Unit-3

1. Mathematical Induction
2. Staong Induction
3. Recursive function
4. Growth function

Mathematical Induction [In Complete Induction]

① Show that If n is a positive integer then Sum of the natural numbers $= \frac{n(n+1)}{2}$

i.e. $1+2+3+\cdots+n = \frac{n(n+1)}{2}$ by using mathematical induction

Sol Let P(n) [denote]: $1+2+3+\cdots+n = \frac{n(n+1)}{2}$ where n is a positive integer

Basis step:

we can take $n=1$

$$1 = \frac{1(1+1)}{2}$$

$$1 = 1$$

$\therefore$ LHS = RHS

Basis step is verified

Inductive step:

In this step we have to prove that

$P(k) \rightarrow P(k+1) = \text{True}$

we assume that $P(k)$ is true

i.e $1+2+3+\cdots k = \frac{k(k+1)}{2}$

we have to prove that $P(k+1)$ is true

$$1+2+3+\cdots +k = \frac{k(k+1)}{2}$$

Adding $(k+1)$ on Both sides

$$1+2+3+\cdots +k+k+1 = \frac{k(k+1)}{2} + k+1$$

$$1+2+3+\cdots + k+1 = (k+1)\left[\frac{k}{2}+1\right]$$

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

$\therefore P(k+1)$ is true

By mathematical induction $P(n)$ is true for all +ve integer values

(2) If n is a positive integer prove that $1.2+2.3+3.4+\cdots +n(n+1) = \frac{n(n+1)(n+2)}{3}$

Sol Let $P(n): 1.2+2.3+3.4+\cdots +n(n+1) = \frac{n(n+1)(n+2)}{3}$

where n is a positive integer

Basis step:

we can take $n=1$

$$1.2 = \frac{1(1+1)(1+2)}{3}$$

$$2 = 2$$

LHS = RHS

Basis step is verified

Inductive step:

we have to prove that $P(k) \rightarrow P(k+1)$ is true

assume that $P(k)$ is true

i.e., $1.2+2.3+3.4+4.5+\ldots+k(k+1) = \frac{k(k+1)(k+2)}{3}$

we have to prove that $P(k+1)$ is true

$$1.2+2.3+3.4+\ldots+k(k+1) = \frac{k(k+1)(k+2)}{3}$$

Adding $(k+1)(k+2)$ on B.S

$$1.2+2.3+3.4+\ldots+k(k+1)+(k+1)(k+2) =$$

$$\frac{k(k+1)(k+2)}{3} + (k+1)(k+2)$$

$$= (k+1)(k+2)\left[\frac{k}{3}+1\right]$$

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

$\therefore P(k+1)$ is true

Hence By the mathematical induction it is true for all positive integer value of n.

(3) Prove that $n < 2^n$ where n is +ve integer by the mathematical Induction

Sol $P(n): n < 2^n$

where n is +ve integer

Basis step :-

we can take $n = 1$

$P(1) = 1 < 2$

$\therefore P(1)$ is true

Basis step is verified.

Inductive step :-

we have to prove that

$P(k) \rightarrow P(k+1) =$ True

we assume that $P(k)$ is true

$P(k) = k < 2^k$

we have to prove that $P(k+1)$ is true

$k < 2^k$

Adding '1' on Both sides

$k+1 < 2^k + 1$

$k+1 < 2^k + 2^k$ $\boxed{1 < 2^k}$ inequality does not change so we consider this

$k+1 < 2^k(1+1)$

$k+1 < 2^k \cdot 2^1$

$k+1 < 2^{k+1}$

$\therefore P(k+1)$ is true

Hence, by the mathematical induction it is true for all +ve integers

(4) Prove that $2^n < n!$ where $n \geq 4$ by the mathematical induction.

Sol Let $P(n) : 2^n < n!$

where $n \geq 4$

Basis step :-

we can take $n = 4$

$P(4) = 2^4 < 4!$

$= 16 < 24$

$\therefore P(4)$ is true

$\therefore$ Basis step is verified.

Inductive step :-

we have to prove that $P(k) \rightarrow P(k+1) =$ true

we assume that $P(k)$ is true

i.e., $P(k) = 2^k < k!$

we have to prove that $P(k+1)$ is true

$2^k < k!$

Multiply '2' on both sides

$2^k \cdot 2 < 2k!$

$2^{k+1} < 2k!$

$n! = n(n-1)$

$(k+1)! = (k+1)k!$

$2^{k+1} < (k+1)k!$

$2^{k+1} < (k+1)!$

$2 < k+1$

k value starts from 4

so,

* $2 < 4+1$

  $2 < 5$

* $2 < 5+1$

  $2 < 6$

$\therefore P(k+1)$ is true

Hence, by the mathematical induction it is true for $n \geq 4$

(5) prove that $n^3 - n$ is divisible by 3 where n is a positive integer by the mathematical Induction.

Sol

Let $P(n): n^3 - n$

where n is +ve integer

Basis step:-

we can take $n = 1$

$P(1) = 1^3 - 1$

$P(1) = 0$

P(n) is true which is divisible by 3

$\therefore$ Basis step is verified

Inductive step:-

we have to prove that $P(k) \rightarrow P(k+1) =$ True

we assume that $P(k)$ is true

i.e., $P(k) = k^3 - k$

Now, we have to prove that $P(k+1)$ is true

$P(k) = k^3 - k$

$P(k+1) = (k+1)^3 - (k+1)$

$= k^3 + 1 + 3k^2 + 3k - k - 1$

$= (k^3 - k) + 3k + 3k^2$

$= (k^3 - k) + 3(k + k^2)$

which is divisible by 3

$\therefore P(k+1)$ is true

Hence, By mathematical Induction $n^3 - n$ divisible by 3 where n is a positive integer.

(6) prove that each $n \in z^+$

$$1^2+2^2+3^2+\cdots+n^2 = \frac{n(n+1)(2n+1)}{6}$$

Let $P(n): 1^2+2^2+3^2+\cdots+n^2 = \frac{n(n+1)(2n+1)}{6}$

where $n \in z^+ = n = 1, 2, 3, \cdots$

Basis step:-

we can take $n=1$
we have to verify $P(1)$ is true

$$P(1) = 1^2 = \frac{1(1+1)(2+1)}{6}$$

$$1 = 1$$

LHS = RHS

$\therefore$ Basis step is verified

Inductive step:-

we have to prove that $P(k) \rightarrow P(k+1) =$ True

we assume that $P(k)$ is true

$$P(k) = 1^2+2^2+3^2+\cdots+k^2 = \frac{k(k+1)(2k+1)}{6}$$

$$P(k) = \frac{k(k+1)(2k+1)}{6}$$

we have to prove that $P(k+1)$ is true

i.e

$$P(k) = 1^2 + 2^2 + 3^2 + \cdots k^2 = \frac{k(k+1)(2k+1)}{6}$$

adding $(k+1)^2$ on Both sides

$$P(k+1) = 1^2 + 2^2 + 3^2 + \cdots k^2 + (k+1)^2 = \frac{k(k+1)(2k+1)}{6} + (k+1)^2$$

$$= (k+1)\left[\frac{k(2k+1)}{6} + (k+1)\right]$$

$$= (k+1)\left[\frac{2k^2+k}{6} + (k+1)\right]$$

$$= (k+1)\left[\frac{2k^2+k+6k+6}{6}\right]$$

$$= (k+1)\left[\frac{2k^2+7k+6}{6}\right]$$

$$= (k+1)\left[\frac{2k^2+3k+4k+6}{6}\right]$$

$$= (k+1)\left[\frac{k(2k+3)+2(2k+3)}{6}\right]$$

$$= (k+1)\left[\frac{(2k+3)(k+2)}{6}\right]$$

$$= \frac{(k+1)(k+2)(2k+3)}{6}$$

$\therefore P(k+1)$ is true

Hence $P(n)$ is true for all $n \in \geq$ by the mathematical Induction.

