
FACE

VIT

FACE

Time & Space Complexity

FACE

Time & Space Complexity

What is Time Complexity?

Time Complexity: The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input. Note that the time to run is a function of the length of the input and not the actual execution time of the machine on which the algorithm is running on.
In order to calculate time complexity on an algorithm, it is assumed that a constant time c is taken to execute one operation, and then the total operations for an input length on N are calculated. Consider an example to understand the process of calculation: Suppose a problem is to find whether a pair (X, Y) exists in an array, A of N elements whose sum is Z. The simplest idea is to consider every pair and check if it satisfies the given condition or not.

Time & Space Complexity

Example: In case of addition of two n-bit integers, N steps are taken. Consequently, the total computational time is t(N) = c*n, where c is the time consumed for addition of two bits. Here, we observe that t(N) grows linearly as input size increases.

FACE

Time & Space Complexity

Types of notations for measuring Time & Space Complexity

1. O-notation: It is used to denote asymptotic upper bound. For a given function g(n), we denote it by O(g(n)). Pronounced as “big-oh of g of n”. It is also known as worst case time complexity as it denotes the upper bound in which the algorithm terminates.
2. Ω-notation: It is used to denote asymptotic lower bound. For a given function g(n), we denote it by Ω(g(n)). Pronounced as “big-omega of g of n”. It is also known as best case time complexity as it denotes the lower bound in which the algorithm terminates.
3. **Θ**-notation: It is used to denote the average time of a program.

Time & Space Complexity

Big – O Definition

An algorithm's Big-O notation is determined by how it responds to different sizes of a given dataset. For instance how it performs when we pass to it 1 element vs 10,000 elements.

O stands for Order Of, so O(N) is read "Order of N" — it is an approximation of the duration of the algorithm given N input elements. It answers the question: "How does the number of steps change as the input data elements increase?"

O(N) describes how many steps an algorithm takes based on the number of elements that it is acted upon.

Time & Space Complexity

Big – O

- Starting with a gentle example: Given an input array[N], and a value X, our algorithm will search for the value X by traversing the array from the start until the value is found.
- Given this 5-element array: [2,1,6,3,8] if we were searching for X=8 the algorithm would need 5 steps to find it, but if we were searching for X=2 it would only take 1 step. So best case scenario is when we look for a value that is in the first cell and worst case scenario is when the value is at the last cell, or not there at all.
- The Big-O notation takes a pessimistic approach to performance and refers to the worst case scenario. This is really important when we describe the complexities below, and also when you try to compute the complexity of your own algorithms: Always think of the worst case scenarios.

Time & Space Complexity

O(1) — Constant

O(1) means that the algorithm takes the same number of steps to execute regardless of how much data is passed in.

Example: Determine if the n-th element of an array is an odd number.

Time & Space Complexity

O(1) — Constant

```
import numpy as np
array = np.array([0, 2, 3, 6, 0, 8, 5, 5, 2])
```

```
def isNthElementOdd(array, n):
    return bool(array[n] % 2)
```

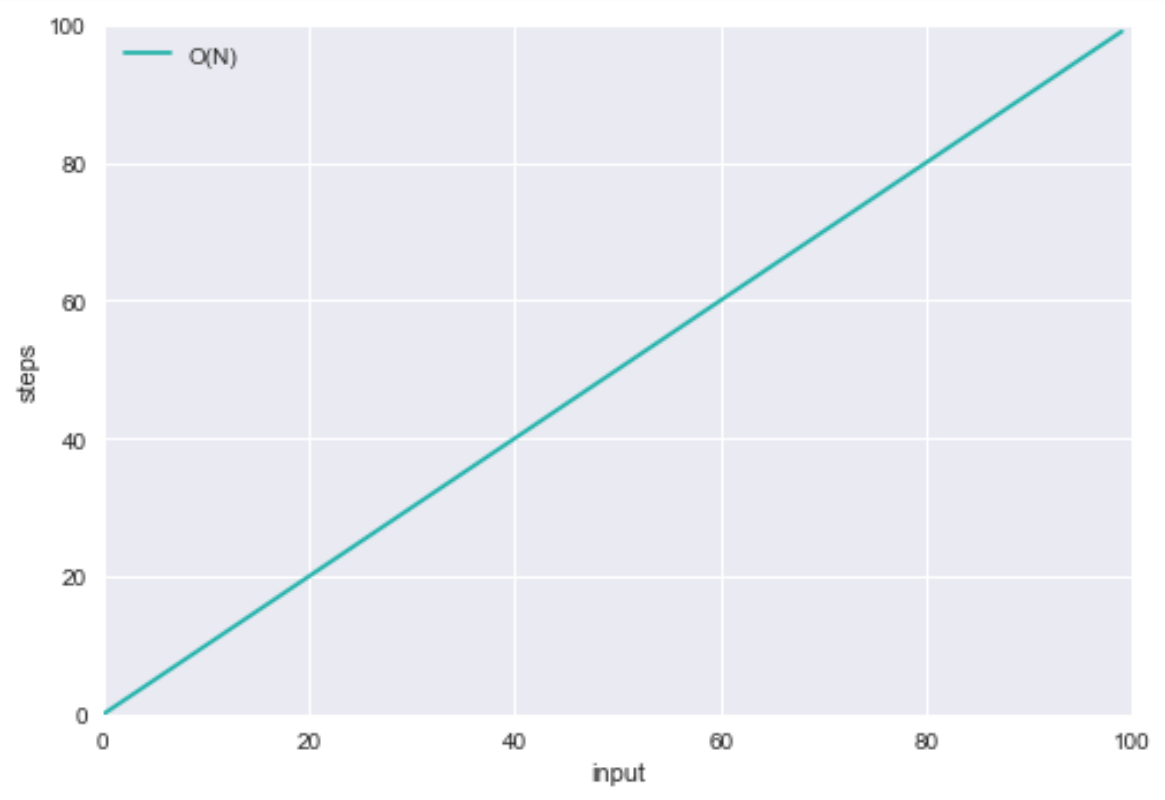
```
isOdd = isNthElementOdd(array, 2)
display(isOdd)
```

```
True
```

