ॐ

(Algorithm Design and Analyse unit-1) 27/09/2019

1 Algorithms:- An Algorithm is a sequence of Computational steps to solve a problem. An Algorithm is a finite set of statement that transform an input into the output. or An Algorithm is any well defined Computational procedure that take some value or set of value as input and produces some value or set of values as output.
An algorithm can be written in any language. So Algorithm is language independent and Algorithm is also hardware independent.

1.1 Characterstics of Algorithm:-

1. Algorithm take some or more input. i.e 0 or more input.
2. Algorithm must generate some output.
3. Definiteness – Algo can be define and understand.
4. Finiteness – Algo must have terminate point.
5. Effectiveness – Algo must be effective.

1.2 Difference between Algorithm and Program.

	Algorithm	Program
1.	Algorithm written at design time.	1. Program written at implementation time
2.	Algo written by the person who have domain knowledge.	2. Program written by the Programmer.
3.	Algorithm can be written in any language.	3. Program can be written in programming language
4.	Algorithm not depened on hardware and software.	4. Program depened on Hardware and Software.
5.	Algorithm Analyse	5. Program can be Testing.

1.3 Analyzing Algorithms:- Analyze an Algorithm has come to mean predicting the resources that the algorithm required. The resources such as memory, Communication bandwidth or Computer hardware are primary concern. But most often we want to measure the Computational time of Algorithm.

The Analysis of algorithm provides background information that give us a general idea of how long an algorithm will take for a given problem set.

The purpose of Analysis of Algorithm is not to give a formula that will tell us exactly how many seconds or Computer Cycles a particular algorithm will take.

1.4 Complexity of Algorithm:-

Efficiency of algorithm depends on two major criteria, first one is run time of that algorithm and second is the space.

One other criteria for Complexity of any algorithm is Comparison of keys and moving of data.

We take three Cases for Complexity of Algorithm-

1. Best Case
2. Worst Case
3. Average Case.

1. Best Case:- Generally most of the Algorithms behave sometimes in best Case. In this case algorithm searches the element in first time. In linear search, if Algo find the element at first time itself then it behaves as best case.

2 Worst Case :- Some times we see the complexity of algorithm in its worst case behaviour, because it tells the behaviour of Algorithm. In this case we find the element at the end or when searching of elements fails.
Worst Case Analysis is necessary in the view that algorithm will perform at least upto this efficiency and it will not go for less than this efficiency.

3. Average case :- Analyzing of average case behaviour of Algorithm is little bit complex than best case and worst case. Here we take probability with list of data.
Average case Algorithm should be the average number of steps but data can be at any place so finding exact behaviour of algorithm is difficult. As the volume of data increases than average case of algorithm behaves likes worst case of algorithm.

1.5 Critera for how to Analyze an Algorithm :-

1. Time taken by Algorithm.
2. Space taken by Algorithm.
3. Network Computation.
4. Power Consumption.
5. CPU Register Consumming.

Ex-1 Algorithm swap two number.

```
Algorithm swap(a, b)
{
    temp = a;   → 1 unit
    a = b;      → 1 unit
    b = temp;   → 1 unit
}
```

Let us take one statement take one unit of time.

Time Complexity -

$f(n) = 1+1+1$

$f(n) = 3$

So $\boxed{f(n) = O(1)}$

Space analysis :-

Total variable used a — 1 unit, b — 1 unit, temp → 1 unit

$S(n) = 1+1+1$

$S(n) = 3$

So $\boxed{S(n) = O(1)}$

Ex-2. Frequency Count method

```
Algorithm Sum(A, n)
{
    s = 0;                          → 1 unit
    for(i = 0; i < n; i++)          → n+1 unit time
    {
        s = s + A[i];               → n time
    }
    return s;                       → 1 time
}
```

Time function $f(n) = 1+n+1+n+1$

$f(n) = 2n+3$

$\boxed{f(n) = O(n)}$

Now Space Complexity :-
Variable used in this algorithm are
A – Array – n elements → n units
n → 1 unit
S → 1 unit
i → 1 unit
Space function S(n) = n + 3
$\boxed{S(n) = O(n)}$