(7) Prove that $4n < n^2 - 7$ for all +ve integers $n \geq 6$

Let $P(n): 4n < n^2 - 7$

where $n \geq 6$

Basis step:

To show that $P(6)$ is true

$$P(6) = 4 \times 6 < 6^2 - 7$$

$$= 24 < 36 - 7$$

$$= 24 < 29$$

$\therefore P(6)$ is true

$\therefore$ Basis step is verified

Inductive step :-

we have to prove that $P(k) \rightarrow P(k+1) =$ Tru

we assume that $P(k)$ is true

$$P(k) = 4k < k^2 - 7 \quad \forall\, k \geq 6$$

we have to prove that $P(k+1)$ is true

i.e $P(k+1) = 4(k+1) < (k+1)^2 - 7$

$= 4k+4 < (k+1)^2 - 7$

Now $4k < k^2 - 7$

Adding 4 on B.S

$4k+4 < k^2 - 7 + 4$

$4k+4 < k^2 - 7 + 2k + 1$

$4k+4 < k^2 + 2k + 1 - 7$

$4k+4 < (k+1)^2 - 7$

$4 < 2k+1$
$4 < 2(6)+1$
$4 < 13$

In place of 4 put $2k+1$

$\therefore P(k+1)$ is true

Hence, $P(n)$ is true for $n \geq 6$ by using mathematical Induction

(8) prove that $2^n > n^2$ for all +ve integers $n > 4$

Sol Let $P(n): 2^n > n^2$

where $n > 4$

Basis step:-

To show that $P(5)$ is true

$P(5) = 2^5 > 5^2$

$= 32 > 25$

$\therefore P(5)$ is true

$\therefore$ Basis step is verified

Inductive step:-

we have to prove that $P(k) \rightarrow P(k+1)$ is true

we assume that $P(k)$ is true

$$P(k) = 2^k > k^2$$

we have to prove that $P(k+1)$ is true

i.e $P(k+1) = 2^{k+1} > (k+1)^2$

Now $2^k > k^2$

Multiply 2 on both sides

$$2^k \cdot 2^1 > k^2 \cdot 2$$

$$2^{k+1} > 2k^2$$

$$2^{k+1} > k^2 + k^2$$

$$2^{k+1} > k^2 + 4k$$

$$2^{k+1} > k^2 + 2k + 2k$$

$$2^{k+1} > k^2 + 2k + 1$$

$$2^{k+1} > (k+1)^2$$

$k^2 > 4k$

$5^2 > 4(5)$

$25 > 20$

$2k > 1$

$2(5) > 1$

$10 > 1$

$\therefore P(k+1)$ is true

Hence $P(n)$ is true for $n > 4$ by the mathematical induction

(9) Prove by mathematical induction for any +ve integer n.
$11^{n+2} + 12^{2n+1}$ is divisible by 133

Sol Let $p(n)$: $11^{n+2} + 12^{2n+1}$ is divisible by 133 where n is +ve integer

Basis step :-

we can take $n = 1$

$$P(1) = 11^3 + 12^3$$
$$= 1331 + 1728$$
$$= 3059$$

$3059 = 23 \times 133$

which is divisible by 133

$\therefore P(n)$ is true

Basis step is verified

Inductive step :-

we have to prove that $P(k) \rightarrow P(k+1) = T$

we assume that $P(k)$ is true

$P(k) = 11^{k+2} + 12^{2k+1}$ which is divisible by 13

we have to prove that $P(k+1)$ is true

i.e $P(k+1) = 11^{k+3} + 12^{2k+3}$

Now $11^{k+2} + 12^{2k+1}$

$$= 11^{k+3} + 12^{k+3}$$

$$= 11^{k+2} \cdot 11^{1} + 12^{2k+1} \cdot 12^{2}$$

$$= 11^{k+2} \cdot 11 + 12^{2k+1} \cdot 144$$

$$= 11^{k+2} \cdot 11 + 12^{2k+1} (11+133)$$

$$= 11^{k+2} \cdot 11 + 12^{2k+1} \cdot 11 + 12^{2k+1} \cdot 133$$

$$= 11\left[11^{k+2} + 12^{2k+1}\right] + 12^{2k+1} \cdot 133$$

↓ which is divisible 133

↓ which is divisible by 133

$\therefore$ P(k+1) is Divisible by 133

Hence by Mathematical Induction it is true for all +ve integers of n.

Strong Induction [Complete Induction]

To prove that $P(n)$ is true for all +ve integer values of n, where $P(n)$ is a proportional statement we have two steps

Basis step :-
we show that $P(1)$ is true

Inductive step :-
we show that the Conditional statement

$P(1) \wedge P(2) \wedge P(3) \wedge \dots \wedge P(k) \rightarrow P(k+1)$

is true for all +ve integers of k

① Show that Every +ve integer > 1 Can be a product of primes.

Sol Let $P(n)$: every +ve integer $n > 1$ is a product of primes.

i.e., $n = 2, 3 \dots$

Basis step :- Here $n = 2$
we have to show that $P(2)$ is a product of primes

$\because$ 2 is a prime number nothing to prove

$P(2)$ is a product of prime number.

Inductive step :-
we assume that $P(2) \wedge P(3) \wedge \dots P(k)$

all are true

$\because$ 2, 3, 5 --- k all are primes

we have to show that $P(k+1)$ is a product of primes

Case(i)

If $k+1$ = prime number

So, $P(k+1)$ is also a prime

Case(ii)

If $k+1$ = Composite number

$k+1 = ab$ $\quad 2 \le a \le k$, $\quad 2 \le b \le k$

$\therefore$ a, b are primes

ab is also a product of primes

$6 = 2 \times 3$, $14 = 2 \times 7$ } Product of Primes

$\therefore$ $P(k+1)$ is a product of primes

Hence $P(n)$ is a true for all +ve integers of $n > 1$.

