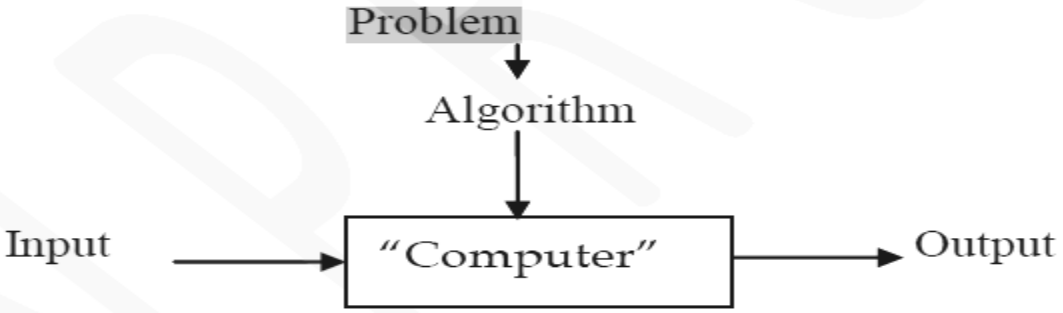
UNIT-I: ALGORITHM ANALYSIS

Algorithm:

Algorithm is a finite set of instructions that if followed accomplishes a particular task.

OR

An algorithm is a sequence of unambiguous instructions for solving a problem, for obtaining a required output for any legitimate input in a finite amount of time.



Characteristics:

Input: Every algorithm takes zero or more inputs.

Output: An algorithm should produce at least one output.

Definiteness: Every instruction must be definable thus implementable by some language of choice.

Finiteness: An algorithm should terminate after a finite number of steps.

Effectiveness: Every instruction must be effective enough.

Efficiency of an algorithm is estimated by Time complexity and Space Complexity of it.

Time Complexity:

Time Complexity is the amount of time it takes to run to Completion

It can be divided in to

Compile Time (Fixed) complexity: (Tf) It is fixed one such as time needed for method name execution etc...This is uncountable.

Run Time (Variable) complexity :(Tv) It is not fixed and depends on particular problem instance.

T(n)=Tf+Tv

Tf<<<Tv

Hence Cp is ignored

T(n)=Tv

Ex: Sum of N Numbers (Iterative method)

Instruction	Units
Initialize a[](Read n,sum=0 are fixed time)	1
While n>0	n+1
Sum=sum+a[i]	n
Return sum	1
	2n+3

Time complexity for sum of n numbers is **T (n) =2n+3**

Ex: Sum of N Numbers (Recursive method)

Instruction	When n<=0	When n>0
Read n(Read n is fixed time)	---------	---------
If n<0	1	1
return 0	1	-------
Return(n+Rsum(n-1))	0	1+TRsum(n-1)

When n>0

T(n) =2 when n<=0

=2n+2 when n>0

Space Complexity: The amount computer memory needed o run to completion

It can be divided in to fixed part and Variable part

Fixed part: (Sf) The fixed memory needed such as memory for procedure instructions

Variable Part:(Sv) It depends on the size of the variables

Fixed part<<<<variablepart

$S(n)=S_f+Sv$ **S(n)=Sv**

Ex: Sum of N Numbers (Iterative method)

Instruction	Units
Read n,sum=0,a[],i	One location for each of **n,i,sum** and n locations for a[]. Hence it is **n+3**
While n>0	
Sum=sum+a[i]	
Return sum	

Hence S(n)=n+3

Ex: Sum of N Numbers (Recursive method)

Instruction	Units
Read n	**When n<=0:** One location for each of n, i. **when n>0:** 3n+1.For each record in stack to store value of n,n-1 and return address.
If n<=0	
return 0	
Return(n+Rsum(n-1))	

Hence S(n)=1 when n<=0

=3n+1 when n>0

Every algorithm takes the complexity one of the below form only

Function	Name
1	Constant
log n	Logarithmic
n	Linear
n log n	n log n
n^2	Quadratic
n^3	Cubic
2^n	Exponential
n!	Factorial

Asymptotic notations:

Asymptotic analysis of an algorithm refers to defining the mathematical boundation/framing of its run-time performance. Using asymptotic analysis, we can very well conclude the best case,
Average case and worst case scenario of an algorithm.

Usually, time required by an algorithm falls under three types –

Best Case − Minimum time required for program execution.
Average Case − Average time required for program execution.
Worst Case − Maximum time required for program execution.