Ex3 Algorithm for Sum of two matricies.

```
Algorithm Add (A, B, n)
{
    for(i=0; i<n; i++)        condition check → n+1 times
    {
        for (j=0; j<n; j++) → n x (n+1) times
        {
            C[i,j] = A[i,j] + B[i,j] → n x n times.
        }
    }
}
```

Time Complexity – $f(n) = n+1 + n(n+1) + n^2$

$f(n) = 2n^2 + 2n + 1$

$\boxed{f(n) = O(n^2)}$

Space Complexity :- Variable used

Array A → $n \times n = n^2$ (elements)
" B → $n \times n = n^2$ "
n C → $n \times n = n^2$ "
n → 1 unit
i → 1 unit
j → 1 unit

So Space function – $S(n) = 3n^2 + 3$

$\boxed{S(n) = O(n^2)}$

Ex-4 Multiplication of two matrices.

```
Algorithm multiply (A, B, n)
{
  for (i=0; i<n; i++) ⟶ n+1 times (Condition changes)
  {
    for (j=0; j<n; j++) → n*(n+1) times
    {
      C[i,j] = 0; ⟶ n*n times
      for (k=0; k<n; k++) → n*n*(n+1) times
      {
        C[i,j] += A[i,k] * B[k,j]; n*n*n times
      }
    }
  }
}
```

Time function $f(n) = (n+1) + n(n+1) + n^2 + n^2(n+1) + n^3$

$f(n) = 2n^3 + 3n^2 + 2n + 1$

So $\boxed{f(n) = O(n^3)}$

Space Complexity – Variable used –

Array	A	$\rightarrow n \times n = n^2$	elements
n	B	$\rightarrow n \times n = n^2$	"
n	C	$\rightarrow n \times n = n^2$	"

Variable n take 1 unit

n	i	1	"
n	j	1	"
n	K	1	"

So Space function $S(n) = 3n^2 + 4$

So $\boxed{S(n) = O(n^2)}$

Ex-5. Time Complexity of for loop if inner for loop depend on the variable of uper for loop.

```
for(i=0; i<n; i++)
{
    for(j=0; j<i; j++)
    {
    }
}
```

Upper loop Condition check n+1 times But lower loop

i	j	no of time
0	0	0 times
1	j=0 j=1	1 0
2	j=0,1 j=2✗	2
3	j=0,1,2	3
⋮	⋮	⋮
n	⋮	n

So inner loop execution time =

$$1+2+3+\cdots+n \Rightarrow \frac{n(n+1)}{2}$$

So $f(n) = n+1 + \frac{n(n+1)}{2}$

$$f(n) = \frac{1}{2}(2n+2+n^2+n)$$

$$= \frac{1}{2}(n^2+3n+2)$$

So $\boxed{f(n) = O(n^2)}$

Ans.

Ex-6

```
p=0
for(i=1; p<=n; i++)
{
  p= p+i;
}
```

i	p
1	0+1 = 1
2	1+2 = 3
3	1+2+3 = 5
⋮	⋮
k	1+2+3 --+k → $p = \frac{k(k+1)}{2}$

Suppose $p > n$ after k times

$$p = \frac{k(k+1)}{2} > n$$

$$k^2 + k > 2n$$

$$k^2 > n$$

$$k = \sqrt{n}$$ so $\boxed{f(n) = O(\sqrt{n})}$

Ex-7

```
for(i=1; i<n; i=i*2)
{
}
```

i
1
1×2 = 2
2×2 = 2^2
2^2×2 = 2^3
⋮
2^k

Assume after k times

$$i >= n \quad \text{---} (1)$$

$i = 2^k$ after k times so put in eqn (1)

$$2^k >= n$$

$$k = \log_2 n$$

So $\boxed{f(n) = O(\log_2 n)}$

Ex-6. `for(i=n; i>=1; i=i/2)`
{
}

1. $i = n$
2. $i = \frac{n}{2}$
3. $i = \frac{n}{2^2}$
4. $i = \frac{n}{2^3}$
$\vdots$
k. $i = \frac{n}{2^k}$

Suppose after k times
$i < 1$ ——— (1)
Value of i after k times is $\frac{n}{2^k}$

So $\frac{n}{2^k} \leq 1$

$2^k = n$ (let)

$k = \log_2 n$

So $\boxed{f(n) = O(\log_2 n)}$

Ex-7.
```
p=0
for(i=1; i<n; i=i*2)        ——— log n times.
{                                So p = log n
  p++;
}
for(j=1; j<p; j=j*2)
{
}
```
log p times
So log log n times.