② Let a_n be the sequence defined by $a_1 = 1$, $a_2 = 8$ and $a_n = a_{n-1} + 2a_{n-2}$ prove that $a_n = 3\,2^{n-1} + 2(-1)^n \; \forall\, x \in N$

Sol Let $P(n)$: $3 \cdot 2^{n-1} + 2(-1)^n \; \forall\, x \in N$

$n = 1, 2, 3 ---$

Basis step:

Take $n = 1$

$a_1 = 3 \cdot 2^{1-1} + 2(-1)^1$

$= 3 - 2$

$= 1$

Take $n = 2$

$a_2 = 3 \cdot 2^{2-1} + 2(-1)^2$

$= 3 \cdot 2 + 2$

$= 6 + 2$

$a_2 = 8$

Inductive step :-

we assume that 1, 2, ----- k are true

i.e., $a_k = 3 \cdot 2^{k-1} + 2(-1)^k$

we have to show that P(k+1) is true

i.e., $a_{k+1} = 3 \cdot 2^{k+1-1} + 2(-1)^{k+1}$

$a_{k+1} = 3 \cdot 2^k + 2(-1)^{k+1}$

we have $a_n = a_{n-1} + 2a_{n-2}$

$n = k+1$

$a_{k+1} = a_{k+1-1} + 2a_{k+1-2}$

$a_{k+1} = a_k + 2a_{k-1}$

$a_{k+1} = 3 \cdot 2^{k-1} + 2(-1)^k + 2\left[3 \cdot 2^{k-2} + 2(-1)^{k-1}\right]$

$$a_{k+1} = 3 \cdot 2^{k-1} + 2(-1)^k + 2 \cdot 3 \cdot 2^{k-2} + 4(-1)^{k-1}$$

$$= \underline{3 \cdot 2^{k-1}} + 2(-1)^k + \underline{3 \cdot 2^{k-1}} + 4(-1)^{k-1}$$

$$= 6 \cdot 2^{k-1} + 2(-1)^k + 4(-1)^{k-1}$$

$$= 6 \cdot 2^{k-1} + 2\left[(-1)^k + 2(-1)^{k-1}\right]$$

$$= 6 \cdot 2^{k-1} + 2\left[(-1)^k + 2\underline{(-1)^k}\right]$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\quad -1$$

$$= 6 \cdot 2^{k-1} + 2(-1)^k [1-2]$$

$$= 6 \cdot 2^{k-1} + 2(-1)^k (-1)$$

$$= 3 \cdot 2^1 \, 2^{k-1} + 2(-1)^{k+1}$$

$$= 3\,2^k + 2(-1)^{k+1}$$

③ f_0, f_1, f_2 and $f_0 = 0$, $f_1 = 1$ than prove that $f_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$

sol :- Given

$$P(n): \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

$f_0 = 0$

$f_1 = 1$

Basis step :-

take $n = 0$

$$f_0 = \frac{1}{\sqrt{5}}[1-1] = 0$$

$n = 1$

$$f_1 = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right) - \left(\frac{1-\sqrt{5}}{2}\right)\right]$$

$$= \frac{1}{\sqrt{5}}\left[\frac{1+\sqrt{5}}{2} - \frac{1}{2} + \frac{\sqrt{5}}{2}\right]$$

$$= \frac{1}{\sqrt{5}} \frac{2\sqrt{5}}{2}$$

$$= 1$$

∴ Basis step is verified.

Inductive step :-

$P(0), P(1) \ldots P(k)$ true

$$P(k) = f_k = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^k - \left(\frac{1-\sqrt{5}}{2}\right)^k\right]$$

we have to prove that $P(k+1)$ is true

$$F_k = F_{k-1} + F_{k-2}$$

$$F_{k+1} = F_k + F_{k-1}$$

$$F_{k+1} = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^k + \left(\frac{1-\sqrt{5}}{2}\right)^k\right]$$

$$+ \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{k-1} - \left(\frac{1-\sqrt{5}}{2}\right)^{k-1}\right]$$

$$= \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{k} + \left(\frac{1+\sqrt{5}}{2}\right)^{k-1} - \left[\left(\frac{1-\sqrt{5}}{2}\right)^{k} + \left(\frac{1-\sqrt{5}}{2}\right)^{k-1}\right]\right.$$

$$\Rightarrow \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{k}\left[1+\left(\frac{1+\sqrt{5}}{2}\right)\right] - \left(\frac{1-\sqrt{5}}{2}\right)^{k-1}\left[1+\left(\frac{1-\sqrt{5}}{2}\right)\right]\right]$$

$$\Rightarrow \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{k-1}\left(\frac{3+\sqrt{5}}{2}\right) - \left(\frac{1-\sqrt{5}}{2}\right)^{k-1}\left[\frac{3-\sqrt{5}}{2}\right]\right]$$

$$\Rightarrow \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{k-1}\left(\frac{6+2\sqrt{5}}{4}\right) - \left(\frac{1-\sqrt{5}}{2}\right)^{k-1}\left[\frac{6-2\sqrt{5}}{4}\right]\right]$$

$$\Rightarrow \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{k-1}\left(\frac{1+\sqrt{5}}{2}\right)^{2} - \left(\frac{1-\sqrt{5}}{2}\right)^{k-1}\left(\frac{1-\sqrt{5}}{2}\right)^{2}\right]$$

$$\Rightarrow \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^{k+1} - \left(\frac{1-\sqrt{5}}{2}\right)^{k+1}\right]$$

$\therefore P(k+1)$ is true Inductive step is verified

$P(n)$ is true for all +ve value

Recursively Defined function

Suppose f is RDT with domain of non -ve integers

f is defined by using 2 steps

(i) Basis step (ii) Recursive step

Basis step :-

specify the initial value of f with domain zero.

Recursive step :-

Give Rule for specifying current term using previous terms.

① Suppose f is defined by $f(0)=3$

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

find $f(1), f(2), f(3)$

Basis step :- $n=0$

$$f(0) = 3$$

Recursive step :-

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

$n=0$

$$f(1) = 2f(0)+3$$

$$= 2(3)+3 = 6+3 = 9$$

$n=1$

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

$$= 2(9)+3 = 21$$

$f(3) = 2f(2) + 3 = 45$

(2) Given an Recursively defined function of the factorial function

$F(n) = n!$

Sol :- Given factor function

$F(n) = n!$

we can R.D.F by using 2 steps

(1) Basis step :-

we will find out initial value of factorial function of domain zero

$F(n) = n!$

$F(0) = 0! = 1$

Here basis step defined initial value of RDF of factorial function.

(2) Recursive step :-

we will find out Rule to specify factorial function in terms of its previous term

$F(n) = n!$

$F(n+1) = (n+1)!$

$= (n+1)n!$

$$\boxed{F(n+1) = (n+1)F(n)}$$

it is R.D.F

③ Give a recursive definition of a^n where a is non zero real number and n is non -ve integer.

Sol :- Given

$$F(n) = a^n \quad n = 0, 1, 2 \ldots$$

we can R.D.F by using 2 steps

Basis step :-

Take $n = 0$

$$F(0) = a^0$$
$$= 1$$

Recursive step :-

we have $F(n) = a^n \longrightarrow (1)$

Take $n = n+1$

$$F(n+1) = a^{n+1}$$
$$= a^n \cdot a$$
$$= F(n) \cdot a$$
$$F(n+1) = a\,F(n)$$

which is R.D.F

④ Give a recursive defined function $\sum_{k=0}^{n} a_k$

Sol :- Let $f(n) = \sum_{k=0}^{n} a_k$

we can write R.D.F by using two steps.

