

# 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 should detect our Algorithm Complexity to know 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

0	1	2	3	4
10	5	15	2	25

- 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 between first and last indices

We denote them as following:

$\Omega$  Best Case     $\Theta$  Average Case     $\mathcal{O}$  Order  
Worst Case

and Of course we will concentrate on the Worst Case

## Calculatin Time Complexity

first thing you should 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 executed 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 times  
     sum += i; // 1  
 }

2]

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

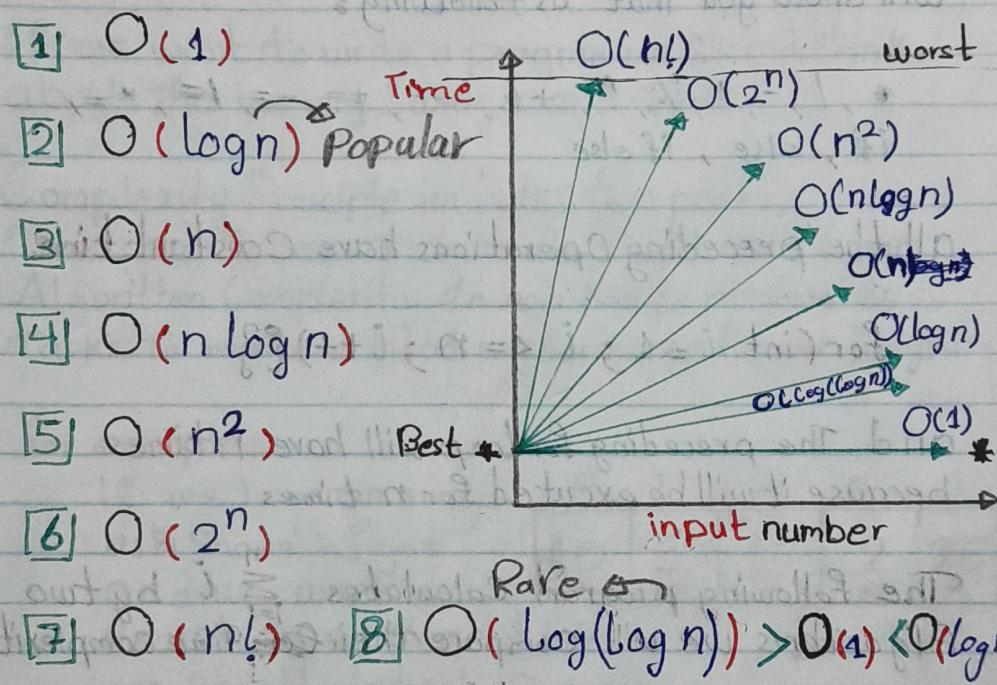
Time Complexity:

Best Case  $\leftarrow$  Worst Case  $\rightarrow$  Big O notation

worst Case 1:  $1+n = O(n)$  Linear Time Complexity  
Best Case 2:  $1 = O(1) < O(n)$

## Time Complexity Functions

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



## 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
```

$$\text{TimeComplexity T.C: } n * n = n^2 = O(n^2)$$

2

```

int i;
i=1; // 1
for( int i; i < n; i+=2) // log2n
    print(i);

```

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

note:

if the increment or Decrement with \* or / Then  
 The Time Complexity will be  $\log b$  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) // log3n
        print(i+j); // 1

```

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

4

Recursion

```

int fib(int n) {
    if(n < 2)
        return n;
    return fib(n-1) + fib(n-2);
}

```

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

# 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 approaches  $+\infty$  (worst case).

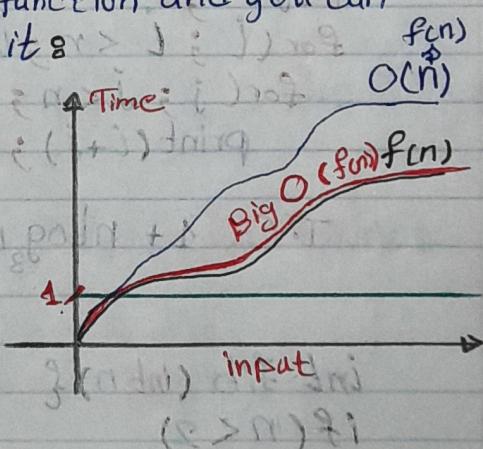
Mathematically,  $O(f(n)) \geq f(n)$

and the order function is a limiting function - which be up to the original function and you can show the following figure for it.

