Complexity :-

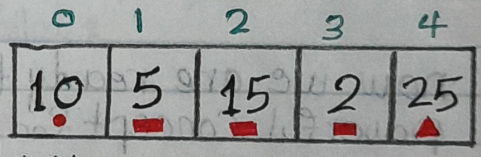
Complexity Meaning :

If we want to write a program we should think about the memory Storage and Time

Def: Complexity Principle includes two parts, Time and Space and we we should detect our Algorithm Complexity to now how to perform its Tasks To make a good performance for our program

for example:

if we have an array as shown in figure and we want to search on an element in it we should detect 3 cases



- Best Case: we found that element in the first index
- Worst Case: we found that element in the last index
- Average Case: we found that element betwin first and last indecies

we denote to them as following: Order

Ω Best Case Θ Average Case O Worst case

and Of, course we will concentrate on the Worst Case

Calculatin Time Complexity:

first thing you Shall Learn about this topic, how To determine The step in your algorithm and we will show you That as following:

ex.

`*` , `/` , `-` , `+` , `%` , `^` , `++` , `--` , `+=` , `-=` , `/=` , `*=` ,
`if` , `else` , `if else`

all the preceding Operations have Constant time

```
for(int i = 1 ; i <= n ; i++){}
```

* and The preceding for loop will have n times because it will be excuted for n times

ex: The following program Calculates $\sum_{i=1}^{n} i$ by two Algorithms we will Compare their Time complexity

1

```
int Sum ;//Declaring Variables out of Calculation
for(int i = 1 ; i <= n ; i++){ //n
    Sum += i ;//1
}
```

2

```
int sum = n*(n+1)/2 ;//1
```

Time Complexity: BigO notation

BestCase → 1 , worstCase → n

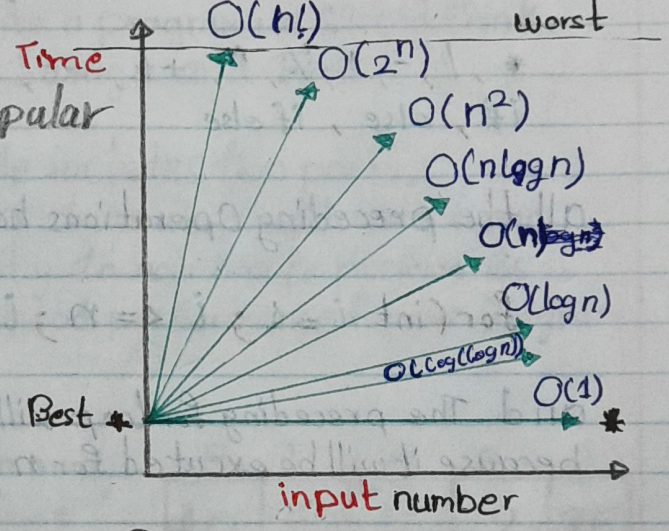
worst Case: $1+n = O(n)$ linear Time Complexity

Best Case: $1 = O(1) < O(n)$

Time Complexity functions :

we will write The Time Complexity functions from The Best to the worst as following :

1. $O(1)$
2. $O(\log n)$ Popular
3. $O(n)$
4. $O(n \log n)$
5. $O(n^2)$
6. $O(2^n)$
7. $O(n!)$
8. $O(\log(\log n))$ $> O(1)$ $< O(\log n)$



Rare

Examples on Time Complexity Functions :

1.

```
for(int i=0; i<n; i++) //n
  for(int k=0; k<n; k++) //n
    print(j+k); //1
```

Time Complexity T.C: $n * n = n^2 = O(n^2)$

[2]

```
int i;
i=1; // 1
for(i; i<n; i*=2) // log n (base 2)
  print(i);
```

T.C: $1 + \log_2 n = \log_2 n = O(\log_2 n)$

note: if the increament or Decreament with * or / Then The Time Complexity will be $\log_b n$ such that b is the Quantity of the operation for any Iteration

note: The nested operation will be represented by * Multiplication of Time Complexities

[3]

