

Complexitat d'Algoritmes i Eficiència

(nocions bàsiques)

...

PRO1. Curs 2026-2027

Measuring Algorithm Complexity / Cost

- **Time Complexity**

- Measures how the running time grows as input size increases

- **Space Complexity**

- Measures how much memory the algorithm uses

- **Other (Less Common) Measures**

- **Network Complexity:** Amount of data transferred
- **I/O Complexity:** Number of disk or input/output operations
- **Communication Complexity:** How much information processes exchange

Measuring Algorithm Complexity / Cost

- **Time Complexity**

- Measures how the running time grows as input size increases
- Expressed using Big O notation (e.g., $O(1)$, $O(n)$, $O(n^2)$).
- Focuses on number of steps or operations, not actual seconds
- Example: Counting how many comparisons a sorting algorithm makes

Time Complexity: Why is it Important?

- Imagine you have to sort a list of numbers
 - If the list has 10 numbers, any algorithm is going to be quick
 - But if the list has 10 million numbers, some algorithms can take significantly longer time to run than others
- Time complexity analysis helps you predict how your program will scale and **choose the most efficient algorithm**
- Not only time is relevant, e.g., there might be a tradeoff between time and space complexity

Time Complexity: Big O notation

- **Big O notation** is a mathematical way to describe how the running time of an algorithm grows as the input size increases
- It doesn't measure exact time, but the **rate of growth** — how performance changes when the input gets larger
- Big O keeps the fastest-growing term (the one that dominates) and ignores constant factors and lower-order terms
- Different computers run at different speeds, but big O ignores hardware details. It helps compare algorithms in a machine-independent way.

Time Complexity: Big O notation examples

Number of steps in terms of n	Big-O	Common name
5	$O(1)$	Constant
$3 \cdot \log n + 10$	$O(\log n)$	Logarithmic
$4 \cdot \text{sqrt}(n) + 20$	$O(\text{sqrt}(n))$	Square-root
$3n + 7$	$O(n)$	Linear
$n \cdot \log n + 5n$	$O(n \cdot \log n)$	Linearithmic
$4n^2 + 3n + 10$	$O(n^2)$	Quadratic
$3^n + n^{(10)}$	$O(3^n)$	Exponential
etc.		