Basis step :-

Take $k=0$

$F(0) = a_0$

Recursive step :-

we have

$$F(n) = \sum_{k=0}^{n} a_k \longrightarrow (1)$$

take $n = n+1$

$$F(n+1) = \sum_{k=0}^{n+1} a_k$$

$$= \sum_{k=0}^{n} a_k + a_{n+1}$$

From (1)

$$F(n+1) = F(n) + a_{n+1}$$

Recursively Defined function (R.D.F)

Suppose f is RDF with domain set of non-ve integer

f is Defined by using 2 steps

(i) Basis step (ii) Recursive step

Basis step: Specify the intial value of f with Domain Zero

Recursive step: Give Rule for Specifying current term using previous terms

(1) Ex:

Suppose f is Defined by

$f(0) = 3$

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

find $f(1), f(2), f(3)$

Basis step

$f(0) = 3$

Recursive step

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

$n = 0$

$f(1) = 2f(0) + 3$

$= 2(3) + 3$

$= 6 + 3 = 9$

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

$= 2 \times 9 + 3 = 21$

$f(3) = 2f(2) + 3 = 45$

11. Give an Recursively defined function of the factorial function

$$F(n) = n!$$

Sol. Given factorial function

$$F(n) = n!$$

We can R.D.f by using 2 steps

(1) Basis Step: We will find out initial value of factorial function at Domain zero

$$F(n) = n!$$
$$F(0) = 0!$$
$$= 1$$

Here Basis step Defined initial Value of RDF of factorial Function

(2) Recursive Step: We will find out Rule to Specify factorial function in terms of its previous term

$$F(n) = n!$$
$$F(n+1) = (n+1)!$$
$$= (n+1)\,n!$$

$$\boxed{F(n+1) = (n+1)\,F(n)}$$

this R.D.F

(11) Give a recursive Definition of a^n where a is non zero Real number and n is non-ve integer

Sol Given $F(n) = a^n$ $n = 0, 1, 2 \cdots$

We can R.D.F by using 2 steps

Basis step:

take $n = 0$

$F(0) = a^0$

$F(0) = 1$

Recursive step:

We have $F(n) = a^n$ —(1)

take $n = n+1$

$F(n+1) = a^{n+1}$

$= a^n \cdot a$

$= F(n) \cdot a$

$F(n+1) = a\,F(n)$

which is R.D.F

(iv) Give a Recursive function Defined

$$\sum_{k=0}^{n} a_k$$

Sol Let $F(n) = \sum_{k=0}^{n} a_k$.

We can write R.D.F by using two step

(1) Basis step

take $k=0$

$$F(0) = a_0$$

(2) Recursive step:

We have

$$F(n) = \sum_{k=0}^{n} a_k \quad \text{— (1)}$$

take $n = n+1$

$$F(n+1) = \sum_{k=0}^{n+1} a_k$$

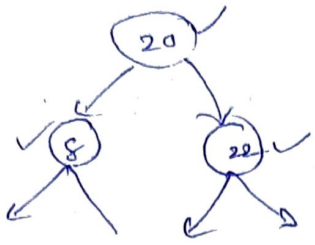
$$= \sum_{k=0}^{n} a_k + a_{n+1}$$

from (1)

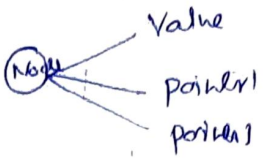
$$F(n+1) = F(n) + a_{n+1}$$

Structural Induction

In computer Science there are Recursive Defined Structure (RDS) Trees - R.D.S



Every Node is associated with pointer 1 and pointer 2



If part of structure exhibit Same properties and the total structure has got Same property We can Say that R.D.S there are Some other R DS in compater Science

RDS — Strings
→ Well - Formed formulae

The properties of these RDS (Strings, Trees, & W.F.F) are proved by using the proof of technique Called Structual Induction by verifying two steps

① Basis step
② Inductive step (or) Recursive step

Let us understand how Structural Induction is used to prove Properties of RDS By Basis Step & Recursive Step

String (RDS

Suppose $P(w)$: propositional function over Set of Strings Σ^*

String $w \in \Sigma^*$

$\exists n$ S.T

$P(w) :=$ true $\forall$ strings $w \in \Sigma^*$

Basis step — To show $P(\lambda) =$ True where λ is empty String

Recursive step:

Assume $P(w) = T$ where $w \in \Sigma^*$

I Want to Show $P[wx] = T$ if $x \in \Sigma$ (Symbol / alphabet) string $w \in \Sigma^*$

Tree (R.D.S)

Basis step — To Show property of Tree is True when Tree consist of Single vertex

Basis step verified

Recursive Step

Assuming the property of Trees T_1 and T_2 are True

I Want to Show the property of Tree $T = T_1 * T_2$ consists of a Root r which has T_1 has its left tree T_2 has its right tree

Recursive Definition of the length of the String

(e) $L(w)$ denote — Recursive Definition
- That String — Represents the length of String

Now $L(w)$ can be defined by

(i) $L(\lambda) = 0$ — λ is empty String

(ii) $L[wx] = l(w) + 1$

$w \rightarrow$ String

$w \in \Sigma^*$ → set of Strings

x – letter (one Symbol)

$x \in \Sigma$ (alaphabet)

Use Structural Induction to prove that

$l[x,y] = l(x) + l(y)$ where x and y belong to Σ^*, the set of strings over the alphabet Σ.

Sol

Let $p(z)$ denote: $L(x,y) = l(x) + l(y)$

to prove $p[\epsilon] = T$, we use two steps of Structural Induction

Basis Step: To verify this step we must show $p(\lambda) = T$

i.e it is to verify

$$L[x\lambda] = l(x) + l(\lambda) \quad \forall x \in \Sigma^*$$

$$l[x\lambda] = l(x)$$

$$= l(x) + 0$$

λ is empty string

$$l[x\lambda] = l(x) + l(\lambda)$$

for all $x \in \Sigma^*$

Recursive step

To verify this step

We show that

$P[za] = \text{true}$

By assuming $P[z] = \text{true}$

$l[xy] = l(x) + l(y) \quad x, y \in S^+$ — (1)

To prove $P[za] = \text{true}$

i.e $l[xya] = l[x] + l[ya]$ ✓

take

$L[xya] = l[xya] + 1$ — (2)

Since R.P.F of $l[wx] = l(w) + 1$

Sing Symb

$L[ya] = L[y] + 1$ — (3)

put (1) in (2)

$L[xya] = l(x) + l(y) + 1$

$L[xya] = l(x) + l[ya]$

$\therefore \quad P[za] = T$

$\therefore$ By Induction Step gets verified

Recursive Def of Full Binary Tree:

n(T) & n(T)

Consider height of FBT as h(T) (Denoted)

h(T) is Defined Recursively using two steps

Basis Step: $h(T) = 0$

T → has only one Node (Root)

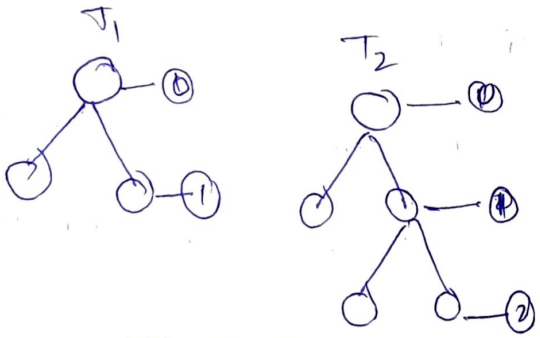
Recursive step:

