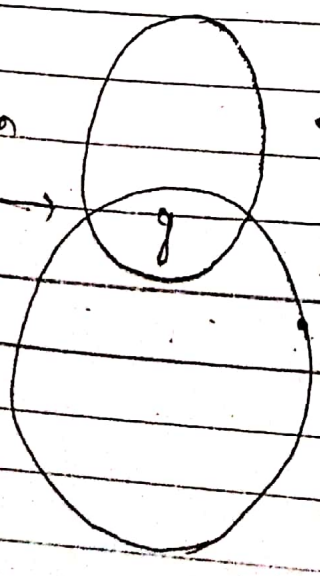
Asymptotic Notation

we can obtain different running time for the algorithm. once an algorithm is given for a problem and determined to be correct, the next step is to determine the amount of resources such as time and space that the algorithm will require. This is called algorithm analysis.

whenever we want to perform analysis of an algorithm, we need to calculate the complexity of that algorithm. Instead of taking exact amount of resources we represent the complexity in a general form which produces the basic nature of that algorithm.

Asymptotic Notation

The asymptotic running time of an algorithm is defined in term of function. Now consider two function 'f' and 'g'. This functions are formed as follows:



g): function that grow at same rate as g.

← Ω(g): function that grow at least as fast as g.

← O(g): function that grow number faster than g.

Best case, Worst case and Average case complexity

Consider an example, we have a linear array consisting of ten different elements. Now we need to find any number from the given list. when we get data as first element it is known as best case.

Average Case

Here no of steps taken by algorithm to search a data item and the no. of step is half of size of array is called average case.

Best case, Average case and worst case

Suppose, we have a list of names in which we have to search for a particular name. we have to search for a particular name, we have designed an algorithm that searches the name in the list of 'n' element by comparing the name to be searched within the each element in the list sequentially.

1. Best case time complexity

The best case time complexity is the minimum amount of time that an algorithm requires for an i/p input size 'n'. Thus, It is a function defined by the minimum of steps taken on any instance of size 'n'.

Example:-
The best case in this scenario would be if the first element in the list matches the name to be searched. The efficiency in this case would be expressed as $O(1)$, because only one comparision was made.

2. Average case time complexity
The average case time complexity is the execution of an algorithm having typical input data of size 'n', thus it is the function defined by the average number of steps taken on any instance of size 'n'.

Example:-

The average case efficiency can be obtained by finding average number of comparision.

minimum no. of comparision = 1
Maximum " " " = n

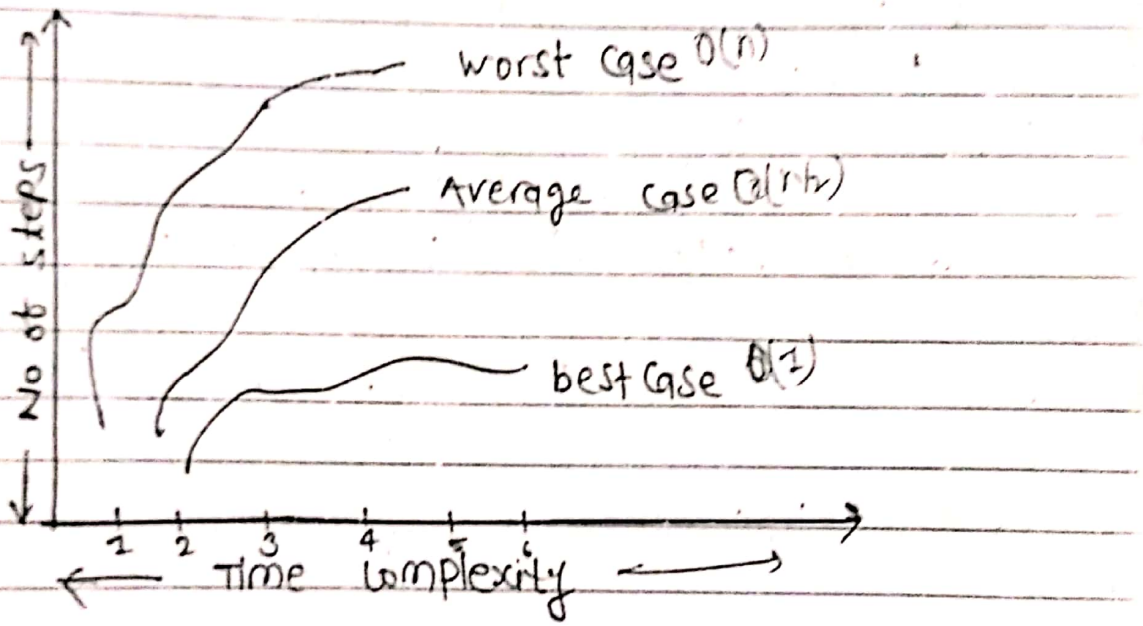
∴ Average no. of comparision = $\frac{(n+1)}{2}$

$(n+1)/2$ is a linear ~~data structure~~ function of n, therefore the average case efficiency will be expressed as $O(\frac{n}{2})$.

