Date / /
Page No.

Unit-I

★ **Data structure:-** A particular organization of data and the relationship among its participating members is called data structure. we can also define " It is an mathematical or logical model of particular organization of data items".
According to Niclaus wirth a program is made up of data structures and an algorithm.

ALGORITHM + DATA STRUCTURE = PROGRAM

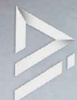
DATA STRUCTURE = ORGANISED DATA + ALLOWED OPERATIONS

★ **Basic Terminology :-**
The various terms connected with the data structure are defined below:-

(1) **Data elements:-** It is the most fundamental Data level which is used as a building block for all other data in a system. This term is also called by names such as field, data items or elementry items.
For exp:- A Date is made up of three parts, → Day, month and year.
Now in this structure

DATE = | DD | MM | YY |

SHOT ON OPPO
By Ghanendra Kumar


SHOT ON OPPO
By Ghanendra Kumar

DD, MM and YY are various data elements and Data is termed as a group or group item.

Primitive Data type :- A group can be divided or decomposed into data elements but a data element cannot be further sub-divided. Therefore, the data elements are called compound data type.
The primitive data can be of many types as given below:

(a) Integer :- An Integer data items are numbers used for counting. An Integer is an integral or whole number without a decimal point. It may be preceded by a sign indicator (+ for positive, - for negative). Exp :- 5, -23, 5267 etc.

General formula for range,

-2^{n-1} to $2^{n-1}-1$
where n = no. of bits

for integer, - 32678 to + 32677.

(b) Real :- Real data items are numbers used for measuring quantities. It is a number with decimal point and can be written in two different forms :-
a) Decimal form
b) Exponential form.

(a) Decimal form :- It is a signed or unsigned string of digits including a decimal point. Even it has an integral value, it must include a decimal point at the end.
Exp :- 234.74, - 921.73, 27.0, 2.009 etc.

b) Exponential form :- It is a floating format containing two parts : matissa and exponent
Exp :- 549.23 E 4 ⇒ 549.23×10^4
(549.23 → Mantissa, 4 → Exponent)

2.0 E - 8 ⇒ 2.0×10^{-8}

(c) Boolean :- The term boolean is from boolean algebra. This is an algebra of logical values i.e, true and false. A Boolean data can have only two values either true or false. This type of data is used to take decisions and answer questions.

(d) Character :- It is a non-numeric data type consisting of single alphanumeric character. The alphanumeric character comprise the following. Alphabets (A-Z and a-z), Digits (0 - 9), Special Symbols (+, -, *, ., /, ,, ₹, $)

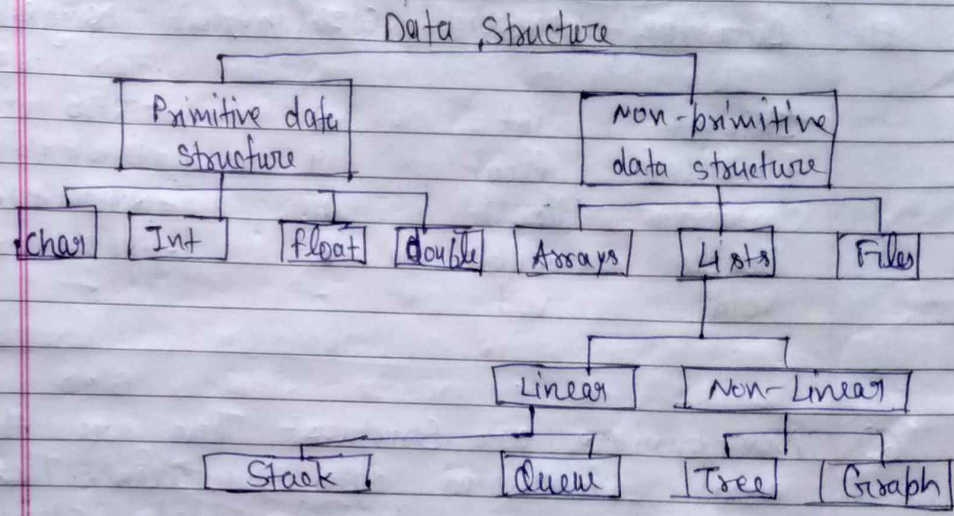
③ Algorithm :- The set of rules that define how a particular problem can be Solved in finite Sequence of steps is known as an Algorithm. the Algorithm written in a Computer language is termed as a program.

④ Constant :- It is a memory location which does not change its value during execution of a program.
for exp:- The value of π is a constant quantity.

⑤ Variable :- It is the most fundamental aspect of any programming language and can be defined as.
" A Variable is a location in the memory where a value can be store. These values can be changed during the execution of a program by its name
Exp: int a , float b;

★ Classification of data structure :-
Data structures are normally divided into two broad categories :-

① primitive data structures
② Non- primitive data structures



(i) Primitive data structure :- Primitive data types are basic data types of any programming language that form the basic unit for the data structure define by the user. A primitive data type defines how the data will be internally represented in, stored and retrieved from the memory. Some primitive data types which are commonly used with most programming languages are as follows.

① Integer (int)
② Character (char)
③ floating point number (float)
④ Double precision number (double)

Data type	keyword	size (in bytes)	Range
Integer	int	2	-32768 to +32767
float/Real	float	4	+ 3.4 E +48 to 3.4 E +48
character	int ~~double~~	1	-128 to +127
Double	double	8	-1.7 E 308 to +1.7 E 308
Void	void	—	—

(2) Non-primitive data structures:- These are the more suffusticated data structures. These are derived from primitive data structures. The non-primitive data structure. The non-primitive data structures emphasize on structuring of a group of homogeneous (same type) or hetrogeneous (different type) data items. Array, lists and files are examples of Non-primitive data structure.

* Data structure operations:- Large number of operations can be performed ~~and~~ on data structures. Some of the important operation are as follows:-

(1) Creation:- A data structure is created.

(2) Insertion:- New items are added to data structure.

(3) Modification:- The values of a data structures are modified by replacing old value.

