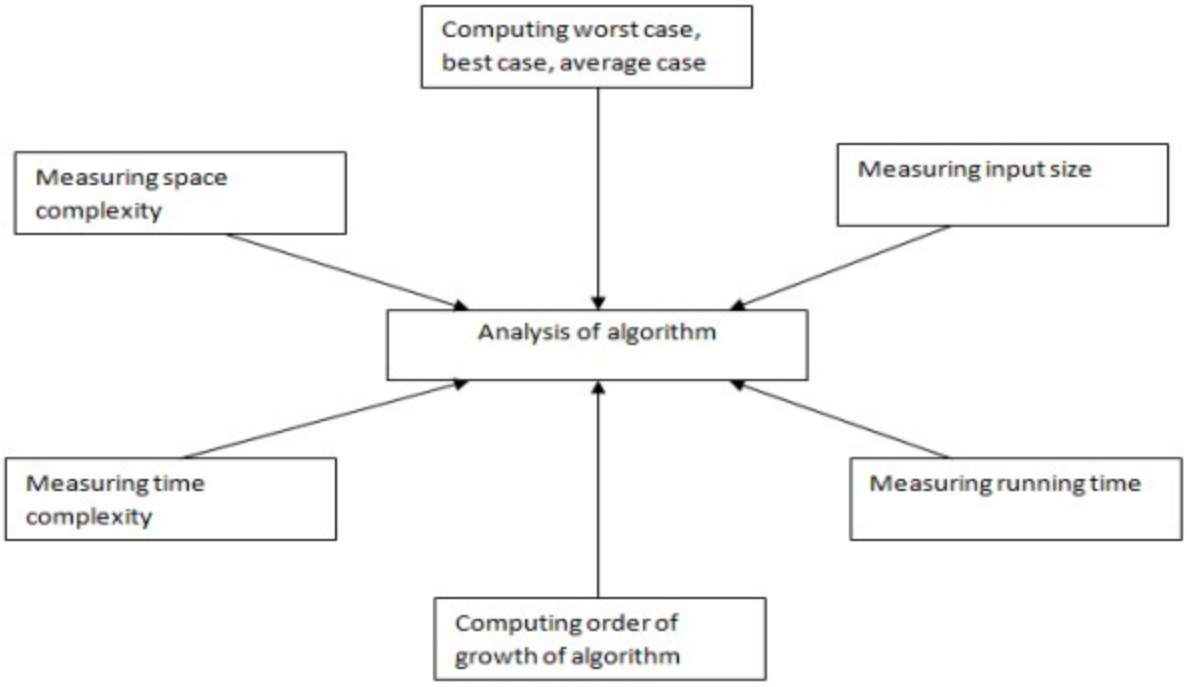
WEEK-2

Algorithm Analysis – Space Complexity, Time Complexity. Run time analysis. Asymptomatic notations, Big-O Notation, Omega Notation, Theta Notation.

Algorithm Analysis:

The efficiency of algorithm can be decided by measuring the performance of an algorithm.



The performance of the algorithm can be computed based on the following factors

1. Space efficiency OR Space Complexity
2. Time efficiency OR Time Complexity

Space complexity:

- Space complexity can be defined as amount of memory required to run the algorithm.
- The following components are important in calculating the space.

1. Instruction space: - Instruction Space is used to store the machine code generated by the compiler.
2. Data Space: - Data Space is used to store constants, static variables, intermediate variables and variables
3. Stack space:- Stack Space is used to store the return address and return values by the function.

These details are used in stack segment.

- The space complexity (Sp) can be given as

Total space = Sp+Cp

Where

Sp -> Space required for the activation record (Instances and variables) that is dynamic part

Cp -> Space required for the constants that is static part

Ex:

```
a = 10
b = 20
c = 30
```

```
avg = (a + b + c )/3
print(avg)
```

- ✓ Here, a,b,c and avg are the integer variables so that Space is
- ✓ Sp=4*2=8 Bytes => 4 is the number of variables and 2 is the size
- ✓ Cp=1*2=2 Bytes => 1 is the number of constants and 2 is the size
- ✓ Total Space=8+2=10 Bytes

Time complexity:

- ➢ Time complexity of an algorithm signifies the total time required by the algorithm or program to run till its completion.
- ➢ Time complexity of an algorithm is depends on the system configuration and "Input Data".
- ➢ Time complexity is calculated on the frequency count.
- ➢ Frequency count is count denoting number of times required for execution of statements.

