Complexity

Performance of the code

Performance of the code is measure how much resource consumes by the code to execute such as time, space etc.

Time: how much time it takes to run the code or no of operations need to perform by your code to accomplish an object

Space: how much extra space or memory need by your code to run

Complexity

Complexity measure how much resources require by your code whenever a change in a large scale is made or whenever the problem gets larger.

Complexity is measure the performance of the code whenever a large no of input is feed to your code. So size of input determine the complexity which determine the efficient algorithm of our code

So complexity is higher less performance of your code.

Complexity always measure in worse case.

Big O notation

It expresses the complexity of an algorithm. It is the unit of complexity

An algorithm whose complexity doesn't change always remain same is call O (1) complexity. No matter whether the size of input is 1 or 10000 or millions its complexity always remain same. It is also called constant complexity.Which is best case

Linear algorithms

If the complexity of an algorithm is O (N) then it is a linear algorithm. Which means the algorithm take 100 sec to process 100 inputs or 10 sec to process 10 inputs (no of inputs=no of times)

As N increases (no of inputs) time also increase linearly

Quadratic algorithms

If the complexity of an algorithms is O (N^2) then it is said to be quadratic algorithms

Basically the sort algorithms.The algorithms take 400 sec to sort 200 elements then it process in quadratic time. I.e. double the time taken by the algorithms

→Example of O (1) constant algorithm

```
Public static void simplification (int n)
{
        Int a=2;
        Int b=6;
        Int sum=a+b;
        Int product=a*b;
}
```

Here no matter what is the value of the input n. algorithm will take same time to produce the result as the input change.

→Example of O (N^2) algorithm

```
Public void loop (int n)
{
        For(int i=0; i<n; i++)
        {
                System.out.println (math.pow(i,2.0));
        }
}
```

No of operation will change as the no of input (n) change. Time increase as n increase

→Example of Linear algorithm O (N)

```
Public void linear (int n)
{
        Int i=0;
        While(i<=n)
                {
                        System.out.println(i)
                        i++;
                }
}
```

Operation will change linearly with the size of the input change.

As the n double the code will take double time to execute

How to calculate complexity

```
Public static worst-case(int n)
{
        IF(n%2)
        {
                System.out.println(even)
        }
        Else
        {
                For(int i=0; i<n; i++)
                {
                System.out.println(i);
                }
        }
}
```

What will be the complexity of the code?

Remember complexity always calculate on worst case

Here on the code the worst case is that the n is odd and we entered the else block so here the complexity id O (N).

If n are even we having best case we get O (1) complexity but complexity is calculating I worse case.

→Additive complexity

```
1. static void MoreExamples1(int[] arr1, int[] arr2)
2.      {
3.              for (int i = 0; i < arr1.Length; i++)
4.              {
5.                  Console.WriteLine(arr1[i]);
6.              }
7.
8.              for (int j = 0; j < arr2.Length; j++)
9.              {
10.                     Console.WriteLine(arr2[j]);
11.                  }
12.             }
```

In this case we have 2 for loops and each iterates through a different set of inputs (arr1 and arr2) and each input size is considered to be very large, so for the first for look the complexity is O (a) and for the second for loop the complexity is O (b) and as both for loops are independent the complexity is additive O (a) + O (b),

O (arr1+arr2)

→Multiplicative complexity

```
1. static void MoreExamples2(int[] arr1, int[] arr2)
2.  {
3.              for(int i =0;i<arr1.Length; i++)
4.              {
5.                  for (int j = 0; j < arr2.Length; j++)
6.                  {
7.                      if (arr1[i] > arr2[j])
8.                      {
9.                          Console.WriteLine(arr1[i] + " , " + ar
   r2[j]);
10.                           }
11.                       }
12.                   }
13.         }
```

This case we have 2 nested for loops and one if condition, so for each element i of arr1 the inner loop j goes through all the elements of arr2, (both arr1 and arr2 are different inputs and the input size is considered to be very large), and the if condition will be executed if the condition is satisfied. so for the first for loop the complexity is O (a), for the inner for loop the complexity is O (b) and for the if condition the complexity is O (1). As it's a nested for loop, the complexity is multiplicative O (a) * O (b) * O(1), we can write O (a*b) or O (ab) where a = arr1.Length and b = arr2.Length.

Note: It's not O (n^2) as there are 2 different inputs arr1 and arr2 of very large size.

O(arr1*arr2)

→Example

```
1. static void MoreExamples3(int[] arr1, int[] arr2)
2.         {
3.             for (int i = 0; i < arr1.Length; i++)
4.             {
5.                 for (int j = 0; j < arr2.Length; j++)
6.                 {
7.                     for (int k = 0; k < 100000; k++)
8.                     {
9.                         Console.WriteLine(arr1[i] + " , " + ar
   r2[j]);
10.                         }
11.                     }
12.                 }
```

In this case we have 3 nested for loops, so for each element i of arr1 and loop j of arr2 the inner loop k goes through 100000 times (both arr1 and arr2 are different and the input size is very large). For the first for loop the complexity is O (a), for the second for loop the complexity is O (b) and for the third for loop the complexity is O (1) (constant time complexity and the input size is fixed – 100000 and would iterate fixed input). As it's a nested for loop we can write the complexity as O (a) * O (b) * O (1), we can ignore O (1) and write the complexity as O (a * b) or O (ab) where a = arr1.Length and b = arr2.Length.

O(arr1*arr2)

→Example of O(log N) complexity

Public static void log_complexity(int n)

{

For(int i=1; i<n; i++)

{

System.out.println(i);

i=i*2;

```
        }
}
```

Iteration 1 i=1 $=2^0$

Iteration 2 i=1*2=2 $=2^1$

Iteration 3 i=2*2=4 $=2^2$

Iteration 4 i=4*2=8 $=2^3$

Iteration (k+1) $i=2^{k-1} * 2=2^k$ $=2^k$

There are some point where 2^k =N and we break out from the loop

I,e value of 2^k (i) equal to N(input) at some point and we out from the loop

So the for loop run k no of times

2^k the value of i is double in every iteration so at what time we break out from the loop. And that will be the complexity of the above code

$2^k = N$

$\text{Log}_2\ 2^k = \log_2 N$ (log base 2)

$K \log_2 2 = \log_2 N$ ($\log_2 2=1$) log 2 of base 2 is =1

$K=\log_2 N$ or $k=\log N$

Since loop will run k no of times so complexity of this code is order of k

And we found value of k is log N

So complexity is O(k) which is O(log N)

Hence complexity is O(logN)

every time we half the input or twice fast the input the complexity will be O(log N)

```
Public static void log complexity(int n)
{
        For(int i=1; i<n; i++)
        {
                System.out.println(i);
                i=i/2;
        }
}
```

Complexity will be O (logN) since we half the input