Why performance analysis?

There are many important things that should be taken care of, like user friendliness, modularity, security, maintainability, etc. Why to worry about performance?

The answer to this is simple, we can have all the above things only if we have performance. So performance is like currency through which we can buy all the above things. Another reason for studying performance is – speed is fun!

To summarize, performance == scale. Imagine a text editor that can load 1000 pages, but can spell check 1 page per minute OR an image editor that takes 1 hour to rotate your image 90 degrees left OR ... you get it. If a software feature can not cope with the scale of tasks users need to perform – it is as good as dead.

Given two algorithms for a task, how do we find out which one is better?

One naive way of doing this is – implement both the algorithms and run the two programs on your computer for different inputs and see which one takes less time. There are many problems with this approach for analysis of algorithms.

```
1) It might be possible that for some inputs, first algorithm performs
better than the second. And for some inputs second performs better.

2) It might also be possible that for some inputs, first algorithm perform
better on one machine and the second works better on other machine for
some other inputs.
```

Asymptotic Analysis is the big idea that handles above issues in analyzing algorithms. In Asymptotic Analysis, we evaluate the performance of an algorithm in terms of input size (we don't measure the actual running time). We calculate, how the time (or space) taken by an algorithm increases with the input size.

For example, let us consider the search problem (searching a given item) in a sorted array. One way to search is Linear Search (order of growth is linear) and the other way is Binary Search (order of growth is logarithmic). To understand how Asymptotic Analysis solves the above mentioned problems in analyzing algorithms, let us say we run the Linear Search on a fast computer A and Binary Search on a slow computer B and we pick the constant values for the two computers so that it tells us exactly how long it takes for the given machine to perform the search in seconds.

Let's say the constant for A is 0.2 and the constant for B is 1000 which means that A is 5000 times more powerful than B. For small values of input array size n, the fast computer may take less time. But, after a certain value of input array size, the Binary Search will definitely start taking less time compared to the Linear Search even though the Binary Search is being run on a slow machine.

The reason is the order of growth of Binary Search with respect to input size is logarithmic while the order of growth of Linear Search is linear. So the machine dependent constants can always be ignored after a

certain value of input size.

Here are some running times for this example:

```
Linear Search running time in seconds on A: 0.2 * n

Binary Search running time in seconds on B: 1000*log(n)
```

```
_____________________________________________
|n | Running time on A | Running time on B |
_____________________________________________
|10 | 2 sec | ~ 1 h |
_____________________________________________
|100 | 20 sec | ~ 1.8 h |
_____________________________________________
|10^6 | ~ 55.5 h | ~ 5.5 h |
_____________________________________________
|10^9 | ~ 6.3 years | ~ 8.3 h |
_____________________________________________
```

Does Asymptotic Analysis always work?

Asymptotic Analysis is not perfect, but that's the best way available for analyzing algorithms.

For example, say there are two sorting algorithms that take 1000nLogn and 2nLogn time respectively on a machine. Both of these algorithms are asymptotically same (order of growth is nLogn).

So, With Asymptotic Analysis, we can't judge which one is better as we ignore constants in Asymptotic Analysis.

Also, in Asymptotic analysis, we always talk about input sizes larger than a constant value. It might be possible that those large inputs are never given to your software and an algorithm which is asymptotically slower, always performs better for your particular situation. So, you may end up choosing an algorithm that is Asymptotically slower but faster for your software.

Complexity Analysis is the foundational knowledge that is needed to better understand DS. It is the bedrock of coding interviews. It also helps in finding the best solution among many solutions.

```
Complexity Analysis is all about Time and Space Complexity.

Time Complexity is the measure of how fast an algorithm runs.

Space Complexity is the amount of memory algorithm uses up.
```

Memory is the foundational knowledge to understand complexity analysis and data structures. A variable is stored somewhere in the PC in memory. The memory is a bounded canvas and has a finite number of memory slots. The less memory a program takes, the better it performs.

The program will always store variables in continuous free memory slots and in back to back slots. Memory is made up of bits 0 and 1. Any piece of data can be transformed into a binary representation. Then it is stored in the PC in memory in blocks of 8 bits.

```
Int in C++/Java = 32 bits = 4 bytes
Long in Java = 64 bits = 8 bytes
```

Integers are fixed-width integers. eg, 64 bits. Endianness helps in ordering the bytes while representing them.

Array/ List of numbers will be stored only in a contiguous memory location, back to back, free memory slots. eg: [1,2,3]. Strings are converted into characters and then mapped to its ASCII value and then that ASCII is converted to binary digits and stored in memory slots.

Pointers are used to point to another memory address in base 2 format. We can quickly point to a very far away memory location by using a pointer. All of these memory slots can be accessed very quickly by PC. Eg; array access

Big-O Notation

We don't describe time complexity in terms of seconds. It is impossible and that doesn't make sense because it depends on the size of the input. The speed of the algorithm is dependant on the input size.

