UNIT-I

ASYMPTOTIC NOTATIONS.

→ To choose the best Algorithm, we need to check efficiency of each Algorithm. The efficiency ~~Notations~~ can be measured by computing time complexity of each Algorithm. Asymptotic Notations is a shorthand way to represent the time complexity.

→ Using asymptotic notations we can give time complexity as "fastest possible", "slowest possible" or "average time

→ Various Notations such as Ω, Θ, and O used are called Asymptotic Notations.

Big-Oh Notation:-

The Big-oh Notation is denoted by 'O'. It is a method of representing the upper bound of Algorithm's running time. Using big oh Notations we can give longest amount of time taken by the Algorithm to complete.

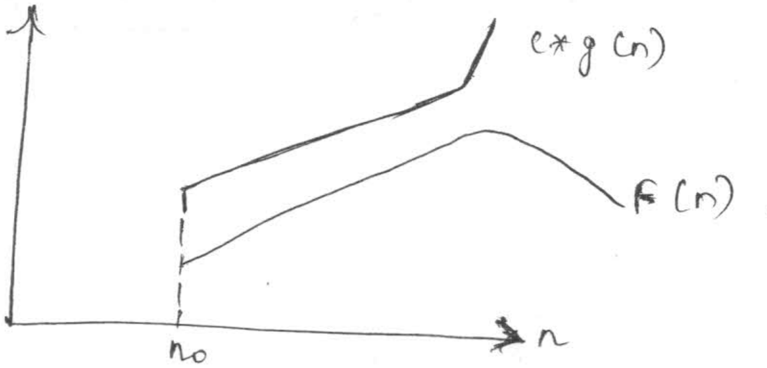
Definition:-

Let $f(n)$ and $g(n)$ be two non-negative functions.

Let n_0 and constant c are two integers such that n_0 denotes some value of input and $n > n_0$. Similarly c is some constant such that $c > 0$. We can write

$$f(n) \leq c * g(n)$$

then $f(n)$ is big oh of $g(n)$ i.e., $F(n) \in O(g(n))$



$F(n) \in O(g(n))$

Example :-

Consider function $F(n) = 2n+2$ and $g(n) = n^2$. Then we have to find some constant c, so that $F(n) \leq c * g(n)$. As $F(n) = 2n+2$ and $g(n) = n^2$ then we find c for $\underline{n=1}$ then,

$f(n) = 2n+2$
$= 2(1)+2$
$F(n) = 4$

and $g(n) = n^2 = (1)^2$
$g(n) = 1$

ie, $F(n) > g(n)$

If $\underline{n=2}$ then,

$f(n) = 2(2)+2 = 6$
$g(n) = (2)^2 = 4$

ie, $F(n) > g(n)$

If $n=3$ then,

$F(n) = 2(3)+2 = 8$
$g(n) = (3)^2 = 9$

ie, $\underline{F(n) < g(n) \text{ is True.}}$

Hence we conclude that for $n > 2$, we obtain

$F(n) < g(n)$

Thus always upper bound of existing time is 'O'.

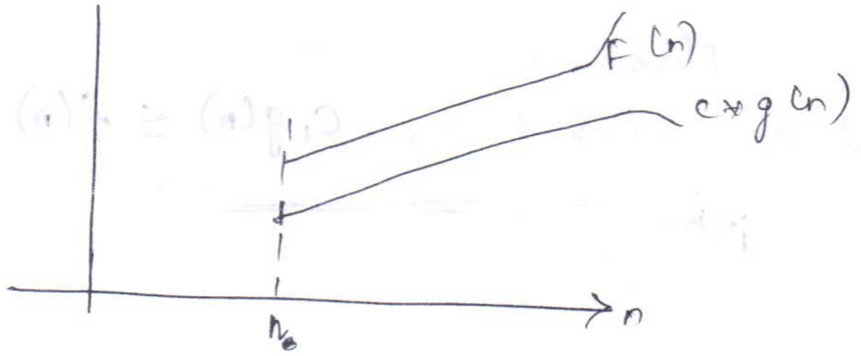
Omega Notations – (Ω)

Omega Notation is denoted by 'Ω'. This Notation is used to represent the lower bound of Algorithms running time. Using Omega Notation we can denote shortest Amount of time taken by Algorithm

Definition :- A function is said to be in Omega (g(n)) i.e., $\Omega(g(n))$ if $f(n)$ is bounded below by some positive Constant Multiple of $g(n)$ such that,

$$f(n) \geq c * g(n) \quad \text{for all } n \geq n_0.$$

It is denoted by $f(n) \in \Omega(g(n))$.



