ASYMPTOTIC NOTATIONS

ACADVIEW

So far you must have encountered and written yourself some algorithms, to perform some task. You may desire to compare your written algorithm with that of your friend’s or simply wish to evaluate its performance. To do so the key is Efficiency of the algorithm.

We can characterize the efficiency of an algorithm in terms of

How long does it takes to execution/completion (Time complexity)

The Space it requires (Space complexity)

For the purpose of analyzing the space & time complexity of some algorithm, generally we can never provide an exact figure, instead we describe it using some standard notations, called, **Asymptotic Notations.**

For instance, when we analyze some algorithm, we derive a formula to represent the amount of time its execution takes, temporary memory requirements etc.

Asymptotic Notations applies to such functions.

The Functions to which we apply asymptotic notations, can characterize to

* + Running time of algorithm (Time complexity)
  + Amount of space required by algorithm (Space Complexity)
  + Or some other aspect

However, we will here, characterize the asymptotic notations to the running time of the algorithms.

The **running time** of an algorithm (which is a function) determines how long the algorithm takes to completion, in terms of the size of its input.

The *behavior of the running time* (which is a function) with the *increase* in i*nput size*, is called, ***rate of growth/growth rate*** of running time; in other words, how fast does the function grows as the input size increases.

When we look at input size large enough, that makes only the rate of growth of the running time relevant, we are studying ***asymptotic efficiency*** of algorithms. And the notations we use to describe the asymptotic running time of algorithm are called **asymptotic notations**.

In short, Asymptotic notations allows us to analyze an algorithm’s running time by finding it’s behavior as the input size increases, hence providing an insight into performance & efficiency and answers questions such as

* Will the algorithm become drastically slow, when input size increases manifold?
* Does it maintain its quick run time as the input size increases?

1. **Big-** **(Big-O) Notation**

* When we say that the running time is , we mean that once the input size gets large enough, the running time is at most / always less than or equal to for some constants .

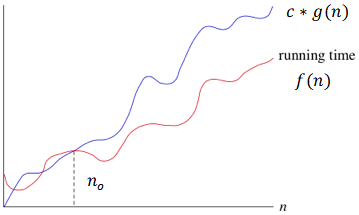
*"the running time grows at most this much, but it could grow more slowly.”*

* Say

: is some algorithm’s runtime,

: some arbitrary complexity, that we are trying to relate to our function

When we say the running time of some algorithm is , where  **it means, “  *is if and only if there exist some positive constants such that”***



* It provides upper limit on the running time i.e., the running time is always less than or equal to the limit boundary for all
* It denotes the ***asymptotically Upper bound*** on the growth rate of runtime of an algorithm,

is *asymptotic upper bound* for

Example:

Since,

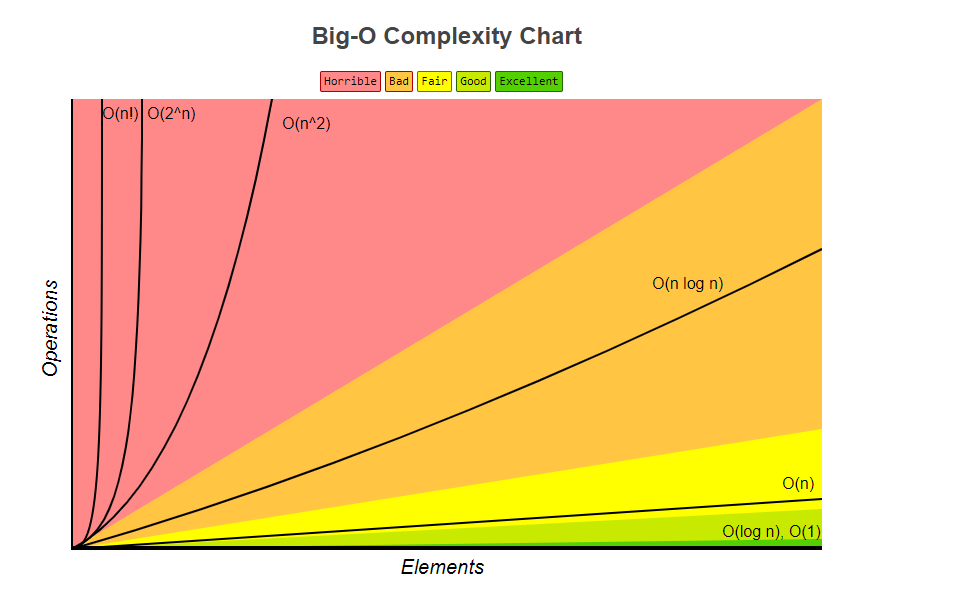
Hence, we can say that

* should be the ***smallest function of***  as one can come up with for which

We often say, We never say, , even though it is correct

* means constant time
* Using Big-O notation we can make statements that are technically correct but may be misleading. Say I have 1000 chocolates, I can truthfully say, that I definitely have no more than 2,00,000 chocolates. Although, I am technically right but greatly imprecise. Hence should be the **smallest** **possible**.

Similarly, the **worst-case** of **linear search** will make comparisons. Hence, I can say it like the worst-case of linear search is ( instead of saying ) which is technically correct, because the running time cannot grow faster than this, in fact, it would go slower.



1. **Big- (Big-Omega) Notation**

* When we say that the running time is , we mean that once the input size gets large enough, the running time is at least / always greater than for some constants .

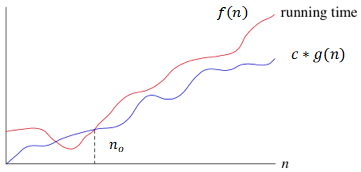
*"This is the slowest the running time can grow, but it can grow much faster.”*

* Say

: is some algorithm’s runtime,

: some arbitrary complexity, that we are trying to relate to our function

When we say the running time of some algorithm is , where  **it means, “  *is if and only if there exist some positive constants such that ”***



* It provides lower limit on the running time i.e.; the running time is always greater than the limit boundary for
* It denotes the ***asymptotically lower bound*** on the growth rate of runtime of an algorithm, is *asymptotic lower bound* for

Example:

Since,

Hence, we can say that

* should be the ***largest function of***  as one can come up with for which

We often say, We never say, , even though it is correct

* We can also make **correct, but imprecise, statements** using big-Ω notation. For example, if you have 1000 chocolates. You can truthfully say I have **at least** 1 chocolate. Though this is correct but highly imprecise and certainly doesn’t help a lot in guessing. Hence should be largest.

Similarly, if you are talking about **worst case scenarios** of **binary search**, which is you can say it as **,** which should make it clear, how misleading it can be at times.

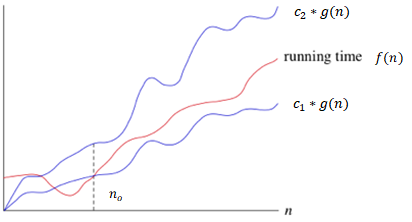
1. **Big-(Big-Theta) Notation**

* When we say that the running time is , we mean that once the input size gest large enough, the running time is at least and at most for some constants and
* Say

: is some algorithm’s runtime,

: some arbitrary complexity, that we are trying to relate to our function

When we say the running time of some algorithm is , where  **it means, “  *is if and only if there exist some positive constants such that ”***

****

* It also implies that
* It provides upper and lower limits on the running time within constant factors
* The running time is sandwiched between the two boundaries
* It denotes the ***asymptotically tight bound*** on the growth rate of runtime of an algorithm,

is *asymptotic tight bound* for

Example:

Since, also,

Hence, we can say that