AI & Robotics

Complexity

AI & ROBOTICS LAB

Goals

The **junior-colleague**

- can explain in own words what Big O notation means
- can explain why only the term with the highest order of magnitude matters
- can derive the term with highest order of magnitude from a given polynomial
- can describe in own words the link between complexity and efficiency
- can explain the two most important parameters to measure efficiency
- can describe in own words the three different atomic step/instruction types of an algorithm
- can evaluate the time complexity of a given algorithm
- can explain the difference between worst-, average- and best-case time complexity
- can understand the implications of complexity growth w.r.t. input for different mathematical functions (constant, linear, logarithmic, quadratic, cubic, exponential and factorial)
- can evaluate the space complexity of data structures and algorithms
- can describe the complexity (space and time) for graph representations

Big O notation

- Big O notation is written in the mathematical notation of O(*f*):
  - O stands for “order of magnitude”
  - *f* represents what we’re comparing the complexity of a task against.
- A task can be handled using one of many algorithms, each of varying complexity and scalability over time.
- Only the term with the largest order of magnitude matters:

  => $O(n + 1) = O(n)$

  => $O(n^2 + n + 1) = O(n^2)$

  => $O(2*n^3) = O(n^3)$

Time Complexity

Efficiency of algorithms

- We need a way to describe the efficiency of an algorithm in a machine independent matter
- How?
  - Number of steps
  - Input

Efficiency of algorithms

- Step:
  - Atomic assignment instruction
  - Atomic Read/Write instruction
  - Atomic test instruction

=> Atomic instruction: unable to be split into sub-instructions

Efficiency of algorithms

- Example: Factorial

```
n = 10
factorial = 1

for i in range(1, n + 1):
    factorial = factorial * i
```

1. atomic assignment instruction
2. atomic assignment instruction
3. atomic test instruction
4. atomic assignment instruction

Efficiency of algorithms

- Input:
  - Size of the input
  - Type of the input

=> For our purposes we will use the number of input data elements as the size of our input

Time complexity

- Written in Big O notation
- Describes the execution time of a task in relation to the number of steps required to complete it.
- Execution time is described according to the term with the largest order of magnitude

  => $O(n + 1) = O(n)$

  => $O(n^2 + n + 1) = O(n^2)$

  => $O(2*n^3) = O(n^3)$

Time complexity

- Worst-case
- Average-case
- Best-case

Time complexity

- Example linear time complexity: Factorial

```
n = 10
factorial = 1

for i in range(1, n + 1):
    factorial = factorial * i
```

1. assignment instruction
2. assignment instruction
3. test instruction
4. assignment instruction

Size of input: n = 10

=> instructions 3 and 4 executed 10 times

=> O(n + n) = **O(n)**

Time complexity

- Example quadratic time complexity

```python
elements = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
sum = 0

for el1 in elements:
    for el2 in elements:
        sum += el1 * el2
```