$f(n) \in \Omega(g(n))$

Example:-

Consider $f(n) = 2n^2 + 5$ and $g(n) = 7n$

Then if, $n = 0$

$f(n) = 2(0)^2 + 5 = 5$

$g(n) = 7(0) = 0$ i.e., $f(n) > g(n)$

if, $n = 1$

$f(n) = 2(1)^2 + 5 = 7$

$g(n) = 7(1) = 7$ i.e., $f(n) = g(n)$

if, $n = 3$ then

$f(n) = 2(3)^2 + 5 = 18 + 5 = 23$

$g(n) = 7(3) = 21$

i.e., $f(n) > g(n)$

ie, for $n \geq 3$ we get $f(n) > c * g(n)$.
It can be represented as,

$$2n^2 + 5 \in \Omega(n)$$

Similarly any, $n^3 \in \Omega(n^2)$

Θ NOTATION :-

The theta Notation is denoted by θ. By this method the running time is between upper bound and lower bound.

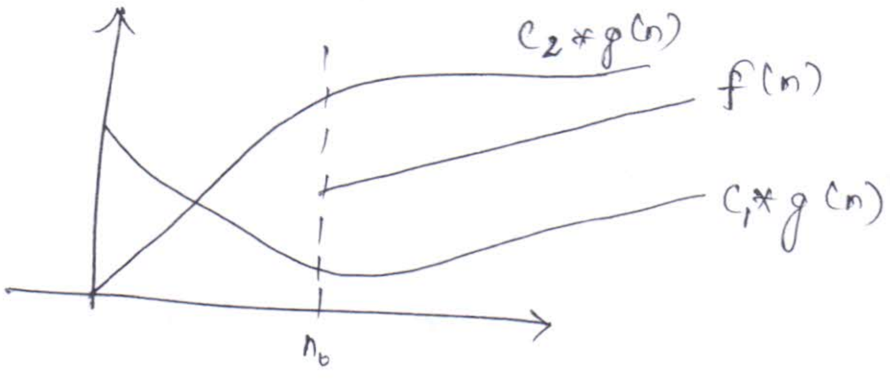
Definition :-

Let $F(n)$ and $g(n)$ be two non-negative functions. There are two positive constants namely c_1 and c_2, such that, $c_1 g(n) \leq f(n) \leq c_2 \cdot g(n)$

$$\cancel{c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)}$$

Then we can say that,

$$F(n) \in \theta(g(n))$$



$$f(n) \in \theta(g(n))$$

Some Examples of Asymptotic Notations,

1) $\log_2 n$ is $f(n)$ then,

$\log_2 n \in O(n)$ ∵ $\log_2 n \leq O(n)$, the order of growth of $\log_2 n$ is slower than

~~$\log_2 n \in \Theta(n$~~

$\log_2 n \in O(n^2)$ ∵ $\log_2 n \leq O(n^2)$, the order of growth of $\log_2 n$ is slower than n^2 as well.