Let T_1 is FBT

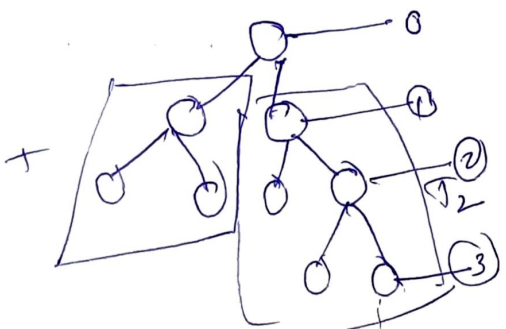
T_2 is FBT

$T = T_1 \cdot T_2$

$$h(T) = 1 + \text{Max}\left(h(T_1), h(T_2)\right)$$



$T = T_1 \times T_2$



$1 + \max[1, 2]$

$1 + 2$

$= 3$

Recursive Def of Full Binary tree of no of vertex

$n(T)$

$n(T)$ is Defined Recursively using two steps

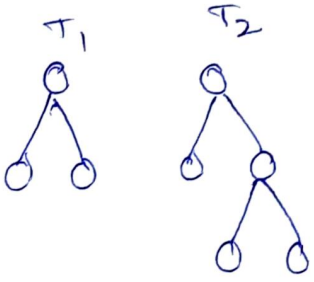
Basis Step: $n(T) = 1$

T has only one vertex (node)

Recursive Step:

T_1 – FBT

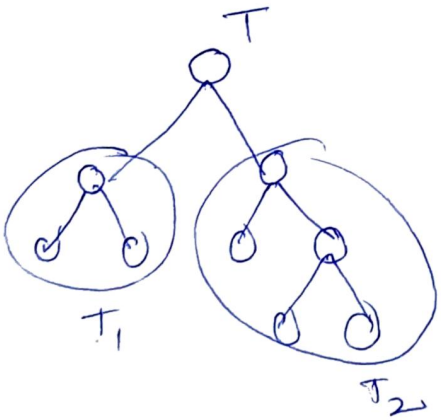
T_2 – FBT



$T = T_1 \times T_2$

$n(T) = n(T_1) + n(T_2)$

$n(T) = 1 + n(T_1) + n(T_2)$



If T is a full binary Tree then

$$n(T) \leq 2^{h(T)+1} - 1$$

To prove this property of FBT we will use Structural Induction

We use two steps,

Basis Step: For FBT consisting only root the result

$$n(T) \leq 2^{h(T)+1} - 1 \text{ is true}$$

i.e $n(T) = 1$

$$h(T) = 0$$

$$1 \leq 2^{0+1} - 1$$

$$1 \leq 2 - 1$$

$$1 \leq 1$$

Inductive Step

Let T_1 & T_2 are FBT

At T_1 & T_2 are FBT

We assume

$$n(T_1) \leq 2^{h(T_1)+1} - 1$$

$$n(T_2) \leq 2^{h(T_2)+1} - 1$$

By Recursive formula of FBT property

$$n(T) = 1 + n(T_1) + n(T_2)$$

$$= 1 + 2^{h(T_1)+1} - 1 + 2^{h(T_2)+1} - 1$$

$$= 2^{h(T_1)+1} + 2^{h(T_2)+1} - 1$$

$$= 2^{h(T_1)+h(T_2)+2}$$

$4 + 8 \leq 2 \max(4, 8)$

$12 \leq 2(8)$

$12 \leq 16$

$n(T) \leq 2 \cdot \left(2^{h(T_1)+1} \cdot 2^{h(T_2)+1}\right)^{\max}$

$$n(T) \leq 2 \cdot \max\left[2^{h(T_1)+1}, 2^{h(T_2)+1}\right] - 1$$

$\max[2^x, 2^y]$

$\max(x, y)$

2

$$n(T) \leq 2 \cdot 2^{\max[h(T_1), h(T_2)]+1} - 1$$

$$n(T) \leq 2 \cdot 2^{h(T)} - 1$$

$$n(T) \leq 2^{h(T)+1} - 1$$ ✓

Unit-3

The Fundamentals: Algorithms, the Integers, and Matrices

Part-I

Algorithms:-

An algorithm is a finite set of precise instructions for performing a computation or for solving a problem.

Properties:

There are several properties that algorithms generally share. These properties are:

Input. An algorithm has input values from a specified set.

Output. From each set of input values an algorithm produces output values from a specified set.

Definiteness. The steps of an algorithm must be defined precisely.

Correctness. An algorithm should produce the correct output values for each set of input values.

Finiteness An algorithm should produce the desired output after a finite number of steps for any input in the set

Effectiveness. It must be possible to perform each step of an algorithm exactly and in a finite amount of time.

Generality. The procedure should be applicable for all problems of the desired form, not just for a particular set of input values.

→ An algorithm can also be described using a computer language. However, when that is done, only those instructions permitted in the language can be used.

→ This often leads to a description of the algorithm that is complicated and difficult to understand.

Furthermore, because many programming languages are in common use, it would be undesirable to choose one particular language.

So instead of using a particular computer language to specify algorithms, a form of pseudocode

→ Pseudocode provides an intermediate step between an English language description of an algorithm and an implementation of this algorithm in a programming language.

The steps of the algorithm are specified using instructions resembling those used in programming languages.

However, in pseudocode, the instructions used can include any well-defined operations or statements.

example:-

Describe an algorithm for finding the maximum value in a finite sequence of integers.

```
procedure max (a1, a2, ..., an : integers)
  max := a1
  for i = 2 to n
      if max < ai then max := ai
  {max is the largest element}.
```

Searching Algorithms:-

The problem of locating an element in an ordered list occurs in many contexts. For instance, a program that checks the spelling of words searches for them in a dictionary, which is just an ordered list of words. Problems of this kind are called searching problems.

The linear search:-

The first algorithm that we will present is called linear search or sequential search algorithm. The linear search algorithm begins by comparing x and a_1. When $x = a_1$, the solution is the location of a_1, namely 1. When $x \neq a_1$ compare x with a_2. If $x = a_2$, the solution is the location of a_2, namely 2. When $x \neq a_2$, compare x with continue this process, comparing x successively with each term of the list.

When $x \neq a_2$ compare x with a_3. continue this process comparing x successively with each term of the list until a match a found, where the solution is the location of that term unless no match occurs. If the entire list has been searches without locating x, the solution is 0. The Pseudocode for the linear search is displayed as Algorithm 2.

Pseudocode:

```
procedure linear search (x: integer, a1, a2 ... an;
                                    distinct integers)
    i := 1
    while (i ≤ n and x ≠ ai)
        i := i + 1
    if i ≤ n then location := i
    else location := 0
{ location is the subscript of the term
  that equals x, or is 0 if x is not found }.
```

The Binary Search :-

We will now consider another searching algorithm can be used when the list has terms occuring in order of increasing size (for instance. if the terms are numbers, they are listed from smallest to largest, if they are words they are listed in lexicographic or alphabetic. This second searching algorithm is called binary search algorithm. It proceeds by comparing the element to be located to the middle term of the list. The list is then split into two smaller sublists of the same size, or where one of these smaller list has one fewer term than the other. The search located and the middle terms.