(iv) Transversing:- Each data item in the data structure is visited for processing purpose

(v) Searching:- A data item is searched in the data structure the data item may or may not be exist in the data structure.

(vi) Deletion:- Deletion is a process of removing an item from the data structure.

(vii) Sorting:- Data items ~~in~~ are sorted in ascending order or descending order.

(viii) Merging:- Data items in more than one sorted data structure are merged together to produce a single new data structure.

(ix) Concatenation:- Data items of a data structure are appended at the end of another same type of data structure.

(x) Copying:- Data items from one structure are copied to another structure.

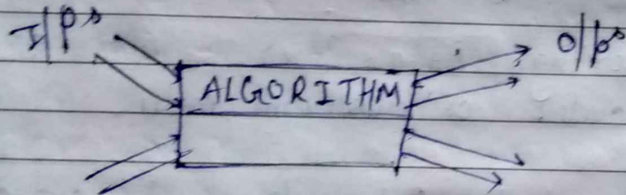
(xi) Splitting:- Data items in a very big data structure are split into smaller structure for processing purpose

SHOT ON OPPO
By Ghanendra Kumar

Algorithm :- An algorithm is a well defined computational procedure that takes some value or set of values as input and It is thus a sequence of computational steps that transform the input into the output.

OR

An algorithm is a procedure or formula for solving a problem step by step.



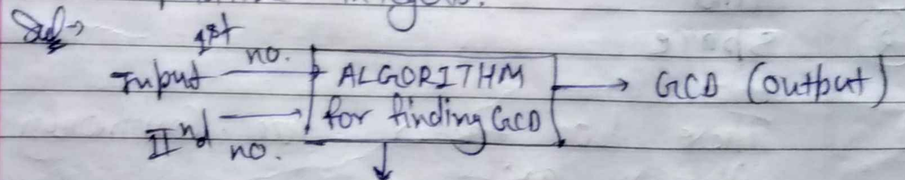
Properties or characteristics of an ALGORITHM:

Algorithm must have following properties!-

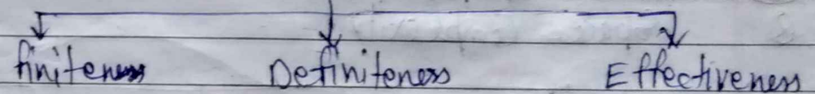
1. Input :- Input data supplied externally (zero or more).
2. output :- There will be at least one output (result of the program).
3. Finiteness! - In every case, algorithm terminates after a finite number of steps.
4. Definiteness :- The steps should be clear and unambiguous.
5. Effectiveness! - An algorithm should be written in using basic instructions. It should be feasible to convert the algorithm in a computer program.

Exp :- Take the problem of finding the GCD (Greatest common divisor) of two positive integers.

Sol →



Algorithm should have these properties



Step 1 : Start
Step 2 :- Read two +ve integer and store them in x and y
Step 3 :- Divide x by y. let the remainder be r and quotient be q.
Step 4 ! - If r=0 then go to step (8)
Step 5 ! - Assign y to x ($x \leftarrow y$)
Step 6 :- Assign r to y ($y \leftarrow r$)
Step 7 :- go to step (3)
Step 8 :- print y (the required GCD)
Step 9 :- Stop

Concept GCD(6,4)

$x \to 6, \; y \to 4$

$r = x \% y = 6 \% 4 = 2$

$r = 2$

if $r = 0$ then $GCD = y$

if $r \neq 0$ then $x \to y$, $y \to r$

$x \to 4$

$y \to 2$

$r = x \% y = 4 \% 2 = 0$

$r = 0$,

$GCD = y = 2$

SHOT ON OPPO
By Ghanendra Kumar

Analysing Algorithms (or efficiency of an algorithm)

The efficiency of an algorithm can be decided by measing the performance of an algorithm. We can measure the performance of an algorithm by computing two factors:-

1. Time
2. Space

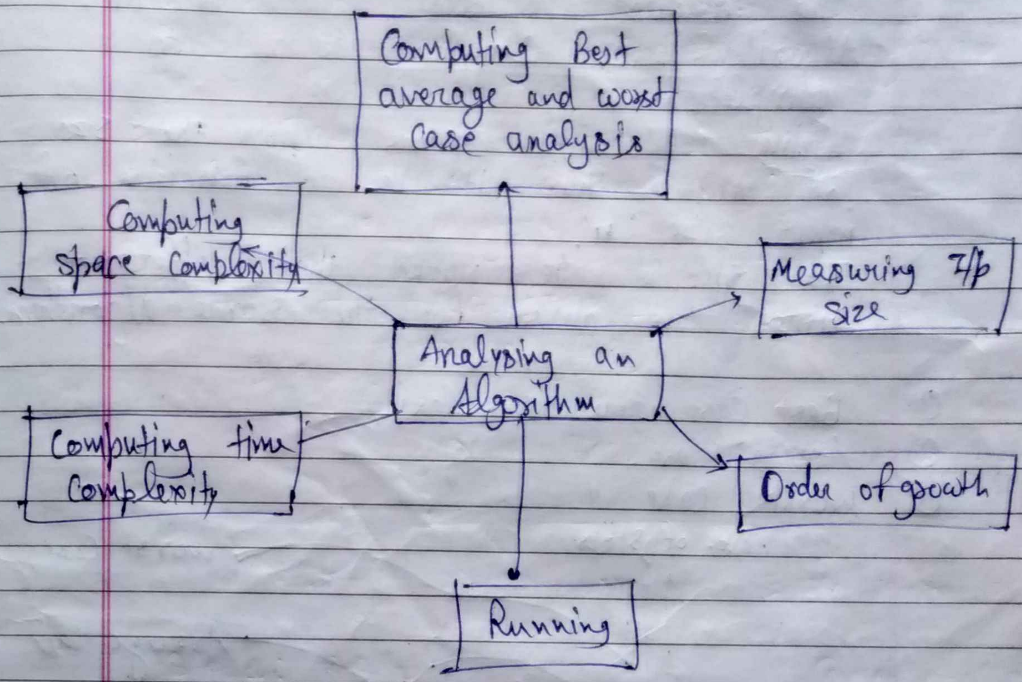
These two factor are commonly known as:-

1. Time Complexity
2. Space Complexity

exp:-

Name of Algorithm	Time (in sec)	Space (in MB)
1	5 sec	100MB
2	10 sec	[illegible]MB
3	12 sec	25MB
4	50 sec	10MB

Algorithm 3 is the best



SHOT ON OPPO
By Ghanendra Kumar

☆ Time Complexity:-

The amount of time needed by an Algorithm or program to run to completion.