But

$\log_2 n \notin \Omega(n)$ ∵ $\log_2 n \leq \Omega(n)$ and if a certain function $f(n)$ is belongs to $\Omega(n)$ it should satisfy the condition $f(n) \geq c * g(n)$

Similarly $\log_2 n \notin \Omega(n^2)$ or $\Omega(n^3)$

2) Let $f(n) = n(n-1)/2$

Then

$n(n-1)/2 \in O(n)$ ∵ $f(n) > O(n)$

But

$n(n-1)/2 \in O(n^2)$ As $f(n) \leq O(n^2)$

and $n(n-1)/2 \in O(n^3)$

Similarly,

$n(n-1)/2 \in \Omega(n)$ ∵ $f(n) \geq \Omega(n)$

$n(n-1)/2 \in \Omega(n^2)$ ∵ $f(n) \geq \Omega(n^2)$

$n(n-1)/2 \in \Omega(n^3)$ ∵ $f(n) \geq \Omega(n^3)$

PROPERTIES OF ORDER OF GROWTH :-

1. If $F_1(n)$ is order of $g_1(n)$ and $f_2(n)$ is Order of $g_2(n)$, then

$$F_1(n) + f_2(n) \in O(\max(g_1(n), g_2(n))).$$

2. polynomials of degree $m \in \Theta(n^m)$
3. $O(1) < O(\log n) < O(n) < O(n^2) < O(2^n)$
4. Exponential functions a^n have different order of growth for different values of a.

BASIC EFFICIENCY CLASSES :-

Name of Efficiency class	Order of growth
Constant	1
Logarithmic	$\log n$
linear	n
n Logn	$n \log n$
Quadratic	n^2
cubic	n^3
Exponential	2^n
Factorial	$n!$

(*) Using limits for comparing Order of growth.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} 0, & \text{big oh Notation} \\ C > 0, & \text{theta Notation} \\ \infty, & \text{Omega Notation} \end{cases}$$

Properties of Big-oh :-

1. If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ then $f(n) \in O(g(n))$ but $f(n) \notin \Theta(g(n))$

Example 1:-

Compare the order of growth of $\frac{1}{2}n(n-1)$ and n^2

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

$g(n) = n^2$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{1}{2} \lim_{n\to\infty} \frac{n^2-n}{n^2} = \frac{1}{2} \lim_{n\to\infty} \left(1-\frac{1}{n}\right)$$

$= 1/2$

Since the limit value is $1/2 > 0$ so,

$\frac{1}{2}n(n-1) \in \Theta(n^2)$

Example 2:-

$f(n) = \log_2 n$ $g(n) = \sqrt{n}$, compare order of growth.

$$\lim_{n\to\infty} \frac{\log_2 n}{\sqrt{n}}$$ use L' Hopital rule,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{t'(n)}{g'(n)}$$

and stirling's formula,

$n! \simeq \sqrt{2\pi n}\,(n/e)^n$ for large value of n

Here use L' Hopital rule,

$$\lim_{n\to\infty} \frac{\log_2 n}{\sqrt{n}} = \lim_{n\to\infty} \frac{(\log_2 n)'}{(\sqrt{n})'} = \lim_{n\to\infty} \frac{(\log_2 e)^{1/n}}{\frac{1}{2\sqrt{n}}}$$

$$= 2\log_2 e \lim_{n\to\infty} \frac{\sqrt{n}}{n} = 0$$

Since $f(n)/g(n)$ is equal to zero, $f(n) \in O(\sqrt{n})$

Example 3:-

Compare $n!$ and 2^n.

$f(n) = n!$ and $g(n) = 2^n$

$$\lim_{n\to\infty} \frac{n!}{2^n} = \lim_{n\to\infty} \frac{\sqrt{2\pi n}\,(n/e)^n}{2^n}$$

$$= \lim_{n\to\infty} \sqrt{2\pi n}\,\frac{n^n}{2^n e^n} = \lim_{n\to\infty} \sqrt{2\pi n}\left(\frac{n}{2e}\right)^n = \infty$$

Since $\frac{f(n)}{g(n)} = \infty$, $\therefore$ $n! \in \Omega(2^n)$.

Mathematical Analysis of Non-Recursive Algorithms :-

An Algorithm can be recursive or non-recursi Algorithms.

First we will see the general plan for analyzi the efficiency of non-recursive Algorithms. This plan tells the steps to be followed, while Analyzi such Algorithms.