Whether we access the 1st or 2nd or millionth item it doesn't matter... We can access it directly by using the index operator array[i]

Time & Space Complexity

O(N) — Linear

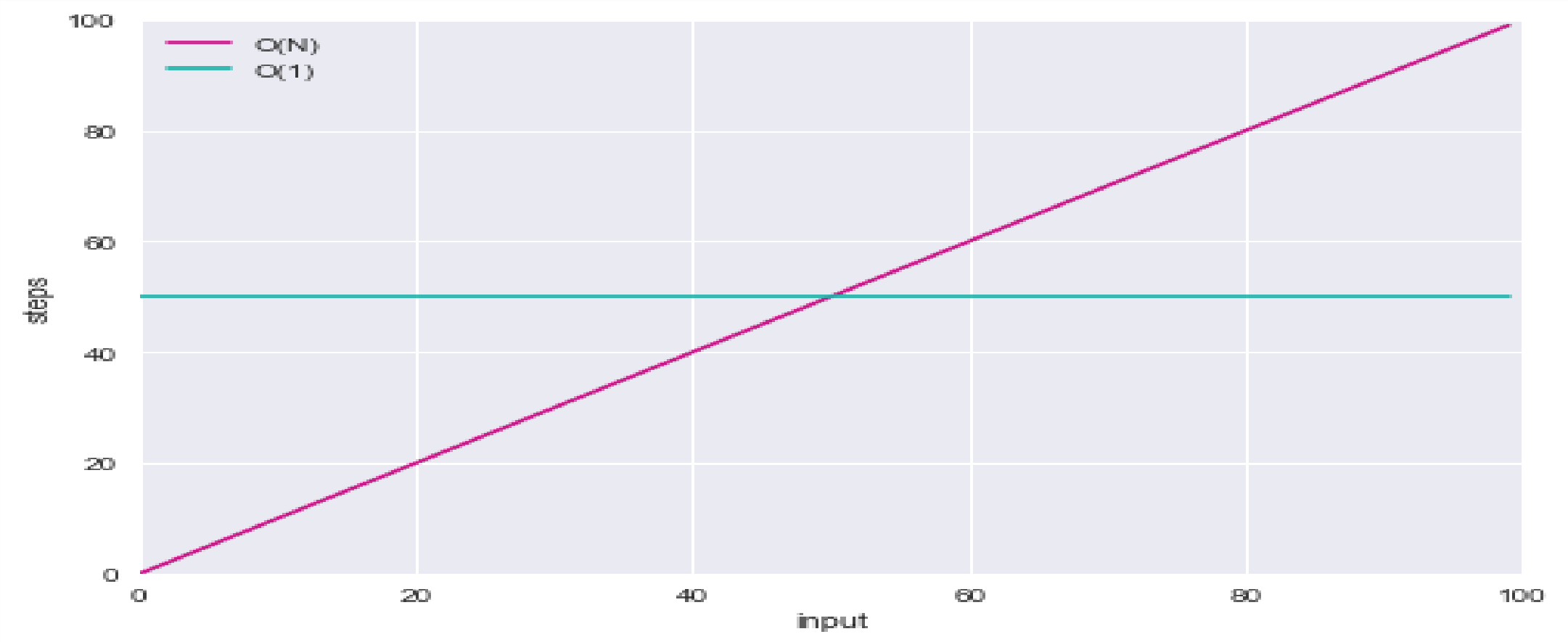


O(N) is a perfect diagonal line, as for every additional piece of data, the algorithm takes one additional step. This is why it is also referred to as linear time.

FACE

Time & Space Complexity

O(N) vs O(1)



FACE

Time & Space Complexity

O(N) vs O(1)

Let's plot the O(1) and O(N) algorithms in the same graph and let's assume that the O(1) algorithm constantly takes 50 steps.

What can we observe?

→ When the input array has less than 50 elements, the O(N) is more efficient.
→ At exactly 50 elements the two algorithms take the same number of steps.
→ As the data increases the O(N) takes more steps.

Since the Big-O notation looks at how the algorithm performs as the data grows to infinity, this is why O(N) is considered to be less efficient than O(1).

Time & Space Complexity

What is Space Complexity?

Space Complexity: When an algorithm is run on a computer, it necessitates a certain amount of memory space. The amount of memory used by a program to execute it is represented by its space complexity. Because a program requires memory to store input data and temporal values while running, the space complexity is auxiliary and input space.

Example: int consumes 4 bytes of memory.

Time & Space Complexity

What is Space Complexity?

Space Complexity: When an algorithm is run on a computer, it necessitates a certain amount of memory space. The amount of memory used by a program to execute it is represented by its space complexity. Because a program requires memory to store input data and temporal values while running, the space complexity is auxiliary and input space.

THANK YOU

FACE