

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:

Ω Best Case Θ 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

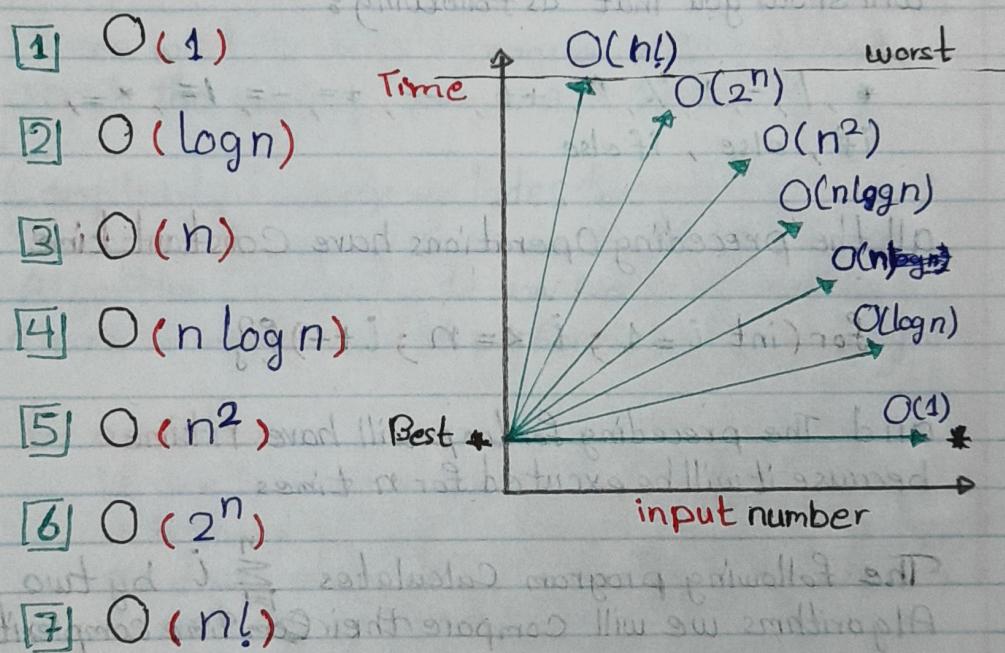
Time Complexity:

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;
    else
        return fib(n-1) + fib(n-2);
}

```

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