# Big O and Complexity

## Big O

The mathematical definition:

“Big O notation is a mathematical notation that describes the **limiting behaviour** of a function when the **argument** tends towards a particular value or infinity”

Breaking it down:

Importantly before we carry on, we are speaking about functions in a mathematical sense like , not about algorithms, but as you will see later we will use analysis to describe a algorithm as a mathematical function.

The limiting behaviour is a way of describing the properties of a function as the argument becomes large. So, given the function what part of the equation becomes most significant as the argument becomes large.

For example, given:

The limiting factor of the function would be as when the argument is large becomes insignificant compared to .

Let ,

This is called **asymptotic analysis** and when writing about this formally you would say that (in this case) is asymptotically equivalent to or .

To define the limiting factor properly, given the function and ,

If

Don’t worry if you didn’t understand all of this, a lot of this is just maths for people who are interested and want to know more about how Big O works.

In terms of computer science, Big O is used to classify algorithms based upon their complexity. It is used to show how an algorithm scales with size of input.

Complexity is used to bridge the gap between the mathematical and computer science definition of a function. So, what is Complexity?

## Complexity

Complexity of an algorithm is the amount of resources needed to run it. These resources are generally time or space, but there are other types.

### Time

When complexity is said without giving what type it is generally people mean time complexity as it is the most common use. When talking about time complexity you won’t use seconds or minutes as how fast an algorithm can carry out a task is dependant on the computer not on the algorithm. For example, an algorithm run on a computer from 20 years ago would almost certainly run slower than the same algorithm run on a modern computer, so the actual time taken is not intrinsic to the algorithm itself.

Instead you will be counting the number of “elementary operations” which basically means operations such as comparisons, adding, subtracting, etc. There is not official name for this but common names for this is steps or operations.

#### Example time

Below is a simple algorithm (in Python) for finding a value in an array:



This will return the index of the value in the array and -1 if it cannot be found. Looking at the algorithm each loop it goes through it will do 1 comparison. It will do this 1 comparison for each value until the value is found.

To use the Big O notation, we assume we have a problem of size . This is a very general term and you will have to decide what that means on a case by case basis. For this example, we assume the array has items in it. There are 3 types of measure of an algorithm’s complexity: best, worst and average case:

##### Best

The best case is very simple to calculate as you assume that everything that can go write does. So, for this case you assume that the value you are looking for is at the first index. This means there will only be 1 comparison no matter how large the array is. So best case is O(1).

##### Worst

The worst case is very similar to best case, but you assume everything goes “wrong”. So, you assume the case for which the algorithm would take the most time. In this case it would be that the value is in the last index, so the algorithm will have to do comparisons every time, giving the worst-case complexity of O(n).

##### Average

To calculate the average complexity, we will assume that the value we are looking for is in a random place and (importantly) it only appears once in the array and the given array has values. The value on average at the index, I could prove this now but its intuitive to see that halfway would be the average position of the given value.

So, this means the algorithm has a complexity of where is the size of the array. In big O notation this is O(n) or linear time.

You may be thinking now that its odd that the average and worst case have the same complexity, but an important factor of Big O is that it’s a measure of how an algorithm scales not which algorithm is “faster”.

### Space

Space (meaning the amount of memory used) is less commonly used but still important. Analysis it very similar to time, for the example above can you guess what the space complexity will be?

At first you may want to say O(1) as its constant no new data it being created but this is wrong as we care about how the space scales as increases. As is the size of the given array its space complexity is O(n).

## References

<https://en.wikipedia.org/wiki/Big_O_notation>

<https://en.wikipedia.org/wiki/Computational_complexity>

<http://pages.cs.wisc.edu/~vernon/cs367/notes/3.COMPLEXITY.html>

<https://www.studytonight.com/data-structures/space-complexity-of-algorithms>

<https://cs.stackexchange.com/questions/80145/how-to-prove-that-average-complexity-is-n-2-for-linear-search-in-the-unsorted-ar>