Algorithm 3: The Binary search.

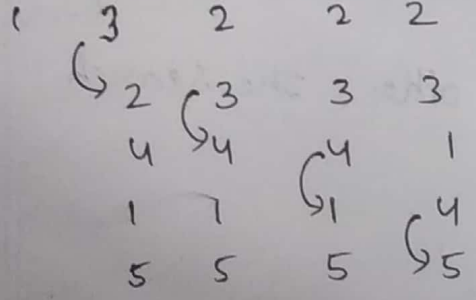
```
binary search (x: integer, a1, a2, .... an; increasing integers)
i := 1   {i is left endpoint of search interval)
j := n   {j is right endpoint of search interval)
while i < j
begin
  m := [(i+j)/2]
  if x > am then i := m+1
     else j := m
end
if x = ai then location := i

else location := 0
{location is the subscript of the term equal to x,
                    or 0 if x is not found}.
```

Bubble sort:-

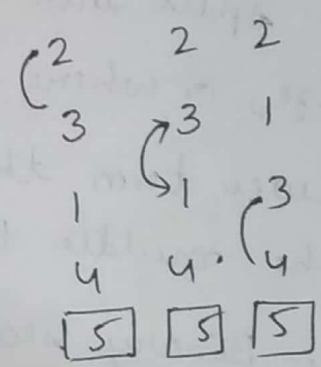
The Bubble sort is one of the simplest sorting algorithms but not one of the most efficient it puts a list into increasing order by successively comparing adjacent elements, interchanging them if they are in the wrong order. To carry out the bubble sort, we perform the basic operation that is the list for a full pass.

We iterate this prodedure until the sort is complete.

first pass



second



Algorithm:

The Bubble sort

procedure bubble sort ($a_1, \ldots a_n$; real numbers with $n \geq 2$)

for i := 1 to n-1
  for j := 1 to n-1
    if $a_j > a_{j+1}$ then interchange a_j and a_{j+1}

($a_j \ldots a_n$ is in increasing order).

The insertion Sort:

The insertion sort is a simple sorting algorithm. but it is usually not the most efficient to sort a list with n elements, the insertion sort begins with the second element.

The insertion sort compares this second element with the first element and after the first element if it exceeds the first element. At this point the first two elements are in the correct order.

The third element is then compared with the first element and if it is larger than the first element it is compared with second, it is inserted into the correct position among the first three elements.

Greedy Algorithm:

Many algorithms we will study in this book are designed to solve optimization problem. The goal of such problems is to find a solution to the given problem that either minimizes or maximizes the value of some parameters.

Suprisingly one of the simplest approaches often lead to a solution of an optimization problem. This approach select the best choice at each step instead of considering all sequences of steps may lead to an

optimal solution. Algorithms that make what seems to be the "best choice at each step are called greedy algorithms. once we know that a greedy algorithm finds a feasible solution we need to determine whether it has found an optimal solution.

```
Procedure insertion sort (a1, a2, a3 ... an; real nu
                          with n≥2)

for j := 2 to n
begin
    i := 1
while aj > ai
    i := i + 1
(m := aj
for k = 0 to j - i - 1
    aj-k = aj-k-1

ai := m
end {a1, a2 ... an are sorted}.
```

Algorithm for Greedy Change-Making Algorithm

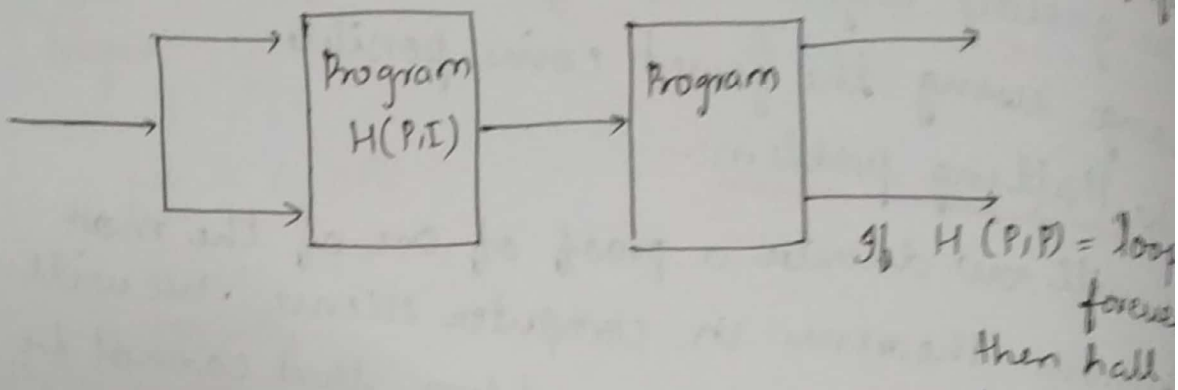
```
Procedure change (c1, c2, ... cr values of
denominations of coins, where
                c1 > c2 > c3 ... > cn, n: a positive inte

for i := 1 to r
while n ≥ ci
    begin
    add a coin with value ci to the chang
    n := n - ci
end.
```

Lemma: If n is a positive integer, then n cents in change using quarters, dimes, nickels and pennies using the fewest coins possible has at most two dimes, at most one nickel, at most four pennies and cannot have two dimes and a nickel. The amount of change in dimes, nickels and pennies cannot exceed 24 cents.

Theorem 1:
The greedy algorithm (Algorithm 6) produces change using the fewest coins possible.

The Halting problem:-
We will now describe a proof of one of the most famous theorems in computer science. We will show that there is a problem that cannot be solved using any procedure. That is, we will show there are unsolvable problems. The problem we will study is the halting problem.
Certainly being able to test whether a program entered into an infinite loop would be helpful when writing programs. However in 1936 Alan Turing showed that no such procedure it would be convenient to have such as procedure, if it existed. Certainly being exists.

To show that no procedure H exists that solves the halting problem, we construct a simple procedure K(P), which works as follows making use of the output H(P,P). If the output of H(P,P) is "loops forever" which means that P halts when given a copy of itself as input then K(P) loops forever. That is K(P) does the opposite of what the output of H(P,P) specifies.



Now suppose we provide K as input to K. We note that if the output of H(K,K) is "loops forever" then by the definition of K we see that K(K) halts. Otherwise if the output of (H(K,K) is halt then by the definition of K we see that K(K) loops forever. In violation of what H tells us. In both cases we have a contradiction. Thus, H cannot always give the correct answers. Consequently, there is no procedure that solves the halting problem.

Complexities of An algorithm

Time complexity!

The time complexity of an algorithm is the amount of computing time required by an algorithm to complete the Execution

The time complexities of an algorithm is the sum of two components

1. compile time
2. Run time

compile time does not depends on Instance charcterstics

→ once the the program is compiled that can be Run several time without Re compilisation

→ Run time Depends on the particular problem Instance

$$T(P) = C(P) + R(P)$$

where T(P) is the time complexities of an algorithm P

C(P) is the compile time of algorithm P

R(P) " Run time of algorithm

The time complexity of an algorithm is calculated by frequency count Method

General Rules (0) Norms for calculating time complexity

Rule 1 Loops

The Running time of for loop is the Running time of statements in the for loop

Ex:

```
for (i=0, i<n, i++)  ---> n+1
    s = s+i;         ---> n
```

So time complexity.