## Notes:

$O(f(n)) \neq f(n)$  Important

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



fit in one second limit at least

$n^6$	$n < 10$
$2^n$	$n < 20$
$n^3$	$n < 500$
$n^2$	$n < 10^4$
$n \log n$	$n < 10^5$
$n$	$n < 10^7$

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  $\exists f(n) = 2n + 6$   
 $g(n) = n$

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

$\therefore [f(n)]_{\text{worst}} = [2n]_{\text{worst}} (n) \quad \square$

let  $C_0 = 3$ , find  $n_0$

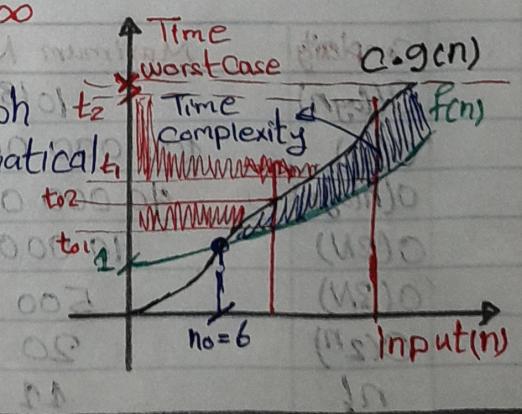
by Def,  $2n+6 \leq 3n$

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

$\Rightarrow 6 \leq 3n \geq (3n+6) \quad \square$

That's mean at  $n_0 = 6$  The  $C \cdot g(n)$  will be greater than  $f(n)$  for  $\infty$

and the following graph will explain The Mathematical Definition above



## Space Complexity

Def:

The 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][n \log n];`

5  $O(\text{length})$  ex: `String s = "abcd - z";`

6  $O(\text{length} * n)$  ex: `String [] s = new String [n];`

Complexity	Maximum N
$O(\log n)$	$10^{18}$
$O(n)$	100 000 000
$O(n \log n)$	40 000 000
$O(2N)$	10 000
$O(3N)$	500
$O(2^n)$	20
$n!$	12

## Challenges of Experimental Analysis:

There are Three major limitations on Algorithm Analysis and Experimental Studies of running Time :

- [1] you must use the same Software and Hardware.
- [2] you ~~should~~ Should Test all possible inputs cause it may be important.
- [3] An algorithm must be fully implemented in order to execute it to study its running Time experimentally.

## Big- $\Omega$ -Notation:

it's provide an asymptotic notation or way of saying that a function grows at a rate That is "greater than or equal to" That of another

MATHDef: if  $g(n) = \Omega(f(n)) = \Omega(g(n))$

$$\exists c > 0, n_0 \geq 1 \ni f(n) \geq c g(n) \forall n \geq n_0$$

## Big- $\Theta$ -Notation:

That notation Allows you to say that Two functions grow at the same rate

MATHDefs: if  $f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$  That is

$$\exists c', c'' > 0, n_0 \geq 1 \ni c'g(n) \leq f(n) \leq c''g(n)$$

Then  $f(n) = \Theta(g(n))$

Keine Zeugnisse über die Zeitblöcke von  
Funktionsgraphen und Formeln

geht zwischen Beobachtung und Formeln möglichst  
gleichzeitig mit einem Zeitpunkt ist es möglich

Bild-S2-p1

To prove we want to show we are giving evidence  
that there is a warp without a fault graph  
with no "at least" of loops to make a warp

$$(n)P(L = (n, t, N))Q \geq (n, t, N) \text{ in } \mathcal{M}(H)$$

$n < a$   $a < (n, t, N) \in \mathbb{N} \times \mathbb{N}, 0 < t$

Bild-S2-p1

aus einer Tabelle aus der es war  
eine Tabelle aus der es war