Time complexity depends upon the size of input, thus it is a function of input size 'n'.

It should be noted that different time can arise for the same algorithm.

1. Best Case time complexity
2. Average Case " "
3. Worst Case " "

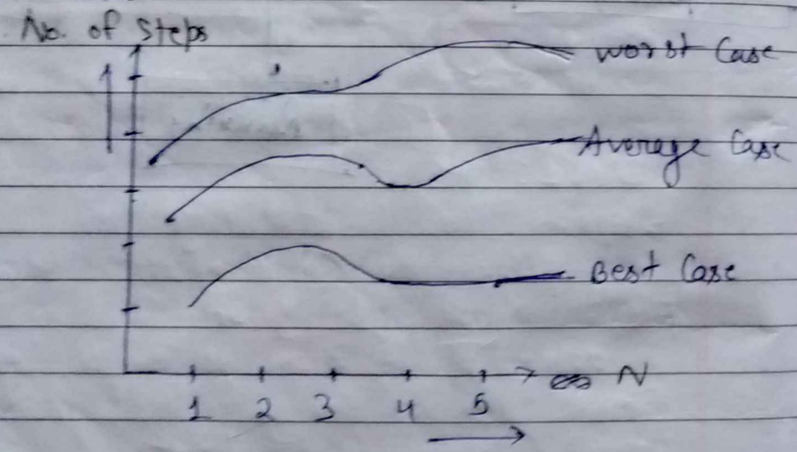


fig: Time complexity

1. Best Case time Complexity:-
   It is the minimum amount of time that an algorithm require for an input of size n.
   It is the function defined by the minimum number of steps taken on any instance of size n.

2. Average Case time Complexity:-
   It is the average amount of time that an algorithm requires for an input of size n.
   It is the function defined by the avg. number of steps taken on any instance of size n.

3. Worst Case time Complexity:-
   It is the maximum amount of time that an algorithm requires for an input of size n.
   It is the function defined by the maximum number of steps taken on any instance of size n.

☆ Space complexity:-

The space complexity can be defined as the amount of memory require by an algorithm or program to run to completion.

To compute the space complexity we use two factors:

→ Constant Characteristic

→ Variable characteristics :-

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

Where C is Contant Characteristic
S_p is variable or instance Char^n

1. Constant Characteristic :-
It is a fixed path and it denotes the space required by input and output. This part includes space for simple variables and space for Constants.

2. Variable or instance Characteristics:-
It is a variable part whose space requirement depends upon perticular problem instance.

Q. Write an algorithm to perform addition of all elements in an array. Compute its space complexity.

Sol-

```
ADD (x, x)
{
   Sum ← 0
  for i ← 1 to n do
   Sum ← Sum + x[i]
  return Sum
}
```

$S(P) = C + S_p$
$C = i + n + Sum = 3$
$S_p = x[\,] = n$

$$S(P) = 3 + n$$

```
#include <stdio.h>
# include <conio.h>
void main ()
{
    int i, n, sum, A[50];      - 1
        sum = 0;               - 1
    printf (" Enter the size of the array \n");   - 1
    scanf ("%d", &n);          - 1
    printf (" Enter the elements of array \n");   - 1
 for (i=1;  i<= n;  i++)       - n+1
  {
       scanf ("%d", &A[i]);    - n
  }
  for (i=1;  i<= n;  i++)      - n+1
   {
       sum = sum + A[i];       - n
   }
  printf (" Sum of elements = %d", sum);   - 1

   getch ();   - 1
}
```

SHOT ON OPPO
By Ghahendra Kumar

Space requirement

$S(P) = C + S_p$

$C = i + n + Sum = 2 + 2 + 2 = 6$ bytes — size of integers

$S_p = A[\,] = n$

$S(P) = 6 + n$

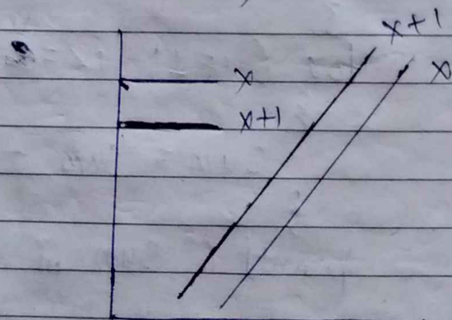
Time complexity

$4n + 9$

$T(n) = O(n)$

☆ Asymptotic Notations (or performance Analysis of Algorithm) :-

→ Asymptotic :- A line that continuously approaches to a given curve but does not meet it at any finite distance.



x is Asymptotic with x+1

The notation which we used to describe the asymptotic running time of an algorithm are defined in terms of the size of the input n.
Asymptotic notation describes the efficiency and performance of an Algorithm in a meaningful way.

Following notations are commonly used to characterise the complexity of an Algorithm.