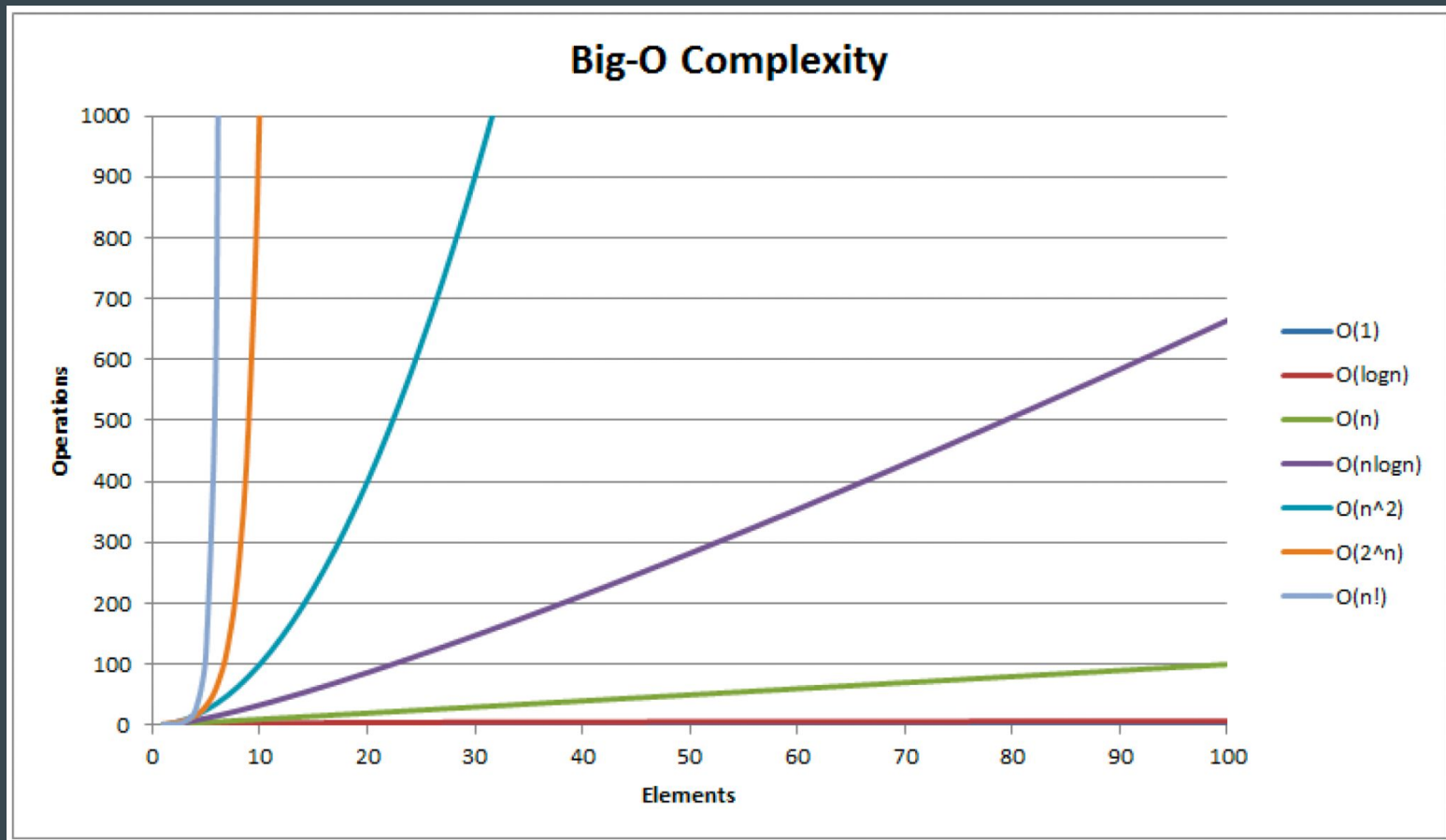
Time Complexity: Big O examples

Notation	Name	Growth Description	Example
$O(1)$	Constant	Same time, no matter input size	Accessing an array element
$O(\log n)$	Logarithmic	Grows very slowly	Binary search
$O(n)$	Linear	Grows in direct proportion to n	Scanning a list
$O(n \cdot \log n)$	Linearithmic	Grows slightly faster than linear	Merge sort, quicksort
$O(n^2)$	Quadratic	Grows quickly with input size	Bubble sort, nested loops
$O(2^n)$	Exponential	Explodes with small increase of n	Recursive subset generation
$O(n!)$	Factorial	Extremely fast growth	Traveling salesman brute-force

Time Complexity: Big O numeric view

n	$O(1)$	$O(\log n)$	$O(n)$	$O(n \cdot \log n)$	$O(n^2)$
1	1	0	1	0	1
10	1	3.32	10	33.2	100
100	1	6.64	100	664	10,000
1,000	1	9.97	1,000	9,970	1,000,000
10,000	1	13.29	10,000	132,900	100,000,000
100,000	1	16.61	100,000	1,661,000	10,000,000,000
1,000,000	1	19.93	1,000,000	19,930,000	1,000,000,000,000

Time Complexity: Big O functions



Time Complexity: Big O notation

n	$O(1)$	$O(\log n)$	$O(n)$	$O(n \cdot \log n)$	$O(n^2)$
1	1	0	1	0	1
10	1	3.32	10	33.2	100
100	1	6.64	100	664	10,000
1,000	1	9.97	1,000	9,970	1,000,000
10,000	1	13.29	10,000	132,900	100,000,000
100,000	1	16.61	100,000	1,661,000	10,000,000,000
1,000,000	1	19.93	1,000,000	19,930,000	1,000,000,000,000

Observations

- For small n (1–10), differences aren't dramatic
- For medium n (100–1,000), quadratic growth explodes compared to linear or logarithmic
- For large n (100,000–1,000,000), $O(n^2)$ is practically unusable, while $O(n \log n)$ is still manageable
- $O(\log n)$ is extremely efficient: even a million inputs take fewer than 20 steps

Time Complexity

Key Takeaways

- Small differences in time complexity matter a lot when input size n is large.
- Algorithms that seem fast for small n can become unusable if they're $O(n^2)$ or worse.
- Aim for $O(n)$ or $O(n \cdot \log n)$ for large inputs whenever possible
- Complexity analysis gives you objective elements to compare algorithms and select the best one given the existing constraints

Time Complexity

Discuss concrete examples from PRO1

- $O(\sqrt{n})$ algorithm for Divisors in Order ([P39308](#))
- Analyze the complexity of dichotomic search
- Analyze the complexity of the sorting algorithms seen in class