General plan for Analyzing Efficiency of Non-Recurs Algorithm

1. Decide the input size based on parameter n.
2. Identify Algorithm's basic operations
3. check how many times the basic operation is executed. Then find whether the execution of basic operation depends upon the input size n.

   Determine worst, average and best cases for input of size n.
4. Set up a Sum (Σ) for the number of times the basic operation is executed
5. Simplify the Sum (Σ) using standard formula &

Summation Formula and Rules used in Efficiency Ana

1. $$\sum_{i=1}^{n} 1 = 1+1+1+1+\cdots+1 = n \in \theta(n)$$
2. $$\sum_{i=1}^{n} i = 1+2+3+\cdots+n = \frac{n(n+1)}{2} \in \theta(n^2)$$
3. $$\sum_{i=1}^{n} i^k = 1+2^k+3^k+\cdots+n^k = \frac{n^{k+1}}{k+1} \in \theta(n^{k+1})$$
4. $$\sum_{i=1}^{n} a^i = 1+a+\cdots+a^n = \frac{a^{n+1}-1}{a-1} \in \theta(a^n)$$
5. $$\sum_{i=1}^{n} (a_i \pm b_i) = \sum_{i=1}^{n} a_i \pm \sum_{i=1}^{n} b_i.$$

6) $\sum_{i=1} ca_i = c \cdot \sum_{i=1} a_i$

7) $\sum_{i=k}^{n} 1 = n-k+1$ where n and k are upper and lower limits.

Examples for Non-Recursive Algorithms:-

1. Finding the element with maximum value in a given array:-

Algorithm : Max_Element (A [0....n-1])

// Problem Description : Finding the maximum value element from the array.

// Input : array A [0.....n-1]

// output: Returns the largest element from Array

```
Max ← A [0]
for i ← 1 to n-1 do
{ if (A [i] > Max) then
      Max ← A [i]
}
return Max
```

2. Finding whether the set of elements in an array are distinct. This problem is called element Uniqueness prob

Algorithm: Uniqueelement (A [0.....n-1])

// problem Description : find whether array elements are distinct or not.

// Input : Array A[0.....n-1]

// output : Return False if elements are not distinct else Return True

```
for i ← 0 to n-2 do
 {
    for j ← i+1 to n-1 do
      { if (A[i] == A[j]) then
          return False.
      }
 }
return True.
```

Mathematical Analysis :-

step 1:- The input size is n.

step 2:- The Basic operation will be comparison of two elements.

step 3:- The number of comparisons will depend upon the input n.

step 4:- $C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$ (outer loop X Inner loop)

(1 is the no. of comparison)

step 5:- Simplify the sum,

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

Since, $\sum_{j=i+1}^{n-1} 1 = (n-1) - (i+1) + 1$

$$= \sum_{i=0}^{n-2} (n-1-i) \Rightarrow \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i$$

$$= (n-1)\sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2} \qquad \left(\because \sum_{i=1}^{n} i = \frac{n(n+1)}{2}\right)$$

$$= (n-1)\sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2} \qquad \left[\because \sum_{i=0}^{n} 1 = (n-1+1) = n\right.$$

$$\text{so, } \sum_{i=0}^{n-2} = (n-2)-0+1 = (n-1)$$

Solving this eqn/: we will get,

$$= \frac{2(n-1).(n-1) - (n-2)(n-1)}{2}$$

$$= 2(n^2-2n+1) - n^2 - 3n+2) / 2$$

$$= (n^2-n)/2 \simeq \frac{1}{2}n^2 \in \Theta(n^2)$$

$\therefore$ Time complexity $= \Theta(n^2)$

3. Write an Algorithm for Multiplication of Matrices using Non-Recursive Algorithm.

<u>Algorithm</u> :- Matrix_Mul (A[0...n-1, 0...n-1], B[0...n-1, 0...n-1])

// problem Description : This Algorithm performs Multiplication of two square Matrices.

// Input : Two Matrices A and B.

// output : C Matrix containing Multiplication of A and B.

Mathematical analysis :-

Step 1:- The input size is n.