1) Big oh (O) notation (worst)
2) Theta (Θ) notation (Average)
3) Omega (Ω) notation (best)

1) Big oh (O) notation :- Big oh (O) notation is the formal method of representing the upper found of an Algorithm running time. Using big oh notation we can give the longest amount of time taken by an Algorithm to complete.
It can be defined as :
Let $f(n)$ and $g(n)$ be two non-negative functions, if there exists an integer n_0 and a constant $c > 0$ such that for for all integers $n \geq n_0$

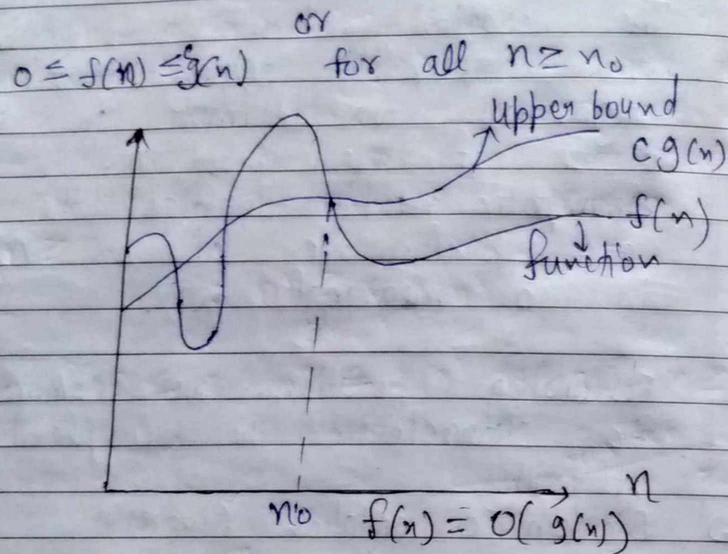
$$f(n) \leq g(n)$$

SHOT ON OPPO
By Ghanendra Kumar

Date / /
Page No.

or

$0 \le f(n) \le g(n)$ for all $n \ge n_0$



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

Q: Consider a function $f(n) = 2n+1$ and $g(n) = n^2$ then find the value of n. To prove that above two functions are in big oh (O) notation.

Sol: We know that

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

or

$$f(n) \le g(n)$$

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

if $n = 0$

$f(0) = 1$, $g(0) = 0$

$f(n) \le g(n)$ false

Date / /
Page No.

if $n = 1$

$f(1) = 3$, $g(1) = 1$

$f(n) \le g(n)$ false

if $n = 2$

$f(2) = 5$, $g(2) = 4$

$f(n) \le g(n)$ false

if $n = 3$

$f(3) = 7$, $g(3) = 9$

$f(n) \le g(n)$ true

if $n = 4$

$f(4) = 9$, $g(4) = 16$

$f(n) \le g(n)$ true

so,

for $n \ge 3$ is true.

All cond^ns are satisfied

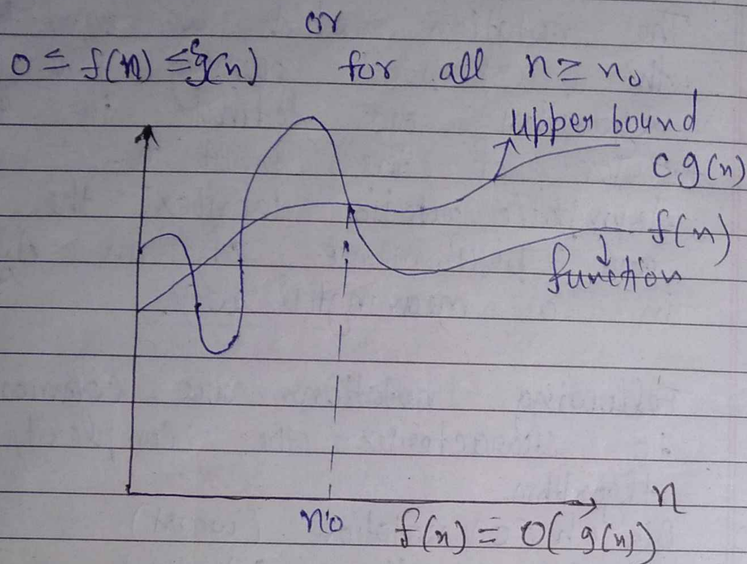
so, f(n) and g(n) both functions are in big oh notation.

2. ~~Worst~~ Best Case (Ω) / omega notation :-

It denoted by simple Ω. This notation is used to represent the lower bound of the algorithm running time. Using Ω notation we can denote the shortest amount of time taken by Algorithm.

or

$0 \le f(n) \le c\,g(n)$ for all $n \ge n_0$



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

Q:- Consider a function $f(n) = 2n+1$ and $g(n) = n^2$ then find the value of n. To prove that above two functions are in big oh (O) notation.

Sol:- We know that

$f(n) \le c.\, g(n)$

or

$f(n) \le g(n)$

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

if $n=0$

$f(0)=1$, $g(0)=0$

$f(n) \le g(n)$ false

if $n=1$

$f(1)=3$, $g(1)=1$

$f(n) \le g(n)$ false

if $n=2$

$f(2)=5$, $g(2)=4$

$f(n) \le g(n)$ false

if $n=3$

$f(3)=7$, $g(3)=9$

$f(n) \le g(n)$ true

if $n=4$

$f(4)=9$, $g(4)=16$

$f(n) \le g(n)$ true

so,

for $n \ge 3$ is true.

All cond^ns are satisfied

So, f(n) and g(n) both functions are in big oh notation.

2. ~~Worst~~ Best Case (Ω) / omega notation :- It denoted by simple Ω. This notation is used to represent the lower bound of the algorithm running time. Using Ω notation we can denote shortest amount of time taken by Algorithm.

Theta (Θ) notation :-

It is denoted by sign 'Θ'. By this method the running time of an algorithm is b/w the lower bound and upper bound.

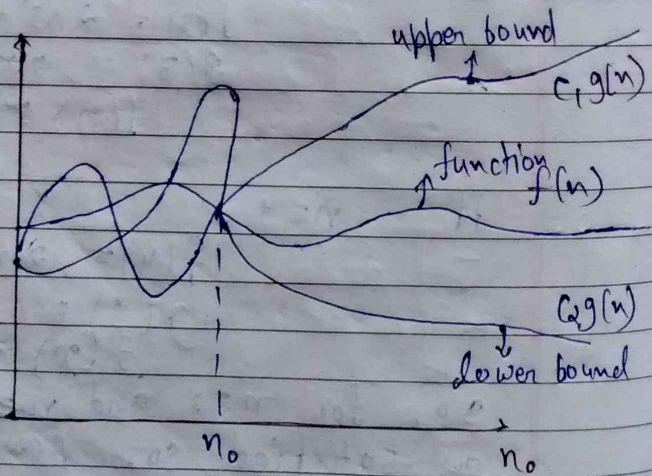
It can be defined as:

Let $f(n)$ and $g(n)$ be to non-negative functions if there exist an integer ~~cannot~~ n_0 and positive (+) constant c_1 and c_2 that is $c_1 > 0$ and $c_2 > 0$ such that for all integers $n \geq n_0$.