So $\boxed{f(n) = \log(\log n)}$

Ex-8.
```
for(i=0; i<n; i++)          ——— n+1
{
  for(j=1; j<n; j=j*2)      n * log n times
  {
  }                         n log n times
}
```
$f(n) = 2n\log n + n + 1$

$\boxed{f(n) = O(n \log n)}$

1. for(i=0; i<n; i++) → O(n)

2. for(i=0; i<n; i=i+2) → $\frac{n}{2}$ → O(n)

3. for(i=n; i>1; i--) → O(n)

4. for(i=1; i<n; i=i*2) → $O(\log_2 n)$

5. for(i=1; i<n; i=i*3) → $O(\log_3 n)$

6. for(i=n; i>1; i=i/2) → $O(\log_2 n)$

Note - This is the Analyzation of loop time Complexity. Similarly we can Calculate other loop (while or dowhile) Complexity.

⇒ Compare Class of functions :-

Types of function.

If Time Complexity is O(1) then Constant function

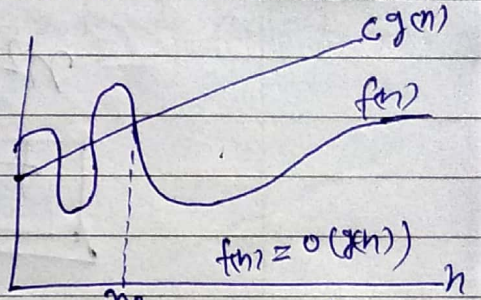
1. f(n) = 2 ⇒ O(1)
   f(n) = 500 ⇒ O(1)
2. O(log n) logrithem function.
3. O(n) Linear "
4. $O(n^2)$ Quadratic "
5. $O(n^3)$ Cubic "
6. $O(2^n)$ or $O(3^n)$ ... $O(n^n)$ is Exponent function.

$1 < \log n < \sqrt{n} < n \log n < n^2 < n^3 < n^4 < \cdots 2^n < 3^n \cdots < n^n$

so $n^k < 2^n$

Asymptotic Notation :- Asymptotic notation is a shorthand way to write down and talk about fastest possible and slowest possible running times for an algorithm by using high and low bounds on speed.
These are also referred as best case and worst case respectively.
Asymptotic notation allow the comparison of the performances of various algorithms.
The Asymptotic notation are

1. Big oh notation
2. Big - omega notation.
3. Theta notation.



diagram

1. Big-oh notation :- Big-oh is the formal method of expressing the upper bound of an algorithms running time. It is measure of the longest amount of time take by the algorithm for completion.

The function $f(n) = O(g(n))$
if $\exists$ two Constant C and n_0
such that $f(n) \leq C * g(n)$ $\forall$ $n \geq n_0$.