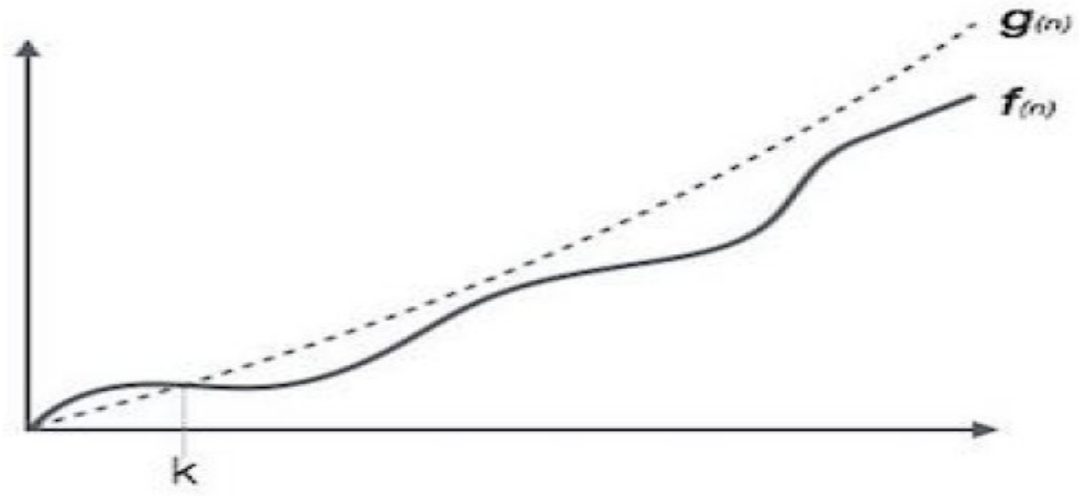
Big-Oh Notation (O): **A function f(n)=O g(n) iff there exists a positive constants c,n_0 such that f(n)<=C.g(n) i.e** It measures the worst case time complexity or longest amount of time an algorithm can possibly take to complete. It defines Upper bound on the algorithm complexity.

Ex: f(n)=2n+3 <=6n where g(n)=n Hence 2n+3=O(n)

$f(n) = 10n^2 + 4n + 2$
When $n \geq 2$, $10n^2 + 4n + 2 \leq 10n^2 + 5n$
Hence $f(n) = O(n^2)$, here c = 11 and n0 = 5

$f(n) = 6*2^n + n^2$
When $n \geq 4$, $n^2 \leq 2n$
So $f(n) \leq 6*2^n + 2n = 7*2^n$
Hence $f(n) = O(2^n)$, here c = 7 and $n_0 = 4$

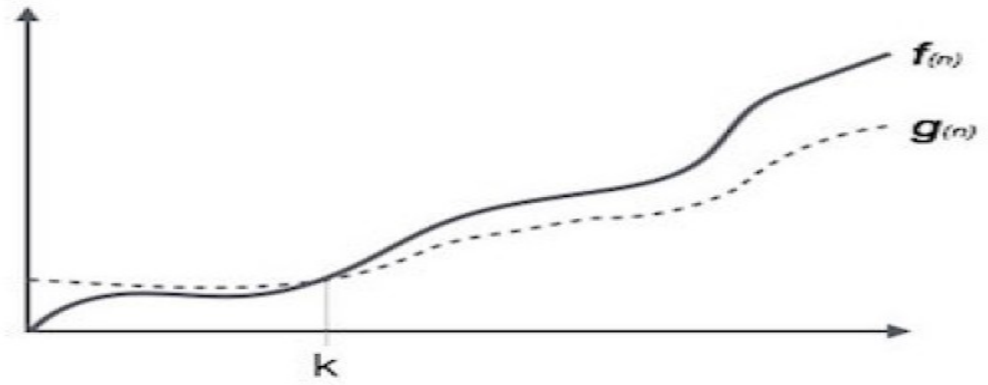
It constrains on selection of smallest g(n) possible.



Omega Notation (Ω) : **A function f(n)= Ω g(n) iff there exists a positive constants c,n_0 such that f(n)>=C.g(n) i.e** It measures the best case time complexity or best amount of time an algorithm can possibly take to complete. It defines Lower bound on the algorithm complexity.

Ex: f(n)=$3n^2+3$ >=$2n^2$ where g(n)= $2n^2$ Hence $3n^2+3$ =O(n^2)

It constrains on selection of Largest g(n) possible.



Theta Notation (θ): A function f(n)= θ g(n) iff there exists a positive constants c1,c2,,n_0 such that C1g(n)<=f(n)<=C2g(n) The θ*n* is the formal way to express both the lower bound and upper bound of an algorithm's running time.

Ex: f(n)=2n+3 <=6n where g(n)=6n Hence 2n+3=O(n)
as 2n<=2n+3<=6n