Ex1:

```
a = 10
b = 20
c = a + b
```

Here 1 unit required for a= 10, 1 unit is required for b = 20 and 1 unit required for c = a + b so total 3 units time is required.

Worst-case, Best-case and Average-case Efficiencies

Worst-case:- Worst case efficiency of algorithm for input size N is its takes longest time to run the algorithm for all possible input values.

Best-case:- Best-case efficiency of algorithm for input size N is its takes fastest time to run the algorithm for all possible input values.

Average-case: - Average-case efficiency of algorithm for input size N is its takes Normal time to run algorithm for all possible input values.

Ex:

```
ALGORITHM Sequential_Search(A[0,1.....n-1],key)
//Input : An Array A[0,1...n-1] and search value is key
//Output : Returns index matched item with key
           otherwise returns -1.
{
    for i=0 to n-1 do
        if(A[i] = = key) Then
            returns i;
    return -1;
}
```

1. **Best-Case:** In Sequential Search, the element key is searched from the array of N elements. If the key element is present at **first** location in array then algorithm runs for a very short time so the Best-Case efficiency is

$$C_{Best\text{-}case}(n) = 1.$$

2. **Worst-Case**: The key element is searched from the array of N elements. If the key element presents at n th location of array, then algorithm will run for longest time.

$$C_{worst\text{-}Case} = n$$

3. **Average-Case:**
   Let C[j] = Number of comparisons up to j^{th} position in the array A.
   Then total number of comparisons C(n) = C[1]+C[2]+.....+C[n]
   Average-case efficiency is

$$C_{Average\text{-}case}(n) = \frac{n+1}{2}$$

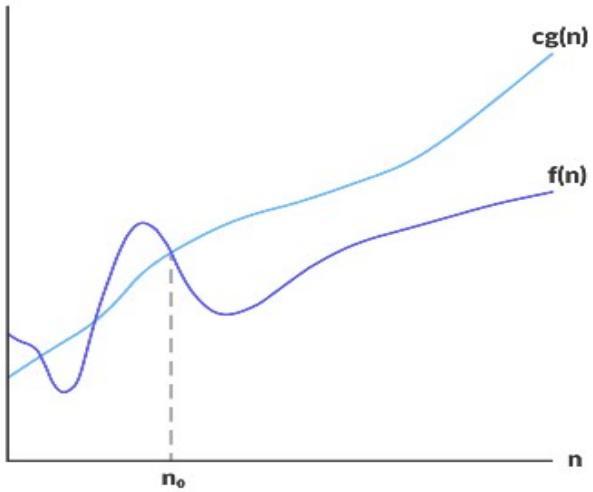
Asymptotic Notations:

- Asymptotic notations are mathematical tools to represents time complexity of the algorithm.
- The efficiency of the algorithm can be measured by computing time-complexity. This time complexity can represent by using asymptotic notations,
- There are 3 asymptotic notations.

1. Big oh notation (O)
2. Omega Notation (Ω)
3. Theta Notation (Θ)

1. Big oh notation

- It represents the upper bound of algorithm running time.
- This notation measures the worst case complexity of the algorithm.
- It indicates the longest amount of time to complete its operations.



f(n) = O(g(n))

- For a function g(n), O(g(n)) is given by the relation:

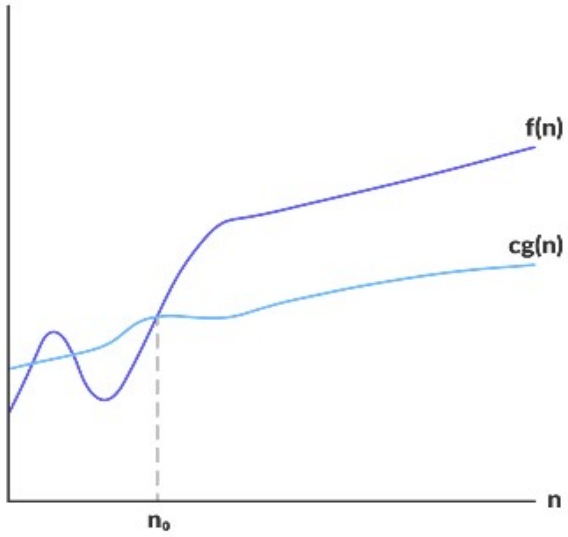
O(g(n)) = { f(n): there exist positive constants c and n0
such that 0 ≤ f(n) ≤ cg(n) for all n ≥ n0 }