Ex-1 $f(n) = 2n+3$

$2n+3 \leq 2n+3n$ forall $n \geq 1$
$n_0 = 1$

$2n+3 \leq 5n$
$\downarrow \quad \downarrow$
$C \quad g(n)$

So $\boxed{f(n) = O(n)}$

Ex-2. $f(n) = 2n^2 + 3n + 4$

Sol: $2n^2 + 3n + 4 \leq 2n^2 + 3n^2 + 4n^2 \quad \forall n \geq 1$

$\leq 9n^2$

$\downarrow c \quad \downarrow g(n)$

So $\boxed{f(n) = O(n^2)}$

Ex-3 $f(n) = n^2 \log n + n$

$n^2 \log n + n \leq 10 n^2 \log n$

$\boxed{f(n) = O(n^2 \log n)}$

Ex-4 $f(n) = n!$

$f(n) = n * (n-1) * (n-2) \cdots 3 * 2 * 1$

$= 1 * 2 * 3 * \cdots (n-2) * (n-1) * n$

So $1 * (*1 \cdots *1 \leq 1 * 2 * 3 \cdots (n-1) * n \leq n * n * n \cdots * n$

$1 \leq n! \leq n^n$

So $\boxed{f(n) = O(n^n)}$

But we can not found the theta bound or Avg bound for this function because it is not have same value for lower and upper bound. Or we can say we do not know the lower bound of this function.

Ex-5 $f(n) = \log n!$ $\quad 1 \leq \log(1 * 2 * 3 \cdots n) \leq \log(n * n * n \cdots * n)$

$1 \leq \log n! \leq \log n^n$

$\log n! \leq n \log n$

So $\boxed{f(n) = O(n \log n)}$

Big-Omega notation: The lower bound of a function f is provided by the big omega notation (Ω). It measure the smallest amount of time for completion of an algorithm.

A function $f(n) = \Omega(g(n))$ iff $\exists$ the constant C and n_0 such that $f(n) \geq C(g(n)) \quad \forall \; n \geq n_0$

Ex1 $f(n) = 2n+3$

$2n+3 \geq 2n \quad \forall \; n \geq 1 \quad (n_0 = 1)$

C $g(n)$

So $\boxed{f(n) = \Omega(n)}$

Ex2: $f(n) = 27n^2 + 16n$

$27n^2 + 16n \geq 27n^2 \quad \forall \; n \geq 1 \quad (n_0 = 1)$

So $\boxed{f(n) = \Omega(n^2)}$

Ex3. $f(n) = 2n^3 + n^2 + 2n$

$2n^3 + n^2 + 2n \geq 2n^3 \quad \forall \; n \geq 1$

Skip $n^2 + 2n$

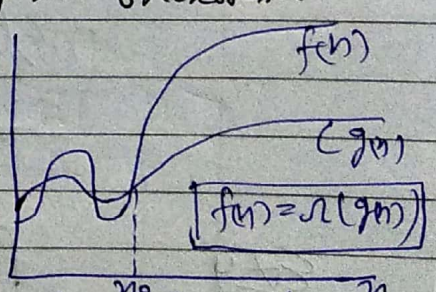
So $\boxed{f(n) = \Omega(n^3)}$

Ex4. $f(n) = 27$

If we have only constant value of $f(n)$

then $\boxed{f(n) = \Omega(1)}$

Diagram of Ω-notation



3. **Theta notation**: The lower and upper bound for a function f is provided by the theta notation (Θ).

The function $f(n) = \Theta(g(n))$ iff $\exists$ two Constant c_1, c_2 and n_0

Such that $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$

Ex $f(n) = 2n+3$

$2n \leq 2n+3 \leq 2n+3n$

$2n \leq 2n+3 \leq 5n \quad \forall n \geq 1 \quad n_0 = 1$

So $\boxed{f(n) = \Theta(n)}$

Ex2. $f(n) = 2n^2+3n+4$

$2n^2 \leq 2n^2+3n+4 \leq 2n^2+3n^2+4n^2$

$2n^2 \leq 2n^2+3n+4 \leq 9n^2 \quad \forall n \geq 1 \quad n_0 = 1$

So $\boxed{f(n) = \Theta(n^2)}$

Ex3. $f(n) = n^2 \log n + n$

$n^2 \log n \leq n^2 \log n + n \leq 10n^2 \log n \quad \forall n \geq 1$

So $\boxed{f(n) = \Theta(n^2 \log n)}$

Ex4 $f(n) = n!$

$1 * 1 * 1 * \cdots * 1 \leq 1 * 2 * 3 * \cdots * n \leq n * n * n * \cdots * n$

$1 \leq n! \leq n^n$

We can not found theta bound or any bound for this function because its not have same value of lower and upper bound. We can not find it lower bound.