$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

or

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 \cdot g(n) \quad \text{for all } n \geq n_0$$



$$f(n) = \Theta(g(n))$$

Q. Consider a f^n $f(n) = 2n+8$ and $g(n) = 7n$. Compute the value of n. To show that the above two functions are in Theta notation.

Sol. We know that by Θ notation:

$$f(n) = 2n+8, \quad g(n) = 7n$$

$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

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

$$7n \leq 2n+8 \leq 7n$$

If $n=0$

$$f(0) = 2\times 0 + 8 = 8$$

$$g(0) = 7\times 0 = 0$$

$$0 \leq 8 \leq 0 \quad \text{is false}$$

If $n=1$

$$f(1) = 2\times 1 + 8 = 10$$

$$g(1) = 7\times 1 = 7$$

$$7 \leq 10 \leq 7$$

If $n=2$

$$f(2) = 2\times 2 + 8 = 12$$

$$g(2) = 7 \times 2 = 14$$

$$14 \leq 12 \leq 14$$

★ Abstract Data types:-
It is a data type having set of values and set of associated operations that are precisely specified independent of any particular implimentation.
Abstract data type is also defined as a mathematical model which is a user defined type along with the set of operations that can be performed on that model.

★ Some examples of Abstract datatype:-

① Lists:- It is an abstract datatype which includes a finite set of items along with the following operations.
→ Creation
→ IsEmpty
→ IsFull
→ Length
→ Insertion
→ Deletion

★ Creation:- This operation creates a list and leaves it empty.

★ IsEmpty:- This operation checks whether the list is empty or not.

★ IsFull:- This operation check whether the list full or not.

★ Length:- This operation computes the length of the list.

★ Insertion:- This operation inserts an item in the list only when it is not full.

★ Deletion:- This operation deletes an items from the list only when list is not empty.

② Stack:- It is an abstract datatype which include a finite set of item along with the following operations.
Creation, IsEmpty, IsFull, Length, Insertion, Deletion.

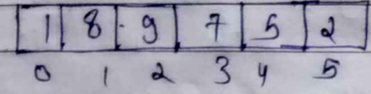
It works on lifo technique
LIFO ÷ [Last in first out]

③ Queue:- It is an abstract datatype which include a finite set of item along with the following operations.
Creation, IsEmpty, IsFull, Length, Insertion, Deletion.
It works on FIFO technique.

SHOT ON OPPO
By Ghanendra Kumar

SHOT ON OPPO
By Ghanendra Kumar

Array :- • Array is a collection of elements
All elements are of the same types (homogeneous)
All elements are stored using continuous memory locations.

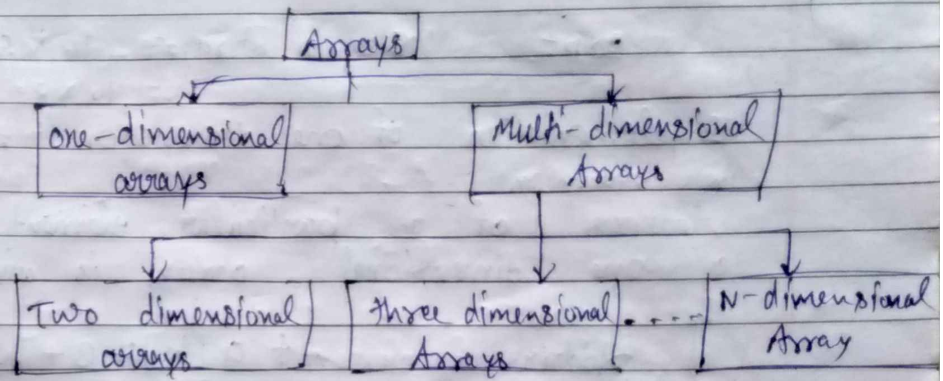


- Elements of an array can be initialised at the time of declaration.
- A particular value is access by writting a no. called index no. OR sub-script in bracket after the array name.
  Exp :- int a [5] = {3, 4, 5, 6}
  a[1] = 4
- Array index always start from 0 and goes upto n-1. where n is the size of the array.

int a [5];

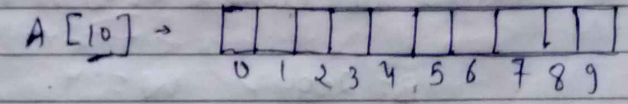
index ← 0	4	→ a[0]
← location 1	5	→ a[1]
[index + 1] 2	2	→ a[2]
3	7	→ a[3]

Classification of Arrays :-



☆ One dimensional Array :-
A one dimensional Array is one in which only one subscript specification is needed two specify a particular elements of the Array.

Syntax :- data type vari-name [size]
exp :- int A [10];

A [10] →


no. of Elements / Max_Size = (upper bound - lower bound) + 1
= (9 - 0) + 1 = 10

The total memory in bytes that array would occupy will be given by -

~~Elements~~ Size of Array = No. of elements in array × Size of (data-type);

Date / /
Page No.

expt Size of Array = 10 × Size(int)
= 10 × 2 = 20 bytes

* Implementation of one dimensional Array in memory.

Address of a particular element in a one dimensional array is given by

Address of element + a[K] = B + W * (K-LB)
Address of element + a[K] = B + W * K

where,
B = Base address
W = size of each elements of the Array
K = no. of required element in the array (or index of element).

Q: Base address (B) = 2000
size of each element (W) = 4 byte.
Address of element a[5] = ?

B = 2000, W = 4, K = 5
a[5] = 2000 + 5×4
a[5] = 2000 + 20 = 2020

Q→ Consider the Linear Array A(5:50), B(-5:10) and C(18).
(a) find the no. of elements in each arrays.
(b) Suppose, Base address of A = 300 and W = 4 bytes for A find the address of A[15], A[40] and A[55].

Date / /
Page No.

