Best Case, Worst Case and Average Case Analysis of an Algorithm (With Notes)

codewithharry.com/videos/data-structures-and-algorithms-in-hindi-4

Life can sometimes be lucky for us:

> Exams getting canceled when you are not prepared, a surprise test when you are prepared, etc. → **Best case**

Occasionally, we may be unlucky:

> Questions you never prepared being asked in exams, or heavy rain during your sports period, etc. → **Worst case**

However, life remains balanced overall with a mixture of these lucky and unlucky times. → **Expected case**

Those were the analogies between the study of cases and our everyday lives. Our fortunes fluctuate from time to time, sometimes for the better and sometimes for the worse. Similarly, a program finds it best when it is effortless for it to function. And worse otherwise.

By considering a search algorithm used to perform a sorted array search, we will analyze this feature.

Analysis of a search algorithm:

Consider an array that is sorted in increasing order.

1	7	18	28	50	180

We have to search a given number in this array and report whether it's present in the array or not. In this case, we have two algorithms, and we will be interested in analyzing their performance separately.

1. **Algorithm 1** – Start from the first element until an element greater than or equal to the number to be searched is found.
2. **Algorithm 2** – Check whether the first or the last element is equal to the number. If not, find the number between these two elements (center of the array); if the center element is greater than the number to be searched, repeat the process for the first half else, repeat for the second half until the number is found. And this way, keep dividing your search space, making it faster to search.

Analyzing Algorithm 1: (Linear Search)

We might get lucky enough to find our element to be the first element of the array. Therefore, we only made one comparison which is obviously constant for any size of the array.

Best case complexity = O(1)

If we are not that fortunate, the element we are searching for might be the last one. Therefore, our program made 'n' comparisons.

Worst-case complexity = O(n)

For calculating the average case time, we sum the list of all the possible case's runtime and divide it with the total number of cases. Here, we found it to be just O(n). (Sometimes, calculation of average-case time gets very complicated.)

Analyzing Algorithm 2: (Binary Search)

If we get really lucky, the first element will be the only element that gets compared. Hence, a constant time.

Best case complexity = O(1)

- If we get unlucky, we will have to keep dividing the array into halves until we get a single element. (that is, the array gets finished)
- Hence the time taken : n + n/2 +n/4 + ... + 1 = logn with base 2

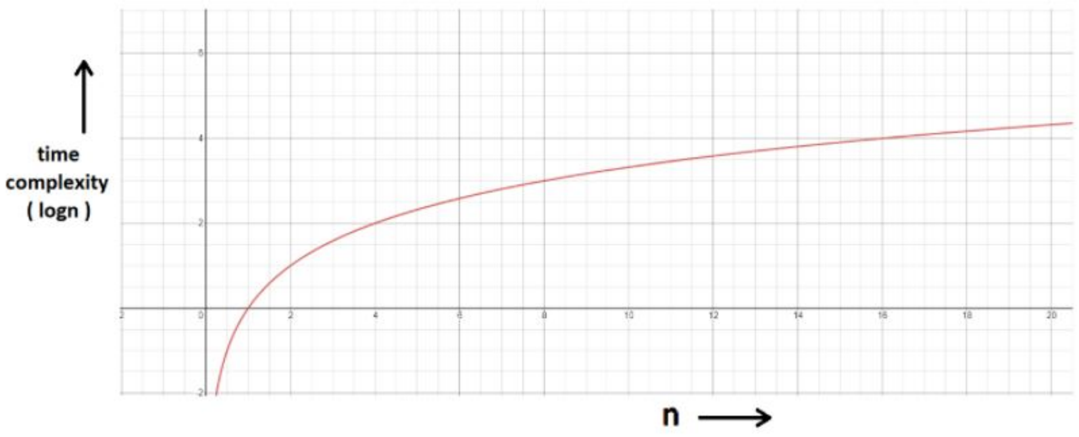
Worst-case complexity = O(log n)

What is log(n)?

Logn refers to how many times I need to divide n units until they can no longer be divided (into halves).

- log8 = 3 ⇒ 8/2 + 4/2 + 2/2 → Can't break anymore.
- log4 = 2 ⇒ 4/2 + 2/2 → Can't break anymore.

You can refer to the graph below, and you will find how slowly the time complexity (Y-axis) increases when we increase the input n (X-axis).



Space Complexity:

- Time is not the only thing we worry about while analyzing algorithms. Space is equally important.
- Creating an array of size n (size of the input) → O (n) Space
- If a function calls itself recursively n times, its space complexity is O (n).

Quiz Quiz: Calculate the space complexity of a function that calculates the factorial of a given number n.

Hint: Use recursion.

You might have wondered at some point why we can't calculate complexity in seconds when dealing with time complexities. Here's why:

- Not everyone's computer is equally powerful. So we avoid handling absolute time taken. We just measure the growth of time with an increase in the input size.
- Asymptotic analysis is the measure of how time (runtime) grows with input.

Thank you for being with me throughout. I hope you enjoyed the tutorial. If you appreciate my work, please let your friends know about this course too. Make sure to download the notes linked below. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. See you all in the next tutorial, where we'll learn how to calculate these time complexities with a few solved examples. Till then, keep learning.

Download Notes