Size of input: n = 10

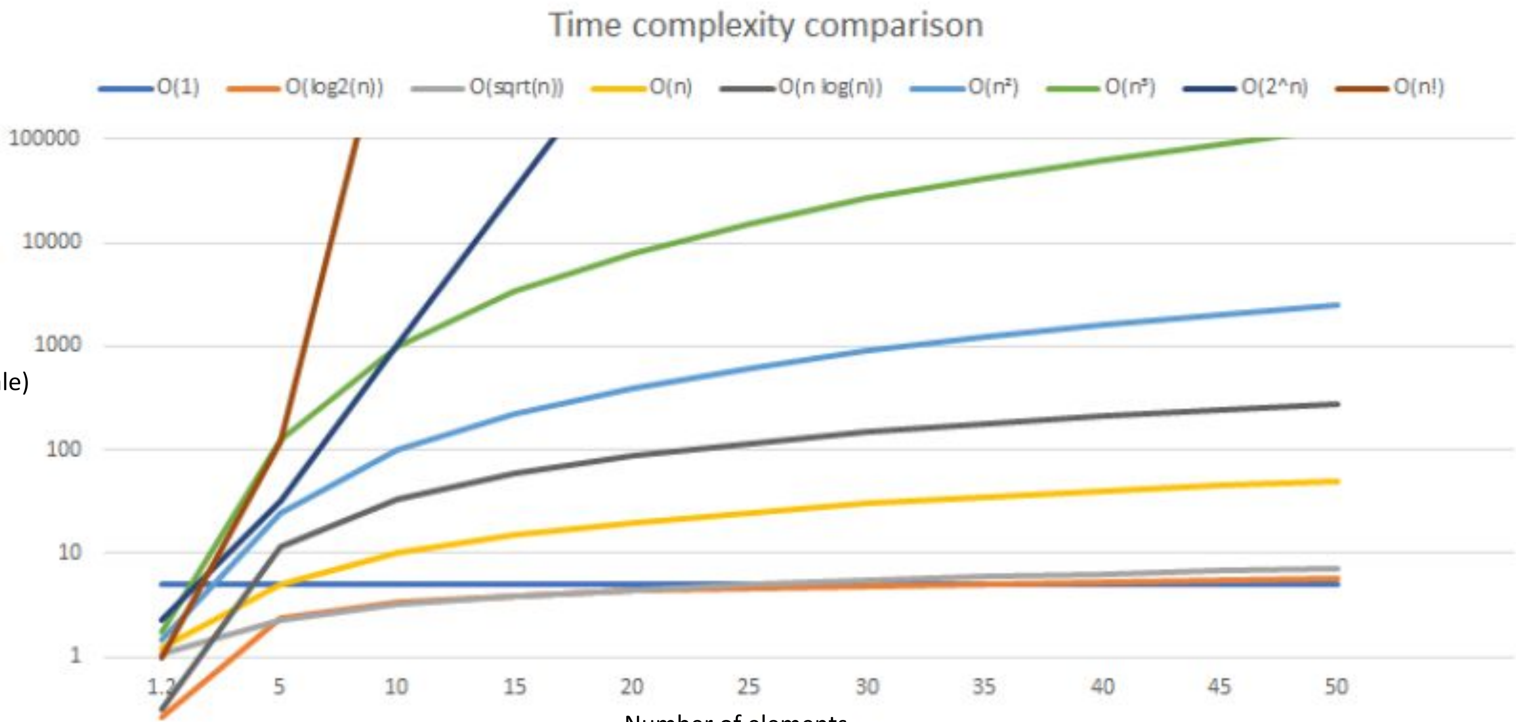
Iterate n times

Iterate n times

$O(n + n*(n + n)) = \mathbf{O(n^2)}$

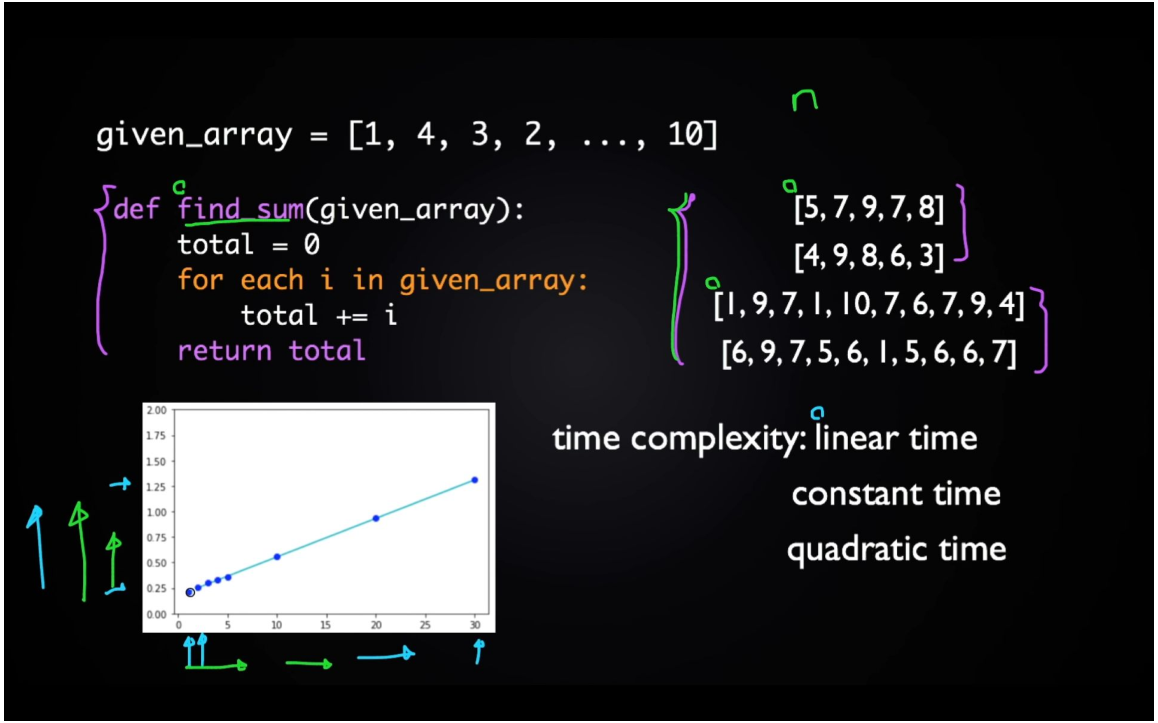
Time complexity

Number
of instructions
(warning: log scale)



Number of elements

Youtube video with excellent examples



[Watch] https://www.youtube.com/watch?v=D6xkbGLQesk

Space Complexity

Space complexity

- Written in Big O notation
- Describes the amount of memory space required to solve a task in relation to the size of the input
- Memory space is described according to the term with the largest order of magnitude

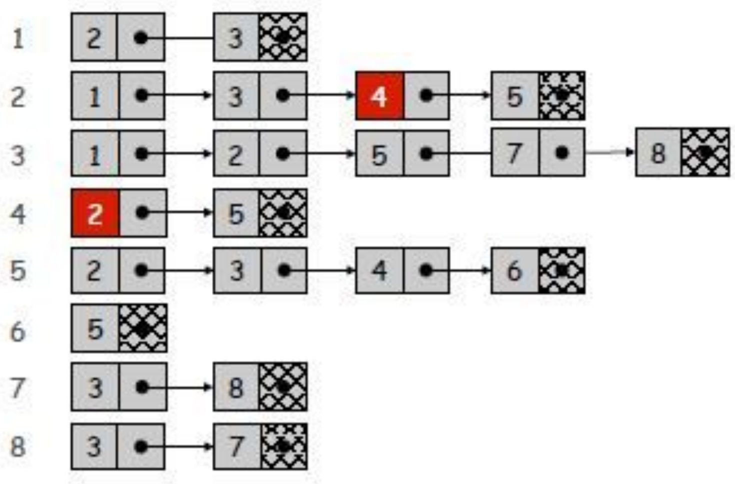
  => $O(n + 1) = O(n)$

  => $O(n^2 + n + 1) = O(n^2)$

  => $O(2*n^3) = O(n^3)$

Revisiting Graph representations

Nodes	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0



Revisiting Graph representations

	Adjacency Matrix	Adjacency List
Space complexity	$O(\|V\|^2)$	$O(\|V\| + \|E\|)$
Time complexity: Edge lookup	$O(1)$	$O(\|Vn\|)$
Time complexity: Edge iteration	$O(\|V\|)$	$O(\|Vn\|)$

$|V|$: number of vertices
$|E|$: number of edges
$|Vn|$: number of neighbouring vertices

Which one to use?

- Depends on the problem:
  - If time is your constraint => Adjacency Matrix
  - If memory is your constraint => Adjacency List