Sol: b) A[15] = 300 + 4 × [15-5]
= 300 + 40
A[15] = 340

A[40] = 300 + 4 × [40-5]
4 × 35
= 300 + 140
= 440

a) A = 50-5+1 = 46
B = 10-(-5)+1
= 10+5+1 = 16
C = 18

② Two dimensional Array :-
Two dimensional array are as a grid
Two dimensional arrays be also called as a table consisting of raws and coloums.
Two dimensional Arrays are used to perform various operations on the matrix such as addition, subtraction, multiplication, transpose, etc.
Two dimensional array are also called matrix.

Declaration of a two-dimensional Arrays :-

Syntax :-

SHOT ON OPPO
By Ghanendra Kumar

```
data-type   array-name [size 1] [size 2];
                          ↓          ↓
                       size of    size of
                        rows      Columns
```

exp:

```
int A[5][4];
      ↓
     rows
```

★ Initializing a two dimensional array:-

① Compile time initialization:-

```
int A[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
            or
int A[3][3] = {{1,2,3}, {4,5,6}, {7,8,9}};
```

② Run-time initialization:-

```
int A[10][10];
printf("Enter the size of raw and column");
Scanf("%d%d", &m, &n);
printf("Enter the element of array\n");
For (i=0; i<m; i++)
{
    for(j=0; j<n; j++)
    {
        scanf("%d", A[i][j]);
    }
}
printf("Matrix");
for (i=0; i<m; i++)
{
    for (j=0; j<n; j++)
    {
        printf("%d\t", A[i][j]);
    }
    printf("\n");
}
getch();
}
```

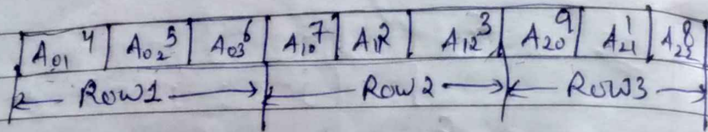
★ Implementation of a two-dimensional array:-

A two dimensional array can be implemented in a programming language in two ways

1) Row major Implementation:-
2) Column Major Implementation:-

1) Row major Implementation:-
Row major Implementation is a linearization technique in which elements of array are read from the keyboard row wise That is the complete first row is stored then the complete second row is stored and so on.
For exp:-
An array A[3][3] is stored in the memory as shown in figure.

SHOT ON OPPO



$$A[3][3] = \begin{bmatrix} 4_{A_{01}} & 5_{A_{02}} & 6_{A_{03}} \\ 7_{A_{10}} & 2_{A_{11}} & 3_{A_{12}} \\ 9_{A_{20}} & 1_{A_{21}} & 8_{A_{22}} \end{bmatrix}_{3\times3}$$

★ Address of elements in Row Major Implementation

Address of element, $A[i][j] = B + w(n(i-L_1) + (j-L_2))$.

where,

- $B \to$ Base address
- $w =$ size of each element
- $n =$ no. of column (i.e $U_2 - L_2 + 1$)
- $L_1 \to$ Lower bound of row
- $L_2 \to$ Lower bound of Column
- $U_1 \to$ Upper bound of row
- $U_2 \to$ Upper bound of Column

Q→1 A two dimensional array defined as a[4...7, -1...3] requires 2 bytes of Storage space for each element. If the array is Stored in Row major from Then Calculate the address of element at location A[6, 2] or a[6][2]. Given that the Base address is 100.

Sol→ Given, $B = 100$, $L_1 = 4$, $U_1 = 7$, $L_2 = -1$, $U_2 = 3$

$w = 2$ bytes, $a[6,2] = ?$

$n = U_2 - L_2 + 1$

$n = 3 + 1 + 1 = 5$

$A[6][2] = 100 + 2[5(6-4) + (2+1)]$
$= 100 + 2[10 + 3]$
$= 100 + 2 \times 13$
$= 100 + 26 = 126$ Ans

Q→2 Each element of an array Data[20][50] requires 4 bytes of storage. base address of Data is 2000. Determine the Location Data[10][10] when the array is ~~storaged~~ Stored in Row major form.

Sol→ $W = 4$, $B = 2000$, $Data[10][10] = ?$

$n = 50$, $Data[0...19][0...49]$

~~$L_1 = 0$, $L_2 = 0$, $U_1 =$~~

$L_1 = 0$, $U_1 = 19$

$L_2 = 0$, $U_2 = 49$

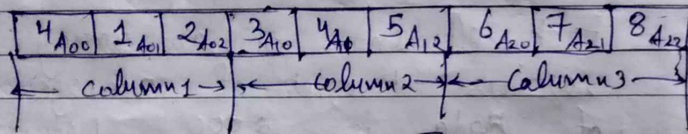
$Data[10][10] = 2000 + 4[50[10-0] + 10 - 0]$
$= 2000 + 4[500 + 10]$
$= 2000 + 4 \times 510$
$= 2000 + 2040$
$= 4040$

SHOT ON OPPO
By Ghanendra Kumar

Column Major Implementation :-

In column major implementation memory allocation is done column by column. i.e first the elements of the complete first column is stored then elements of complete second column is stored and so on.

For exp :-
An Array A[3][3] is stored in memory as shown in the figure.



$$A = \begin{bmatrix} 4_{A_{00}} & 3_{A_{01}} & 6_{A_{02}} \\ 1_{A_{10}} & 4_{A_{11}} & 7_{A_{12}} \\ 2_{A_{20}} & 5_{A_{21}} & 8_{A_{22}} \end{bmatrix}$$

Address of element in Column Major Implementation

Address of element, $A[i][j] = B + w(m(j-L_2) + (i-L_1))$.

Q→1 Each element of an array A[-20....20, 10....35] requires 1 byte of storage. If the array is column major implementated and the begging of the array is at location 500. Determine the address of element A[0,30] or a[0][30].