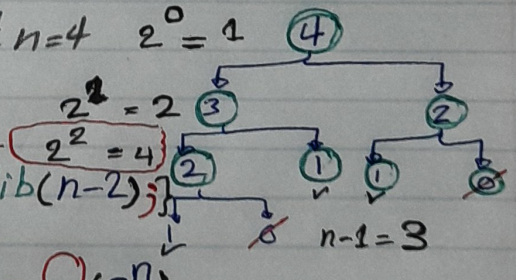
```
int i,j;
i=j=0; // 1
for(i; i<n; i++) // n
  for(j; j<n; j*=3) // log3 n
    print(i+j); // 1
```

T.C: $1 + n\log_3 n * 1 = 1 + n\log_3 n = O(n\log_3 n)$

[4]

Recursion

```
int fib (int n){ let n=4   2^0 = 1
if(n<2)
  return n;   worst
return fib(n-1) + fib(n-2);}
```



T.C: $2^{n-1} = 2^n = O(2^n)$

Big-O-Notation:

Def: It's an abbreviation for Order and represents a function that returns Big Order in some Complexity function of n

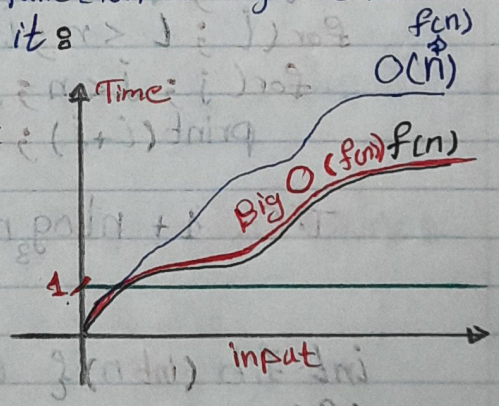
ex: $f(n) = 6n^2 + 100n + 300$

$O(f(n)) = O(n^2)$

and that represents rate of change for time while n is increasing until approaces $+\infty$ (worst Case)

Mathematically $O(f(n)) \geq f(n)$

and the order function is a limiting function which be up to the original function and you can show the follow in figure for it:



note: $O(f(n)) \not< f(n)$ Important

The Big O is The least function up to the original function which be greater Than or equals it

$n!$	$n < 10$
2^n	$n < 20$
n^3	$n < 500$
n^2	$n \leq 10^4$
$n \log n$	$n < 10^5$
n	$n < 10^7$

fit in one Second limit at least

MATHDef: $f(n) = O(g(n))$, $\exists\ c, n_0 \ni f(n) \leq c \cdot g(n)$

$\forall n \geq n_0$

Proof:

let $f(n) = 2n + 6 = O(n)$??

Assume that: $f(n) = 2n+6$

$g(n) = n$

we want to prove that $f(n) = O(g(n))$

let $C_0 = 3$, find n_0

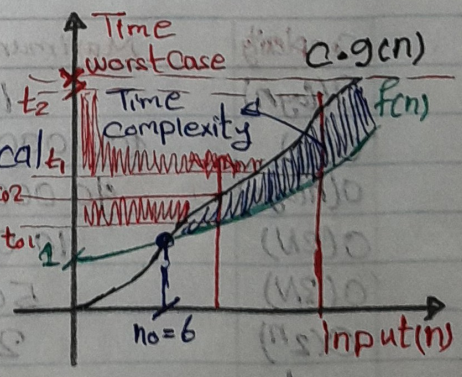
by Def: $2n+6 \leq 3n$

$6 \leq 3n - 2n \Rightarrow n = n_0$

$\Rightarrow 4n \geq (3n+6)$

That's mean at $n_0 = 6$ The $c \cdot g(n)$ will be greaterthan $f(n)$ for $+\infty$

and the following graph will explain The Mathematical Definition above



Space Complexity:

Def: Is a concept of rate of change of the storage which The algorithm takes in the memory and also we represented by Big O notation and we Concentrate on the worst case

we will write them from the best to worst as following:

1. $O(1)$ ex: `int a = 1;`
2. $O(n)$ ex: `int[] a = new int[n];`
3. $O(n^2)$ ex: `int[][] a = new int[n][n];`
4. $O(n \log n)$ ex: `int[][] a = new int[n][(int)Logn];`
5. $O(Length)$ ex: `String s = "abcd...z";`
6. $O(Length * n)$ ex: `String[] s = new String[n];`

Complexity	Maximum N
O(logn)	10^{18}
O(n)	100 000 000
O(nlogn)	40 000 000
O(2N)	10 000
O(3N)	500
$O(2^n)$	20
n!	11