Step 2:- The basic operation is in the innermost loop and which is,

$$c[i,j] = c[i,j] + A[i,k] * B[k,j]$$

Step 3:- The basic operation depends upon input size. There are no best case, worst case and average case efficiencies.

Step 4:- The sum can be denoted by M(n)

M(n) = outermost loop x Inner loop x Innermost loop (1 execution)

$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1$$

$$= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n$$

$$= \sum_{i=0}^{n-1} n^2$$

$$M(n) = n^3$$

∴ The Time Complexity of Matrix Multiplication is $\Theta(n^3)$.

4) Counting Numbers of bits in an Integer using Non-Recursive Algorithm.

// problem Description : This Algorithm is for counting binary digits from decimal Integer

// Input :- The decimal Integer n.

// output :- Returns total number of digits from the input.

```
count ← 1
while (n>1)
{  count ← count +1
   n ← ⌊n/2⌋
}
return count.
```

Mathematical Analysis :-

Step 1 :- The input size is n

Step 2 :- The basic operation is denoted by while loop. And it is each time checking whether $n>1$.

Step 3 :- The value of n is halved on each repetition of the loop. Hence efficiency of Algorithm is equal to $\log_2 n$.

Step 4 :- Hence total number of times the while loop gets executed is,

$$\lfloor \log_2 n \rfloor + 1$$

Hence Time complexity for counting number of bits of given number is $\Theta(\log_2 n)$.

RECURRENCE EQUATION & RECURRENCE RELATION

The recurrence equation is an equation that defines a sequence recursively. It is normally in following form -

$$T(n) = T(n-1) + n \quad \text{for } n > 0 \quad \rightarrow (1)$$
$$T(0) = 0 \quad \rightarrow (2)$$

Here equation 1 is called recurrence relation and equation 2 is called Initial condition. The recurrence equation can have infinite number of sequences. The general solution to the recursive function specifies some formula.

For example :- Consider a recurrence relation,

$$f(n) = 2f(n-1) + 1 \quad \text{for } n > 1$$
$$f(1) = 1$$

Then by solving this recurrence relation we get $f(n) = 2^n - 1$. when $n = 1, 2, 3$ and 4.

SOLVING RECURRENCE EQUATIONS :-

The recurrence relation can be solved by following methods -

1. Substitution method
2. Master's Method

SUBSTITUTION METHOD :-

The Substitution method is a kind of method in which a guess for the solution is made.

Two types of substitution method,

→ Forward Substitution

→ Backward Substitution

FORWARD SUBSTITUTION METHOD :-

This method makes use of an initial condition in the initial term and value for the next term is generated.

For example :-

Consider a recurrence relation,

$$T(n) = T(n-1) + n$$

with initial condition $T(0) = 0$

Let, $T(n) = T(n-1) + n \longrightarrow (1)$

If $n=1$ then,

$\because$ Initial condition $T(0) = 0$

$$T(1) = T(0) + 1 = 0 + 1$$

$$T(1) = 1 \longrightarrow (2)$$

If $n = 2$, then

$$T(2) = T(1) + 2 = 1 + 2$$

$$T(2) = 3 \longrightarrow (3)$$

If $n = 3$, then

$$T(3) = T(2) + 3 = 3 + 3 \longrightarrow (4)$$

$$T(3) = 6$$

By observing above generated equations we can derive a formula,

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

we can also denote $T(n)$ in terms of big-oh notation as follows -

$$T(n) = O(n^2)$$

BACKWARD SUBSTITUTION :-

In this method backward value are substitution recursively in order to derive some formula.