Sol→
$B = 500$, $L_1 = -20$, $U_1 = 20$
$W = 1$, $L_2 = 10$, $U_2 = 35$
$m = U_1 - L_1 + 1 = 20 + 20 + 1 = 41$

$A[0,30] = 500 + 1[41(30-10) + (0+20)]$
$= 500 + (41 \times 20 + 20)$
$= 500 + 820 + 20$
$= 500 + 840$
$= 1340$

Q→2 Remain same as Row major Implementation But find data [10][0] by column major Implementation.

Sol→
$B = 2000$, $W = 4$
$L_1 = 0$, $U_1 = 19$
$L_2 = 0$, $U_2 = 49$
$m = 19 - 0 + 1 = 20$

$A[10][10] = 2000 + 4[20(10) + (10-0)]$
$= 2000 + 4[200 + 10]$
$= 2000 + 4 \times 210$
$= 2000 + 840$
$= 2840$ Ans

☆ Three dimensional Array :-

Consider a '3' dimensional array A[a:b, c:d, e:f]
let α_1, α_2 and α_3 are total sizes of 1st, 2nd and 3rd dimensions of array.
suppose. we want to calculate add. of A[P, Q, R], the effective index.
I = P-a , J = Q-c and K = R-e

☆ Row major order :-

For calculating add. of a A[P,Q,R] Base add + W * [(i * α_2 + J) * α_3 + k]

☆ Column Major order :-

→ For calculating add of A[P,Q,R] =
B + W * [(k × α_2 + J) × α_1 + i]

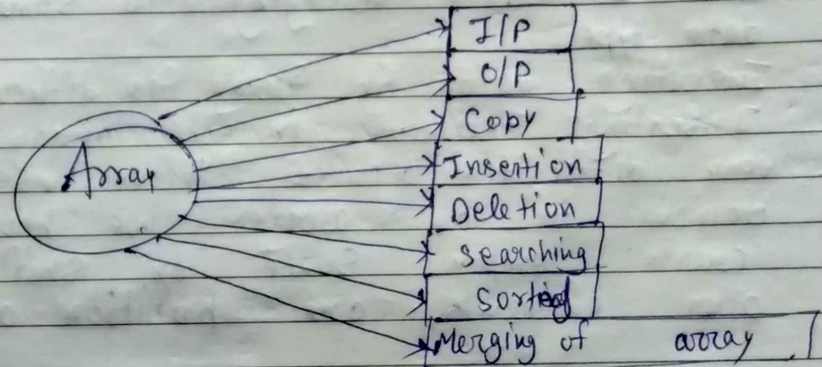
Q→ A[2,3,4], B = 100, W = 4 bytes obtain memory location of A[1,1,2], $\alpha_1 = 2$, $\alpha_2 = 3$, $\alpha_3 = 4$, Row major, column.

Sol→ A[1:2 1:3 1:4]
I = 1-1 = 0, J = 1-1 = 0
K = 2-1 = 1

= 100 + 4 × [(0 × 3 + 0) × 4 + 1] = 104

Column:
= 100 + 4 [(1×3+0) × 2 + 0]
= 100 + 4 [6]
= 124

☆ operation of Array :-



☆ Application of Array :-

(i) Addition of '2D' Matrixes.
(ii) Subtraction of '2D' Matrixes
(iii) Transpose of a sq. matrixes.
(iv) Multiplication of two matrix
(v) Finding whether given sq. matrix is symmetrical or not.

☆ Limitation of Array :-

(i) static data :-
(a) Array is Static data structure

(b) (c) Memory allocated during compile time once's memory is allocated at Compile time. it can not be change during run time.

2) Can hold data belonging to same data type.

3) Inserting data in array is difficult :-
Inserting element in the array is very difficult because before inserting element in an array. we have to create empty space by shifting other elements one position a head.

(4) Deletion operation is difficult :-
Deletion is not easy because the element are stored contingous memory location.

5). Wastage of memory:-
If array of large size is defines

Date 16/07/22

☆ Time Space Trade off :-

Time and space requirement of an Algorithm should be minimum. It is desirable to design an algorithm which consumes minimum memory and time.
However it is very difficult to design an Algorithm which archieves these parameters.

Table 1

Algorithm	Time in Second	Space (in MB)
Algo 1	10	100
Algo 2	12	120

It means that Algo 1 is better than Algo 2. (In Case of Time).

Table 2

Algorithm	Time (in Second)	Space (in MB)
Algo 1	8	200
Algo 2	10	100

It means that Algo 2 better than better than Algo 1 (Increase of memory space).

' which one is going to be used up.
The answer is Simple.
There are two Criteria (Time and space) for developing the software in which the parameter is more important, time or space.
If time matters select Algorithm 1. in table 1.
If space matter select Algorithm two in table Algo 2.

SHOT ON OPPO
By Ghanendra Kumar

Date / /
Page No.

☆ Sparse Matrices :-
In Sparse Matrices, no. of '0' elements are more than non zero elements.
For exp :-
The following 4x4 matrix A is a sparse matrix

	0	1	2	3
0	0	0	9	0
1	0	1	0	0
2	5	0	9	0
3	8	0	0	6

A =

fig :- A sparse Matrix

☆ Representation of sparse Matrices :-

☆ Sparse matrix contain three parts

1) Its Row (No. of rows in original matrix)
2) Its Row (" " Column " " ")
3) Its Value (" " Non-zero elements).

B =

No. of Rows (4)	No. of Columns (4)	No. of Non-zero elements (6)
0	2	9
1	1	1
2	0	5
2	2	9
3	0	8
3	3	6

Date / /
Page No.

fig :- Representation of space Matrix :-

☆ operations on a sparse matrix :-

→ From a '2D' representation to sparse representation.
→ sparse representation to a '2D' representation
→ Transpose
→ Addition
→ Multiplication
→ Reading a Sparse Matrix as a list of triples.
→ Displacing a Sparse Matrix as a list of triple.
→ Inserting an element (row, column, and Values) In a list of triples.

Q :- Write an algorithm to find the Kth element is a sequence of n elements.

Sol :- Let A be the array, k is the key to Search and n is the no. of elements.