3. Worst case time complexity
The worst ~~case~~ time complexity is the function

defined by the maximum amount of time needed by an algorithm for an input size 'n'. Thus, it is the function defined by the maximum number of steps taken at any time instance of size 'n'.

The worst case in the scenario would be if the list is traversed and the element is found at the end of the list or in not found in the list. The efficiency of this case would be expressed as O(n) because 'n' comparisions were made.



Types of Asymptotic notation

2] Big-Oh Notation (O):

The upper bound for the function 'f' is provided by the big oh notation (O). It is the asymptotic notation for the worst case, or ceiling of growth for a given function.

It provides us with an asymptotic upper bound for the growth rate of runtime of an algorithm

Definition

Considering 'g' to be a function from non-negative integers into the positive real numbers. The O(g) is the set of function f, also from non-negative integers to the positive real numbers, such that for some real constant >0 and some non-negative integers constant n_0, $f(n) \leq cg(n)$ for all n>numbers. The set O(g) is usually called as "Oh of g" or "Big oh of g". O(g) is described as a set, it is good practise to say "f is Oh of g" or "f is Big oh of g".

In general, $O(g) = \{f(n)$; there exists +ve constant such that $0 \leq f \leq g(n)$ for all n, $n \geq n_0\}$

Constant function

$f(n) = 16$
$f(n) = 17$

Then for satisfying the big oh condition, the above function can be expressed as follows:

$f(n) \leq 16 * 1$ where $c = 16$ and $n_0 = 0$
$f(n) \leq 27 * 1$ where $c = 27$ and $n_0 = 0$

Thus for above function we can specify the big oh notation as $O(1)$. so, $f(n) = O(1)$.

Linear function

Consider following linear function

$f(n) = 3n+5$

$f(n) = 2n+3$

for $f(n) = 3n+5$ where 'n' is a least 5, $n \geq 5$

$3n+5 \leq 3n+n \leq 4n$

So, $f(n) = O(n)$

Thus above function bound by the linear function.

For $f(n) = 2n+3$, for $n \geq 3$

$2n+3 \leq 2n+n \leq 3n$ ($c=3$, $n_0=3$)

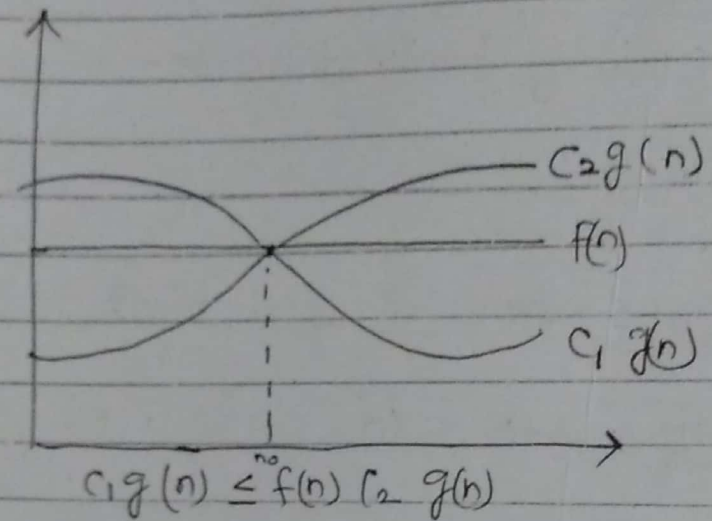
So, $f(x) = O(n)$

Big theta (θ) notation

The lower and upper bond for the function F is provided by the big theta notation.

Consider 'g' be a function from the non-negative integers into the positive real numbers. Then $\theta(g) = O(g) \cap \Omega(g)$ that means the set of

functions that are both in O(g) and Ω(g) where positive constant c_1 and c_2 and n_0 exists such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all n, $n \geq n_0$.



$f(n) = 3n+5$

Sol^n

$f(n) = 3n+5$

$3n \leq 3$ for all n, $c_1 = 3$

Also $3n+5 \leq 4n$ for all $n \geq 5$ $c_2 = 4$ $n_0 = 5$ thus

$3n < 3n+5 < 4n$ $c_1 = 3$, $c_2 = 4$, $n_0 = 5$

So, $f(n) = \Theta(n)$

3. The Big omega (Ω) notation

Let f(x) and g(x) be two function, each from the set of natural numbers or set of positive

real number then $f(n)$ is said to be big omega of $g(x)$. If there exist two positive integers a real number constant(c) and n such that, $f(x) \geq cg(n)$ where $x \geq n$.

This function gives an asymptotic lower bond for a given function.

Constant function

$f(n) = 16$
$f(n) = 27$

Then for satisfying the big Omega condition the above function can be expressed as:.

$f(n) \geq 15 * 1$, where $C = 15$ & $n_0 = 0$
$f(n) \geq 26 * 1$; where $C = 26$ & $n_0 = 0$

So $f(n) = n(1)$ for all above function.