For example :-

consider, a recurrence relation.

$$T(n) = T(n-1) + n \longrightarrow (1)$$

with initial condition $T(0) = 0$

$$T(n-1) = T(n-1-1) + (n-1) \longrightarrow (2)$$

putting equation (2) in equation (1) we get,

$$T(n) = T(n-2) + (n-1) + n \longrightarrow (3)$$

Let, $T(n-2) = T(n-2-1) + (n-2) \longrightarrow (4)$

putting equation (4) in equation (3) we get,

$$T(n) = T(n-3) + (n-2) + (n-1) + n$$

$$\vdots$$

$$= T(n-k) + (n-k+1) + (n-k+2) + \dots + n$$

If $k = n$ then

$$T(n) = T(0) + 1 + 2 + \dots + n$$

$$T(n) = 0 + 1 + 2 + \dots + n \qquad (\because T(0) = 0)$$

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

Again we denote $T(n)$ in terms of Bigoh notation

$$T(n) \in O(n^2)$$

$$\because \frac{n^2}{2} + \frac{n}{2} \simeq n^2$$

Example 1:-

Solve the following relation $T(n) = T(n-1) + 1$ with $T(0) = 0$ as initial condition. Also find big oh Notation

Solution:-

Let,

$$T(n) = T(n-1) + 1$$

By Backward substitution,

$$T(n-1) = T(n-2) + 1$$

$$\therefore T(n) = \underbrace{T(n-1)}_{\downarrow} + 1$$

$$= \overbrace{(T(n-2) + 1)} + 1$$

$$T(n) = T(n-2) + 2$$

Again $T(n-2) = T(n-2-1) + 1$

$$= T(n-3) + 1$$

$$\therefore T(n) = \underbrace{T(n-2)}_{\downarrow} + 2$$

$$= (T(n-3) + 1) + 2$$

$$T(n) = T(n-3) + 3$$

$\vdots$

$$T(n) = T(n-k) + k \longrightarrow (1)$$

If $k = n$ then equation (1) becomes

$$T(n) = T(0) + n$$

$$T(n) = 0 + n \simeq n$$

[$\because$ Initial condition $T(0) = 0$.]

$$T(n) = n$$

$\therefore$ we can denote $T(n)$ in terms of big oh notation.

$$T(n) = O(n)$$

Example 2:-

$$x(n) = x(n-1) + 5 \quad \text{for } n>1,\ x(1)=0$$

$$= [x(n-2)+5]+5$$

$$= [x(n-3)+5]+5+5$$

$$= x(n-3)+5*3$$

$$\vdots$$

$$= x(n-i)+5*i$$

If $i=n-1$ then,

$$= x(n-(n-1))+5*(n-1)$$

$$= x(1)+5(n-1)$$

$$= 0+5(n-1) \qquad \because x(1)=0.$$

$$\therefore x(n) = 5(n-1) = 5n-5$$

$$\therefore x(n) \in O(n)$$

Example 3:-

$$x(n) = 3x(n-1), \quad \text{for } n>1,\ x(1)=4.$$

$$= 3[3x(n-2)] = 3^2 \cdot x(n-2)$$

$$= 3\cdot 3[3x(n-3)] = 3^3 \cdot x(n-3)$$

$$\vdots$$

$$= 3^i \cdot x(n-i)$$

If we put $i=n-1$ then

$$= 3^{(n-1)}\, x(n-(n-1))$$

$$= 3^{(n-1)}\, x(1)$$

$$x(n) = 3^{(n-1)} \cdot 4 \qquad \because x(1)=4$$

$$x(n) \in O(3^n)$$

Example 4:-

$$x(n) = x(n/2) + n \quad \text{for } n>1, \quad x(1)=1$$

put $n = 2^k$, then,

$$x(2^k) = \cancel{x}[2^k/2] + 2^k$$
$$= x(2^{k-1}) + 2^k$$
$$= [x(2^{k-2}) + 2^{k-1}] + 2^k$$
$$= [x(2^{k-3}) + 2^{k-2} + 2^{k-1} + 2^k$$
$$\vdots$$
$$= \cancel{x}(2^{k-i}) + 2^{k-i+1} + 2^{k-i+2} + \cdots + 2^k$$

Backward sub/:-

$$x(n) = x(n/2) + n$$
$$x(n) = [x(n/4) + n) + n = x(n/4) + 2n = x(n/2^2) +$$
$$= [x(n/8) + n) + 2n = x(n/8) + 3n = x(n/2^3) +$$
$$\vdots$$
$$T(n) = x\left(\frac{n}{2^k}\right) + k.n$$