---> O(n)

2n + 1
O(n)

Let n = 3

0 < 3
1 < 3
2 < 3
3 < 3

Rule: 2 Nested loop:

ie writing one loop inside another loop

The TOTal Running time of statements inside a group of nested loop's the product of size all loops

Ex:

```
for (i=0; i<n; i++)          ---> n+1
{
    for (j=0; j<n; j++)      ---> n+1 * n
    {
        c[i,j] = a[i][j] + b[i][j]   ---> n*n
    }
}
```

$2n^2 + 2n + 1$

ignore constants, lower Exponential of n

The time complexity is $O(n^2)$

Rule 3: Consecutive Statements

Here Running time is the Maximum one

Ex:

```
for (i=0; i<n; i++)          ---> n+1

    s = s+i;                 ---> n
for (i=0; i<n; i++)          ---> n+1
for (j=0; j<n; j++)          ---> (n+1)*n
{
    c[i,j] = a[i][j] + b[i][j]   ---> n*n
}
```

$2n^2 + 4n + 2$

The time complexity is $O(n^2)$

Rule 4: if-else

```
if (cond)
    S1
else
    S2
```

Here Running time is Max of S_1 and S_2

Ex:

```
if (n<0)        ---> 1
```

Step count Method (Time complexity)

① s/e : Steps per Execution

it Defines whether statement is a Valid Instruction or not

if s/e = 0 (—) if it not Valid Instruction

= 1 if it is a Valid Instruction

② frequency: How many times an Instruction is Executed

∴ Total no of Steps = s/e * frequency

Example

Sum of 'n' no of array elements

Statement no	Statement	S/e	frequency	Total steps (S/e * freq)
1	Algorithm Sum(a,n)	—	—	—
2	{	—	—	—
3	s = 0	1	1	1
4	for i = 1 to n do	1	n+1	1 * (n+1) = n+1
5	{	—	—	—
6	s = s + i	1	n	1 * n = n
7	}	—	—	—
8	Return s;	1	1	1
9	}	—	—	—

Total no of steps = (n+1) + n + 1 + 1

= 2n + 3

∴ The time complexity is,

O(n)

Ex: 2

Matrix Addition

Statement no	Statement	s/e	frequency	Total step
	Algorithm MatAdd (a,b,c,m,n)	—	—	—
	{	—	—	—
	for i:= 1 to m do	1	(m+1)	(m+1)
	for J:= 1 to n do	1	m*(n+1) = mn+m	mn+m
	c[i,j] = a[i][j] + b[i][j]	1	m*(n)	mn
	}	—	—	—
	Total no step			2mn + 2m + 1

the time complexity = O(mn)

= O(n²) (for m=n)

Matrix Multiplication

Algorithm Mat Mul (A, B, C)

```
{
  for i = 1 to n do                    (n+1) times
  {
    for j = 1 to n do                  ---> (n+1) n
    {
      c[i,j] = 0                       n^2
      for k = 1 to n do                ---- n(n+1)
      {
        c[i,j] = A[i][k] + B[k][j]     -> n^3 times
      }
    }
  }
```

$\therefore \quad f(n) = n+1 + n^2 + n + n + n^3 + n^2 + n^2 + n^3$

$= \quad 2n^3 + 3n^2 + 2n + 1$

$O(n^3)$

Binary Search

```
Algorithm BinSearch(A, x, n)
{
    i = 1                                   1 time
    j = n                                   1 time
    while (i <= j)                          log n
    {
        mid = (i+j)/2                       log n
        if [x < A(mid)]                     log n
            j = mid-1;                      log n (or)
        else if
            [x > A([mid])]                  log n (or)
            i = mid+1;                      log n time
        else
            Return mid.                     log n
    }
}
```

one of these Executes, not all three Hence log n time

3 log n + 2

The time Complexity

$f(n) = 3\log n + 2$

Iteration

Length of array = n

At iteration 1

length of Array = $n/2$

At iteration 3

length of Array = $\left(\frac{n/2}{2}\right) = \frac{n}{2\times 2} = \frac{n}{4} = \frac{n}{2^2}$

$\therefore$ after iteration $\frac{n}{2^K}$

After K iteration length of array become 1

$\log_b^a = a\log b$

$\log_2 n = K\log_2 2$ $\quad K = \log_2 n$ $\quad \frac{n}{2^K} = 1$ $\quad 2^K = n$ $\quad \log_2 n = \log_2 2^K$

Binary Search

```
Algorithm Bin Search (A, x, n)
{
    i = 1                                   1 time
    j = n                                   1 time
    while (i <= j)                          log n
    {
        mid = (i+j)/2                       log n
        if [x < A(mid)]                     log n
            j = mid - 1;                    (or) log n
        else if
            [x > A([mid)]                   log n (or) log n times
            i = mid + 1;
        else
            Return mid.                     log n
    }
}
```

One of three Executes, not all three Hence log n times

3 log n + 2

the time Complexity

$f(n) = 3\log n + 2$

Iteration

length of array = n

At iteration 2

length of Array = $n/2$

At iteration 3

length of Array = $\left(\frac{n}{2}\right)/2 = \frac{n}{2\times 2} = \frac{n}{4} = \frac{n}{2^2}$

$\therefore$ after iteration $\frac{n}{2^K}$

After K iteration length of array become 1

$\log_b^a = a\log b$

$\frac{n}{2^K} = 1 \quad 2^K = n \quad \log_2 n = \log_2 2^K$

$\log_2 n = K\log_2 2 \quad K = \log_2 n$

Space complexity

An algorithm can be evaluated based on their performance

→ performance Evaluation can be done in two different ways

(i) Before execute evaluation termed as priori estimate

(ii) After Executing Evaluation termed as posteriori testing

→ priori Estimate is also known as performance analysis

→ posteriori Estimates is known as performance Management

performance analysis may Deals with Time and Space complexities

Space complexity

It is the amount of Memory (Space) Required by the algorithm to run to completion

Space needed is Sum of two components

Fixed part	Variable part
→ Independent of Input and output charcteristics	→ Depend on particular problem Instance
→ Typically Include	→ Typically Include
* Instruction Space	* Space needed by Reference of Variables
* Space for Variables	* Recursion Stack Space
* Space for constants	

let P be the algorithm

let S(P) be the Space complexity for algorithm

$$S(P) = C + S_p$$

where C = fixed part

S_p = Variable Space

Example :

```
Algorithm Sum (a, n)
{
    s = 0;
    for i = 1 to n
        s = s + a[i];
    Return s;
}
```

Space needed for this algorithm

Size Variable n → 1 word
array a' values → n words
loop variable → 1 word
Sum variable s → 1 word
Total size → (n+3)

Here the Space Required to Calculate Sum of 'n' number in array

$Sp(n) >= n+3$

Example : 2

```
Algorithm Rsum (a, n)
{
    If (n <= 0) then Return 0;
    else
        Return Rsum (a, n-1) + a[n]
}
```

Space needed for this algorithm

Each call of Rsum Requires

Return address : 1 word
pointer to a : 1 word
Local Variable n : 1 word
TOTAL Size 3 words