The same program may take different run time each time because of the computer's performance. But this is a very very negligible difference.

Time and Space Complexity is the measure of an algorithm speed/ run time when the size of input increases. Asymptotic analysis means that we are analyzing the behavior of a function as its value tends to infinity.

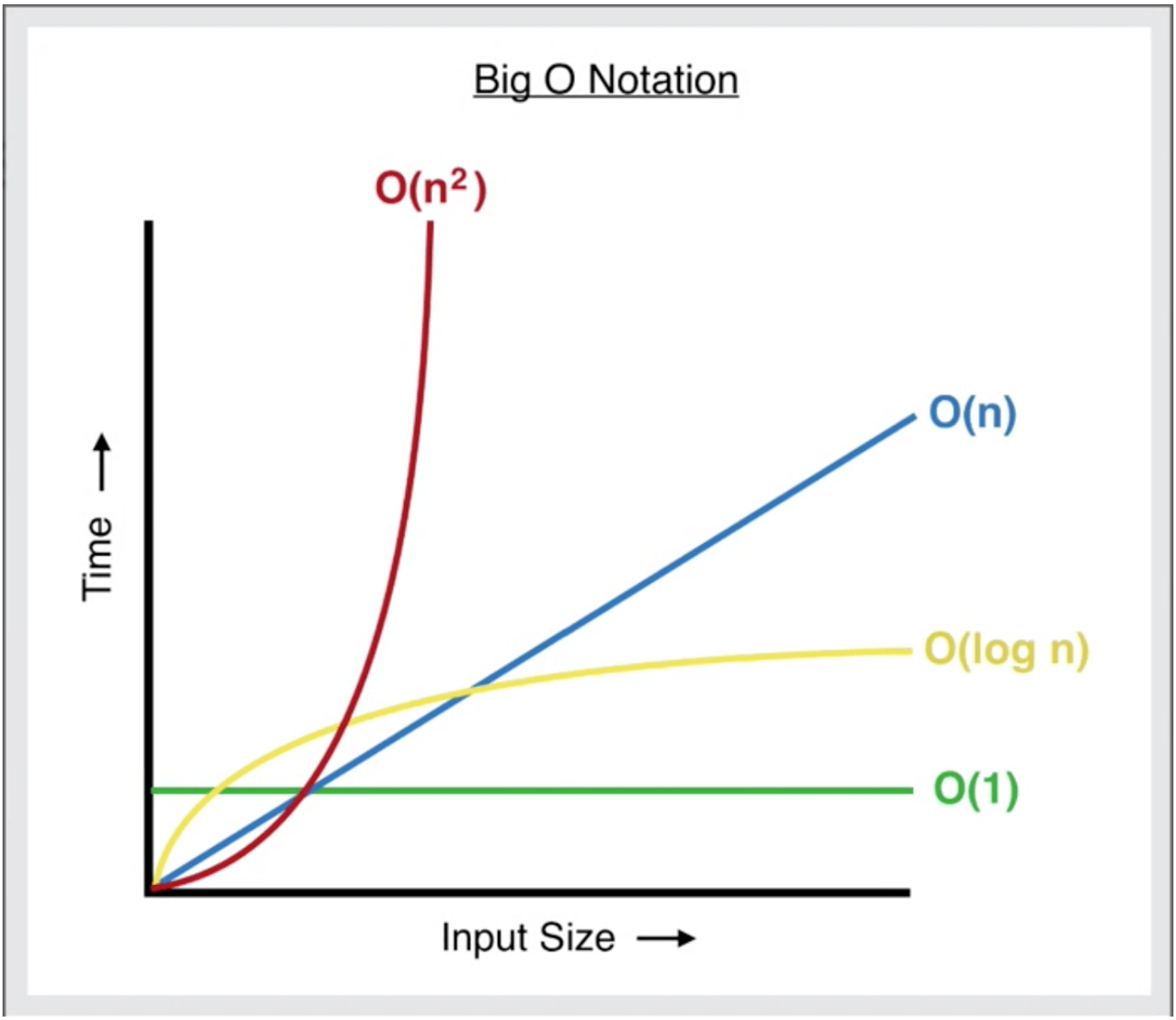
Big-O is the space/time complexity in the worst-case scenario. The unit of big-O is vague but can be defined as the time taken for one operation.

```
O(8) and O(1) is the same in asymptotic analysis

O(2N) is the same as O(N) as well.

O(N³) is removed as compared to O(N²)

But O(M+N) can not be removed as M and N are different
```


Big O Notation
O(n²)
O(n)
O(log n)
O(1)
Time
Input Size