If we assume $2^k = n$

$$x(n) = x(n/n) + \log_2 n . n \qquad \because T(1) = 1$$
$$= x(1) + n.\log n$$
$$T(n) = n\log(n)$$

Hence in terms of big oh notation

$$T(n) \in O(n \log n)$$

Example 5:-

Solve the recurrence relation

$$T(n) = 2T(n/2) + n$$

Solution : With $T(1) = 1$ as initial condition

$$T(n) = 2\left(2T(n/4) + n/2\right) + n$$

$$T(n) = 4T(n/4) + 2n$$

$$T(n) = 4\left(2T(n/8) + n/4\right) + 2n$$

$$= 8T(n/8) + 3n$$

$$T(n) = 2^3\, T(n/2^3) + 3n$$

$$\vdots$$

$$T(n) = 2^k\, T(n/2^k) + k \cdot n$$

If we assume $2^k = n$

$$T(n) = n \cdot T(n/n) + \log_2 n \cdot n \qquad \because \overset{\text{(given)}}{T(1) = 1}$$

$$= n \cdot T(1) + n \cdot \log n$$

$$T(n) = n + n\log(n)$$

$$T(n) \simeq n\log n$$

Hence in terms of big-oh notations

$$T(n) = O(n \log n)$$

2. MASTER'S THEOREM :-

The recurrence relation can also be solved by using Master's theorem (like substitution method)

Consider the following Recurrence relation,

$$T(n) = a \cdot T(n/b) + F(n)$$

Where $n \geq d$ and d is some constant.

efficiency analysis as,

If $F(n) = \Theta(n^d)$ where $d \geq 0$ in the recurrence relation then,

Formula I:-

1) $T(n) = \Theta(n^d)$ if $a < b^d$

2) $T(n) = \Theta(n^d \log n)$ if $a = b$

3) $T(n) = \Theta(n^{\log_b a})$ if $a > b^d$

<u>Example 1:-</u>

$$T(n) = 4T(n/2) + n$$

we will map this equation with

$$T(n) = aT(n/b) + f(n)$$

Now $f(n)$ is n ie., n^{1} ∴ Hence $d = 1$

$a = 4$ and $b = 2$ and

$a > b^d$ ie., $4 > 2^1$

$$T(n) = \Theta(n^{\log_b a})$$

$$= \Theta(n^{\log_2 4}) = \Theta(n^2)$$

[∴ $\log_2 4 = 2$]

Hence Time complexity is $\Theta(n^2)$

Formula II:-

Another formula to use is,

4) If $f(n)$ is $\Theta(n^{\log_b a} \log^k n)$, then

$$T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$$

Example 2:-

$$T(n) = 2T(n/2) + n\log n$$

Here

$$f(n) = n \log n$$

$$a = 2, \quad b = 2$$

use formula 4

$$f(n) = \Theta(n^{\log_2 2} \log^1 n) \qquad \text{ie., } k = 1$$

$$\text{Then } T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$$

$$= \Theta(n^{\log_2 2} \log^2 n)$$

$$= \Theta(n^1 \log^2 n)$$

$$\therefore T(n) = \Theta(n \log^2 n)$$

Example 3:-

$$T(n) = 8T(n/2) + n^2$$

Here $f(n) = n^2$

$a = 8$ and $b = 2$, $d = 2$

$$\therefore \log_2 8 = 3$$

use formula 3 $\left[\because \begin{array}{l} a > b^d \\ 8 > 2^2 \end{array}\right]$

$$f(n) = O(n^{\log_2 8})$$

$$T(n) = \Theta(n^3)$$

Example 4:-

$$T(n) = 9T(n/3) + n^3$$

Here $a = 9$, $b = 3$ and $d = 3$

$$a < b^d \qquad (\because 9 < 3^3)$$

so use formula 1

$$f(n) = \Theta(n^d)\cancel{\log n}$$

$$\therefore f(n) = \Theta(n^3)\cancel{\log n}$$