + 1
        elif seq[m] > t:
            max = m - 1
        else:
            return m
```

$$T(n) = O(\log(n))$$

PRACTICAL USAGE

ADD DB “INDEX”



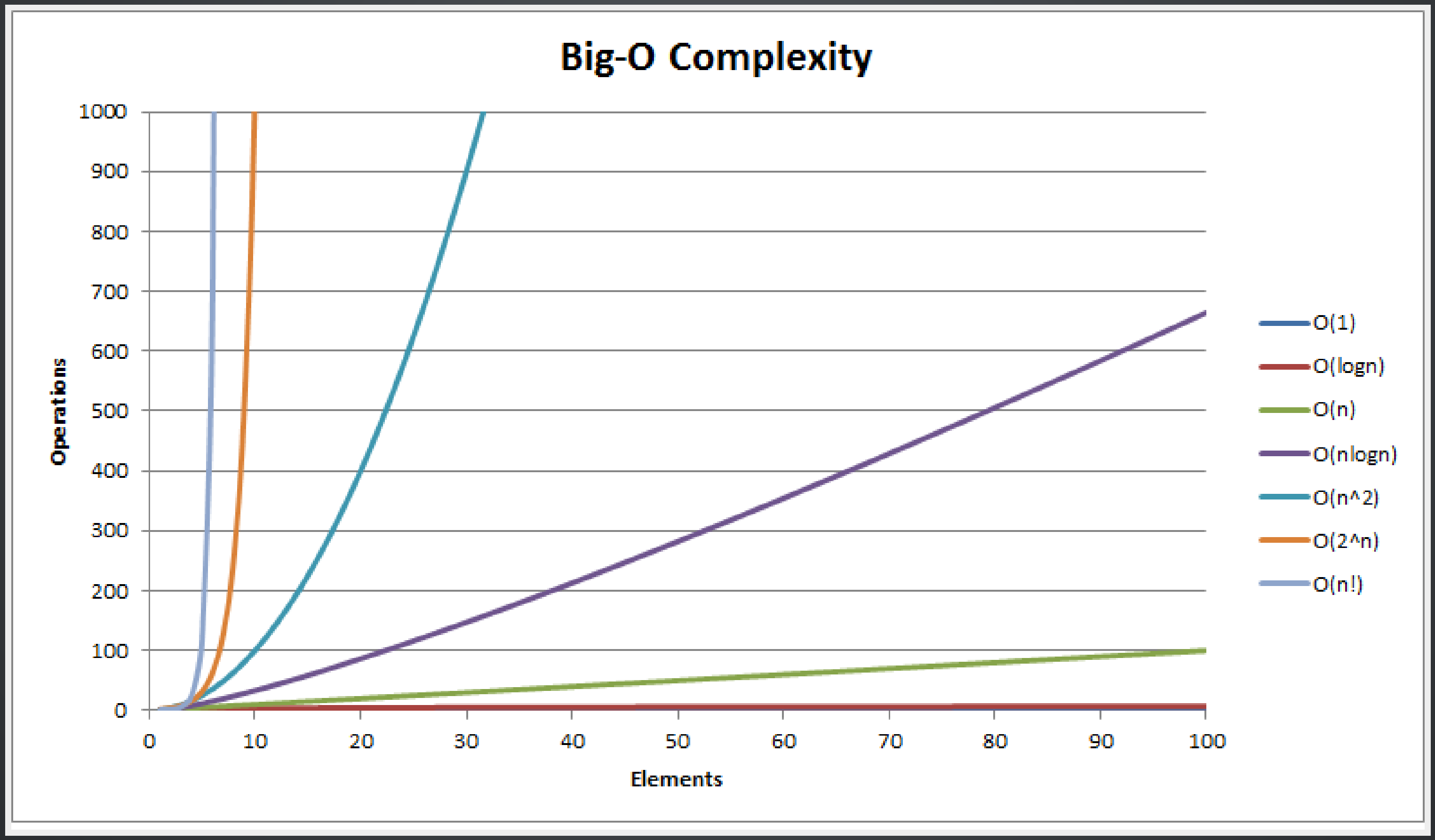
TYPES OF ORDER

Notation	Name
$O(1)$	Constant
$O(\log(n))$	Logarithmic
$O(\log(\log(n))$	Double logarithmic (iterative logarithmic)
$o(n)$	Sublinear
$O(n)$	Linear
$O(n\log(n))$	Loglinear, Linearithmic, Quasilinear or Supralinear
$O(n^2)$	Quadratic
$O(n^3)$	Cubic
$O(n^c)$	Polynomial (different class for each $c > 1$)
$O(c^n)$	Exponential (different class for each $c > 1$)
$O(n!)$	Factorial
$O(n^n)$	- (Yuck!)

However, all you really need to understand is that:

$$n! \gg 2^n \gg n^3 \gg n^2 \gg n \cdot \log(n) \gg n \gg \log(n) \gg 1$$

THE BIG OH COMPLEXITY FOR DIFFERENT FUNCTIONS



GROWTH RATES OF COMMON FUNCTIONS MEASURED IN NANOSECONDS

Each operation takes one nanosecond (10^{-9} seconds).

n $f(n)$	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 years
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.4×10^{15} yrs
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 days	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} yrs	
1,000	0.010 μs	1.00 μs	9.966 μs	1 ms		
10,000	0.013 μs	10 μs	130 μs	100 ms		
100,000	0.017 μs	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 μs	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 μs	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 μs	0.10 sec	2.66 sec	115.7 days		
1,000,000,000	0.030 μs	1 sec	29.90 sec	31.7 years		

HOW CAN YOU QUICKLY FIND OUT COMPLEXITY?

O(?)

“ On the basis of the issues discussed here, I propose that members of SIGACT, and editors of computer science and mathematics journals, adopt the O, Ω and Θ notations as defined above, unless a better alternative can be found reasonably soon. ”

D. E. Knuth, "Big Omicron and Big Omega and BIg Theta", SIGACT News, 1976.

BENCHMARKS OR ASYMPTOTIC ANALYSIS?

USE BOTH APPROACHES!

SUMMARY

1. We want to predict running time of an algorithm.
2. Summarize all possible inputs with a single “size” parameter n.
3. Many problems with “empirical” approach (measure lots of test cases with various n and then extrapolate).
4. Prefer “analytical” approach.
5. To select best algorithm, compare their $T(n)$ functions.
6. To simplify this comparision “round” the function using asymptotic (“big-O”) notation
7. Amazing fact: Even though asymptotic complexity analysis makes many simplifying assumptions, it is remarkably useful in practice: if A is $O(n^3)$ and B is $O(n^2)$ then B really will be faster than A, no matter how they’re implemented.

LINKS

BOOKS:

- "Introduction To Algorithms, Third Edition", 2009, by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein
- "The Algorithm Design Manual, Second Edition", 2008, by Steven S. Skiena

OTHER:

- "Algorithms: Design and Analysis" by Tim Roughgarden **https://www.coursera.org/course/algo**
- Big-O Algorithm Complexity Cheat Sheet **http://bigocheatsheet.com/**

THE END

THANK YOU FOR ATTENTION!

- Vasyl Nakvasiuk
- Email: vaxxxa@gmail.com
- Twitter: @vaxXxa
- Github: vaxXxa

THIS PRESENTATION:

https://github.com/vaxXxa/talks