# Time Complexity and Big O Notation (with notes)

🌐 codewithharry.com/videos/data-structures-and-algorithms-in-hindi-2

An analogy to a real-life issue:

- This morning I wanted to eat some pizza; So, I asked my brother to get me some from Dominos, which is 3 km away.
- He got me the pizza, and I was happy only to realize it was too little for 29 friends who came to my house for a surprise visit!
- My brother can get 2 pizzas for me on his bike, but pizza for 29 friends is too huge of an input for him, which he cannot handle.
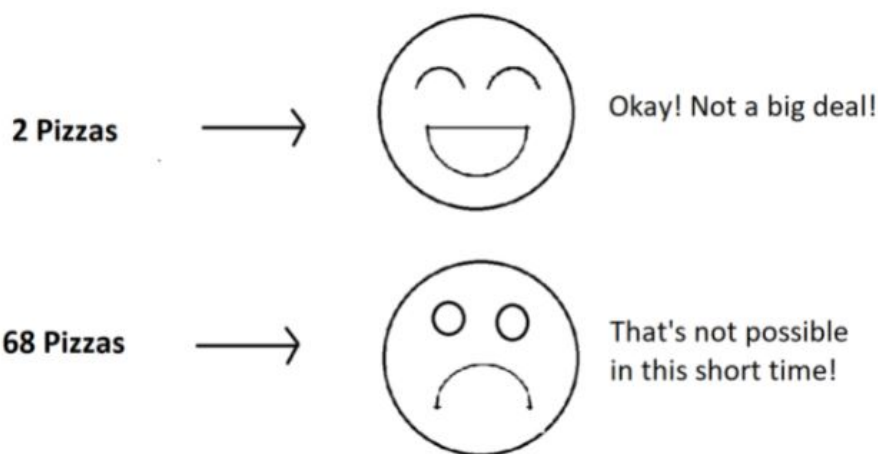
What is Time Complexity?

Time Complexity is the study of the efficiency of algorithms. It tells us how much time is taken by an algorithm to process a given input. Let's understand this concept with the help of an example:

Consider two developers Shubham and Rohan, who created an algorithm to sort 'n' numbers independently. When I made the program run for some input size n, the following results were recorded:

No. of elements (n)	Time Taken By Shubham's Algo	Time Taken By Rohan's Algo
10 elements	90 ms	122 ms
70 elements	110 ms	124 ms
110 elements	180 ms	131 ms

1000 elements	2s	800 ms

We can see that at first, Shubham's algorithm worked well with smaller inputs; however, as we increase the number of elements, Rohan's algorithm performs much better.

Quick Quiz: Who's algorithm is better?

Time Complexity: Sending GTA 5 to a friend:

- Imagine you have a friend who lives 5 km away from you. You want to send him a game. Since the final exams are over and you want him to get this 60 GB file worth of game from you. How will you send it to him in the shortest time possible?
- Note that both of you are using JIO 4G with a 1 Gb/day data limit.
- The best way would be to send him the game by delivering it to his house. Copy the game to a hard disk and make it reach him physically.
- Would you do the same for sending some small-sized game like MineSweeper which is in KBS of size? Of Course no, because you can now easily send it via the Internet.
- As the file size grows, the time taken to send the game online increases linearly – $O(n)$ while the time taken by sending it physically remains constant. $O(n0)$ or $O(1)$.

Calculating Order in terms of Input Size:

In order to calculate the order(time complexity), the most impactful term containing n is taken into account (Here n refers to Size of input). And the rest of the smaller terms are ignored.

Let us assume the following formula for the algorithms in terms of input size n:

Algo 1:
$$k_1 n^2 + k_2 n + 36 = O(n^2)$$

$\underbrace{k_1 n^2}_{\text{Highest Order Term}}$ $\underbrace{k_2 n + 36}_{\text{Can ignore other lower order terms}}$

Algo 2:
$$k_1 k_2^2 + k_3 k_2 + 8 = O(n^0) = O(1)$$

Here, we ignored the smaller terms in algo 1 and carried the most impactful term, which was the square of the input size. Hence the time complexity became n^2. The second algorithm followed just a constant time complexity.

Note that these are the formulas for the time taken by their program.

What is a Big O?

Putting it simply, big O stands for 'order of' in our industry, but this is pretty different from the mathematical definition of the big O. Big O in mathematics stands for all those complexities our program runs in. But in industry, we are asked the minimum of them. So this was a subtle difference.

Visualizing Big O:

If we were to plot O(1) and O(n) on a graph, they would look something like this:
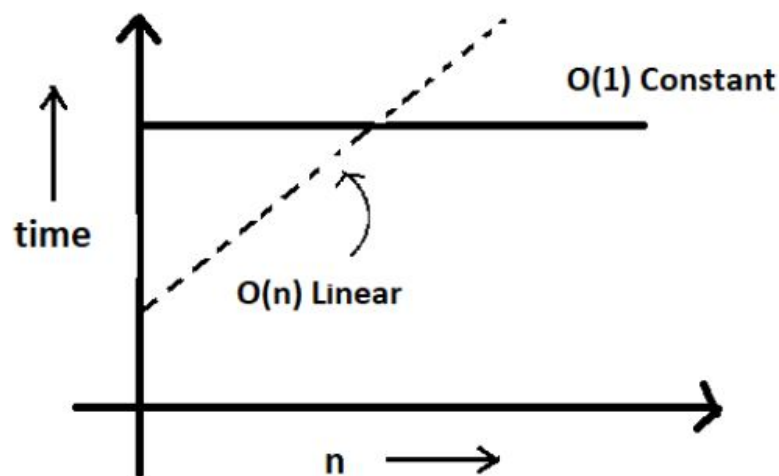
So, this was the basics of time complexities. You don't have to worry about things passing you by. We'll learn more deeply in the coming videos. For now, just stay with me.

Thank you for being with me throughout the tutorial. I hope you enjoyed it. If you appreciate my work, please let your friends know about this course too. Make sure to download the notes linked below. See you all in the next tutorial, where we'll learn more about time complexities, Big O, and other asymptotic notations. Till then, keep learning.

Download Notes here