# **Analysis of Algorithms**

## **Given a problem and 2 algorithms to solve it how do decide which algorithm is better?**

*Approach 1:* Implement both algorithms and run against predefined test cases. But there is problem in this approach because it depends on test cases.It also depends upon server if server has more load it takes more time

*Approach 2*: Asymptotic Analysis is theoretical approach used to measure order of growth (time and space) in terms of input size.

Good thing about asymptotic analysis is it guarantees that if there is algorithm which is asymptotically better there is going to be value of n after which this algorithm will always take less time irrespective of the computer on which we are running

## **What does asymptotic analysis do?**

Asymptotic analysis measures time taken by algorithm in terms of input size

Example 1:

Void func(int n)

{

for(inti=0;i<=n;i++)

{

Println(“hello world”)

}

}

In the above example order of growth of this function in terms of input value n is **linear**because it grows linearly based on input size.

Example 2:

Void func(int n)

{

for(inti=0;i<=n;i++)

{

Println(“hello world”)

}

*for(inti=0;i<=1000;i++)*

*{*

*Println(“hello world”)*

*}*

}

Say I add one more loop with 1000 iterations. In this case first loop takes time based on value n and then loop with 1000 iteration runs. In this case also order of growth is linear. We can ignore constant(which is 1000 fixed iterations)

Example 3:

Void func(int n)

{

for(inti=0;i<=n;i++)

{

*for(intj=0;j<=n;j++)*

{

println(“hello world”);

}

}

}

In this case inner loop runs n times for so in this case order of growth is quadratic

## **What are the asymptotic notations used in analysis of algorithm?**

There are three kind of asymptotic notations.

Theta (Θ):

If we sayΘ(n2) it represents all the functions which has highest growing term as n2 like

100n2 +2n,

1000n2 + n + 3000,

n2,

109n2

Equation should have highest growing term as n2 constants do not matter.

Big O (O):

If we say O (n2) it represents all the functions whose highest order term is n2 or less than n2like

100n2 + 2n

n + 10

10

Omega (Ω):

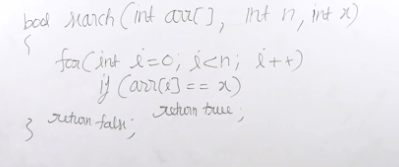
If we say Ω(n2) it represent all the values have same order of growth or highest order of growth like

100n2 + 2n

100n3 + 3n + 1

## **What is best case, average case and worst case?**

Let’s consider below example given an array we need to find element x in the array this function. It returns true if element exits otherwise it returns false.



Best Case: when the element is present in the 0th index.

Average Case: Need to consider all possible input for every possible input we need to find how much time this algorithm takes then we need to add all the possible times and need to take average of it.

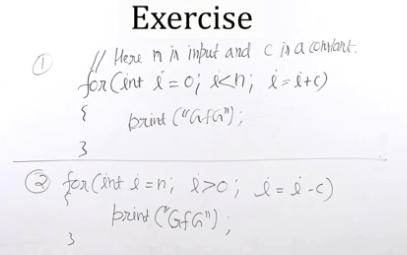
Worst case: in this case worst case is when we want to search element which is last element of the array.

Out of best case, average case and worst case best case and average case mostly not considered we always focus on worst case.

Worst case time complexity of this algorithm is Θ (n)

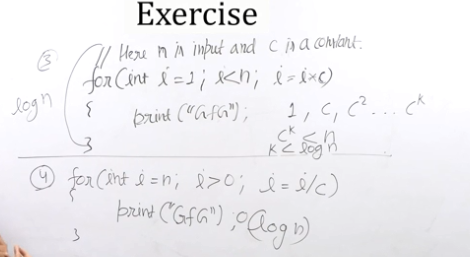
Or generally we can mention time complexity of this algorithm is O (n) because in worst case it is n and in best case it is 1

## **Find order of growth of below loops?**

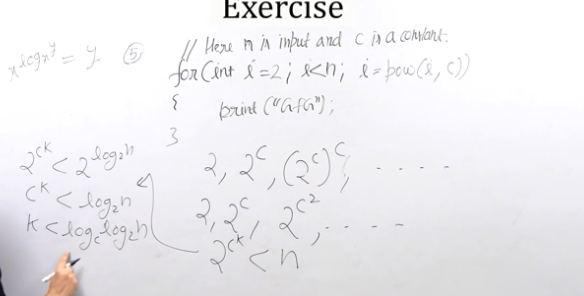


The first loop executed n/c times so in worst case loop is executed n/c times time complexity is Θ(n)

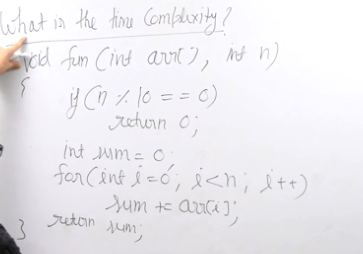
Second loop also executes n/c times so time complexity is Θ (n)



log n is the time complexity of the above 2 loops.



Time complexity of above loop is loglogn

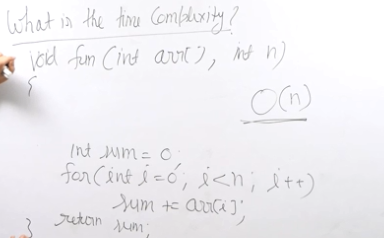


We can say time complexity of this function as O(n) because in worst case it takes linear time.

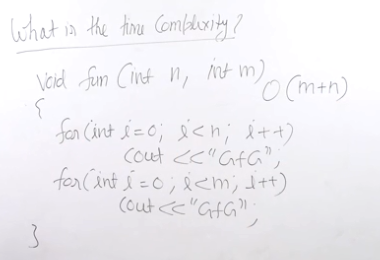
Can we say Ɵ (n) no because if the number is multiple of 10 then control exits before running for loop

So time complexity in worst is Ɵ (n)

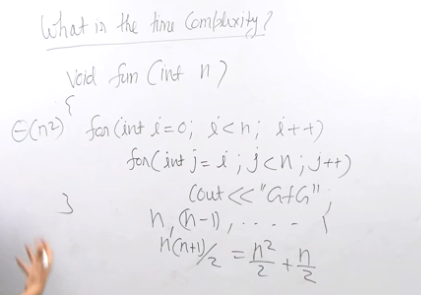
Time complexity in best case Ɵ (1)



Suppose if we remove above mod condition then we can say time complexity of algorithm is Ɵ (n)



Time complexity of above algorithm is O(m+n)



## **What is the time complexity of merge sort and quick sort?**

# **Analysis of Recursion**