Depth of Recursion : n+1 ∴ $Sp(n) >= 3(n+1)$

Example: 1

```
Algorithm Sum (a, n)
{
    s = 0;
    for i = 1 to n
        s = s + [i];
    Return s;
}
```

Space needed for this algorithm

Size Variable n – 1 word
array a' value – n words
loop variable – 1 word
Sum variable s → 1 word
TOTAL size → (n+3)

Here the Space Required to calculate Sum of 'n' number in array

$Sp(n) >= n+3$

Example: 2

```
Algorithm Rsum (a, n)
{
    If (n <= 0) then Return 0;
    else
        Return Rsum (a, n-1) + a[n]
}
```

Space needed for this algorithm

Each call of Rsum Requires

Return address : 1 word
pointer to a : 1 word
local variable n : 1 word
TOTAL size 3 words
Depth of Recursion: n+1 ∴ $Sp(n) >= 3(n+1)$

Asymptotic Notations

An asymptote is value that you get closer and closer to but never quite reach

1. Big oh (O) notation
2. omega (Ω) notation
3. Theta (Θ) Notation
4. Little oh Notation (o)
5. Little omega Notation

1. Big oh Notation:-

→ it represents the symbol 'O'.

→ it is used to Represent upper bound of algorithms Running time

→ it gives the longest Running time to complete the algorithm

The function $f(n) = O[g(n)]$ iff there exists some +ve constant such as c, n_0 such that

$f(n) \leq c \cdot g(n)$ $\forall n, n \geq n_0$

Ex:

$f(n) = 3n+2$

$g(n) = n$

$c = 4$

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

$3n+2 \leq 4n$

2. Omega Notation:

it is represents the Symbol (Ω)

it is used to Represents lower bounds of algorithm Running time

it gives the shortest Running time to complete the algorithm

Def: The function $f(n) = \Omega[g(n)]$ iff there exists some +ve constant, c and n_0 such that

$f(n) \geq c\, g(n) \quad \forall n \quad n \geq n_0$

For example.

$f(n) = 3n+2$

$g(n) = n$

$f(n) \geq c \cdot g(n)$

$3n+2 \geq 3 \cdot n$ where $c=3$

Suppose $n=1$

$5 \geq 3$ true

$n=2$

$8 \geq 5$ True

$f(n) \geq c \cdot g(n) \quad \forall n \quad n \geq 1$

i.e $3n+2 \geq 3n \quad \forall n \quad n \geq 1$

Suppose $n=1$

$3(1)+2 \leq 4(1)$

$5 \leq 4$ — false

$n=2$

$3(2)+2 \leq 4(2)$

$8 \leq 8$ — True

$n=3$

$3(3)+2 \leq 4(3)$

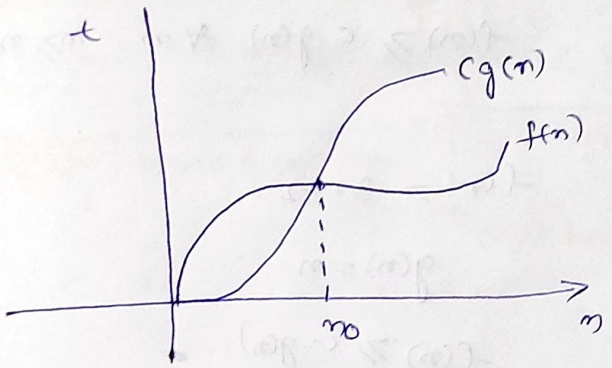
$11 \leq 12$ — True

$\therefore 3n+2 \leq 4n \quad \forall n$

$n \geq 2$

$\therefore f(n) = O[n]$ if $3n+2 \leq 4n \quad \forall n \quad n \geq 2$

Graph



Ex:2

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

$g(n) = n^2$ [consider largest Degree]

$2n^2+3n+1$ Replace n terms with n^2

$2n^2+3n^2+n^2$

$\Rightarrow 6n^2$

$f(n) = 2n^2+3n+1 \leq 6n^2 \quad \forall n$

$\forall n \geq 1$

If $n=1$

$6 \leq 6$ — true

$n=2$

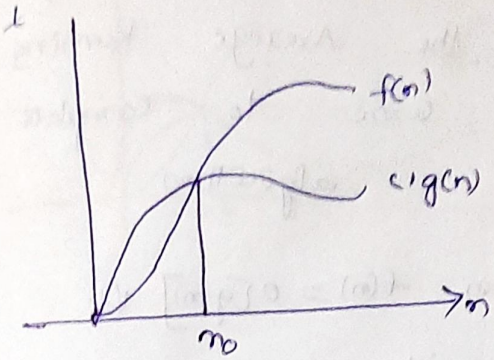
$8+6+1 \leq 6 \times 4$

$14 \leq 24$ True

$3n+2 = \Omega(n)$ when

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

Graph:



Note

$1 < \log n < n < n\log n < n^2 < n^3 < \ldots < 2^n < 3^n < \ldots < n^n$

lower bound of function f(x)

upperbound of function f(n)

tight bound

3. Theta notation (Θ)

it is gives the Average bound of algorithm Running time

it gives the Average Running time to complete the algorithm

Def: The function $f(n) = \Theta[g(n)]$ if there exists some +ve constant c, n_0 such that

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall n, n \geq n_0$$

($c_1 g(n)$: lower bound; $c_2 g(n)$: upper bound)

Example:

$f(n) = 3n+2$

$g(n) = n$

$c_1 = 3, \quad c_2 = 4$

$3 \cdot n \leq 3n+2 \leq 4n$ ∀.

Suppose $n=1$

$3 \leq 5 \leq 4 \rightarrow$ false

$n=2$

$6 \leq 8 \leq 8$ — true

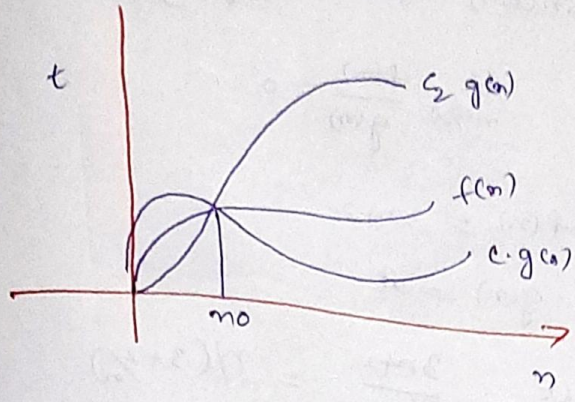
$n=3$

$9 \leq 11 \leq 12 \rightarrow$ True

$\therefore \quad 3n+2 = \Theta[n]$ when

$3n \leq 3n+2 \leq 4n \quad \forall n$

$n \geq 2$



4. Little oh Notation:

The function $f(n) = o[g(n)]$ iff

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

Ex:

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

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

$$\lim_{n \to \infty} \frac{3n+2}{n^2} = \frac{n(3+2/n)}{n^2 n}$$

$$= \frac{1}{\infty} = 0$$

$\therefore f(n) = o[n^2]$ when $\lim_{n \to \infty} \frac{3n+2}{n^2} = 0$

5. Little omega Notation: (ω)

The function $f(n) = \omega[g(n)]$ iff

$$\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0$$