- The above expression can be described as a function f(n) belongs to the set O(g(n)) if there exists a positive constant c such that it lies between 0 and cg(n), for sufficiently large n.

Omega Notation:

- It represents the lower bound of the algorithm running time.
- It measures the best case time complexity of the algorithm.

- It indicates the shortest amount of time taken by the algorithm to complete its operation.



f(n) = Ω(g(n))

- For a function g(n), Ω(g(n)) is given by the relation:

Ω(g(n)) = { f(n): there exist positive constants c and n0 such that 0 ≤ cg(n) ≤ f(n) for all n ≥ n0 }

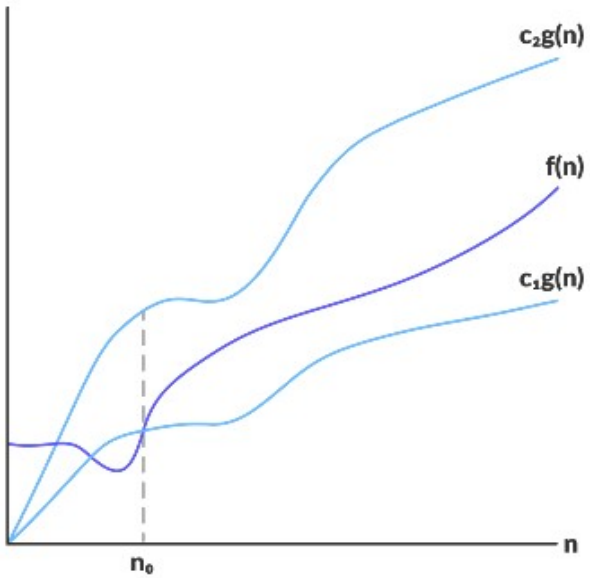
- The above expression can be described as a function f(n) belongs to the set Ω(g(n)) if there exists a positive constant c such that it lies above cg(n), for sufficiently large n.

Theta Notation:

- Its represents the both lower bound and upper bound of algorithm running time.
- Its measure the average case time complexity of the algorithm.
- For a function g(n), Θ(g(n)) is given by the relation:

Θ(g(n)) = { f(n): there exist positive constants c1, c2 and n0 such that 0 ≤ c1g(n) ≤ f(n) ≤ c2g(n) for all n ≥ n0 }

- The above expression can be described as a function f(n) belongs to the set Θ(g(n)) if there exist positive constants c1 and c2 such that it can be sandwiched between c1g(n) and c2g(n), for sufficiently large n.



f(n) = Θ(g(n))

Basic asymptotic efficiency classes:

Name of the efficiency class	Order of growth	Description	Example
Constant	1	As input size grows then we get larger running time	Scanning array elements
Logarithmic	logn	When we get logarithmic running time then it is sure that the algorithm doesn't consider all its input rather the problem is divided into smaller parts on each iteration.	Performing binary search operation.
Linear	n	The running time of algorithm depends on the input size n.	Performing sequential search operation.
nlogn	nlogn	Some instance of input is considered for the list of size n	Sorting the elements using merge sort or quick sort.
Quadratic	n^2	When the algorithm has two nested loops then this type of efficiency occurs.	Scanning matrix elements.
Cubic	n^3	When the algorithm has three nested loops then this type of efficiency occurs.	Performing matrix multiplication.
Exponential	2^n	When the algorithm has very faster rate of growth then this type of efficiency occurs.	Generating all subsets of n elements.
Factorial	n!	When an algorithm is computing all the permutations then this efficiency occurs.	Generating all permutations.