1) for J = 0 to n
2) if (A [I] = k)
   print (A[I])
3) Stop or EXIT

Search k = 11

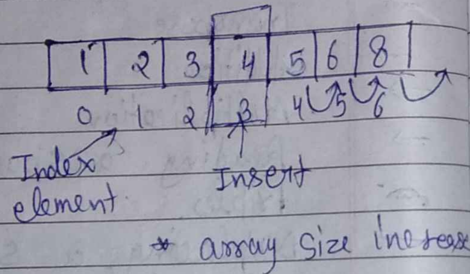
5	6	7	11	12
0	1	2	3	4

SHOT ON OPPO
By Ghanendra Kumar

Q. Write an Algorithm to insert an element is the array at a specified position.

Sol. Let A be the array n is the no. of elements and k is the location.

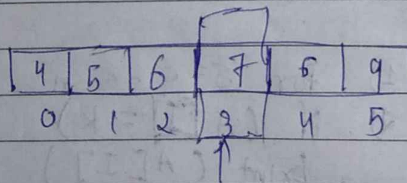
Sol.
1) Initialize a Counter I=n
2) While (I ≥ K)
3) A[I+1] = A[I]
4) I = I-1
5) End loop
6) Set A[K] = item
7) n = n+1
8) EXIT



* array size increase

Q. Write an algorithm to delete an element from the array at a specified position.

Sol.
1) Set item = A[K]
2) for I=K to n-1
3) Set ~~I=K to n-1~~ A[I] = A[I+1]
4) ~~Set A[E]~~ End loop
5) n = n-1
6) Exit

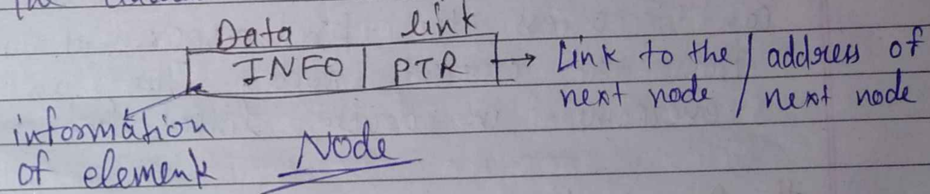


delete → put this in ~~this~~ item and decrease

☆ Linked List :-

Linked list is a non-primitive linear data structure. A linked list is an ordered Collection of finite homogeneous data elements Called notes, where the linear order ~~is called~~ is maintain by means of links or pointers that is :- each ~~is~~ node is divided into two parts:

The first part Contains the information of the element and the second part contains the address of the next node in the list



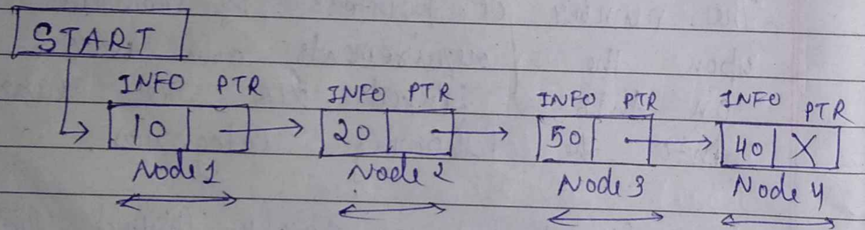
The number of pointers is maintain depending upon the requirements and uses: Based on this, Linked list are classified into the following Categories.

1) Linear linked list or singly linked list
2) Doubly linked list
3) Circular linked list
4) Header linked list
5) Circular doubly linked list.

1) Singly linked list :-
A singly linked list is one in which are nodes are linked together in some sequential manner. Hence it is called linear linked list.
That is each node has a single pointer to the next node and in the last node case a null pointer representing that there are no more nodes in the linked list.

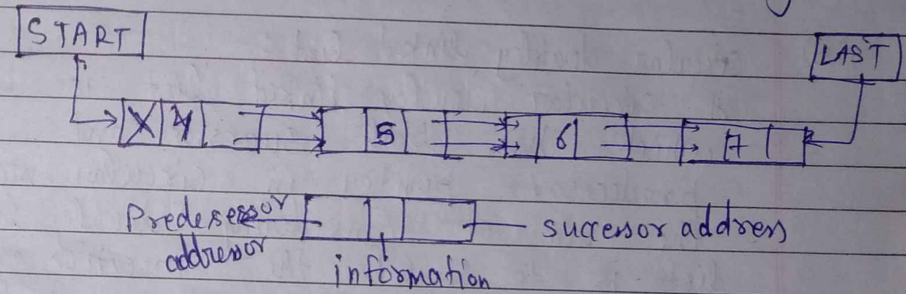
* The problem with this list is that we can't access the predecessor of node from the current node. This can be overcome in doubly linked list.

Diagram:-

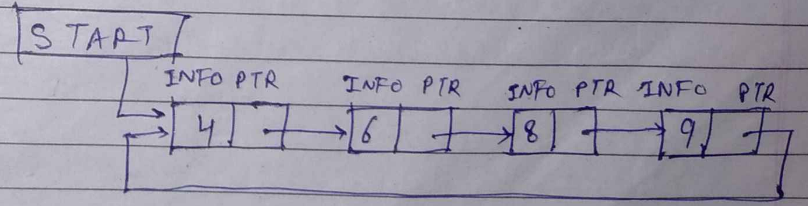


* Fig - singly linked list

* Doubly linked list :- A doubly linked list one in which all nodes are linked together by two links which help in accessing successor (next node) and the predecessor node (previous node) for an arbitary node with the list. Therefore each node in a doubley linked list fields (pointers) to the left node (previous node) and the right node. This help to travel the list in the forward direction.
→ A doubly linked list is shown in figure

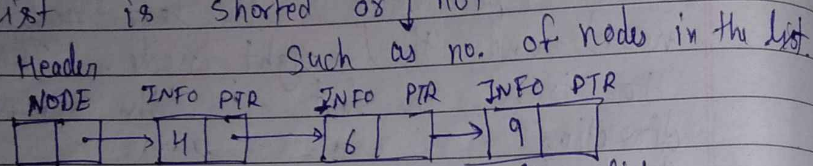


③ Circular linked list :- It is just like a linear list in which the second part of the list that is the second field of the last node does not point to the new pointer rather it point back to the link list.



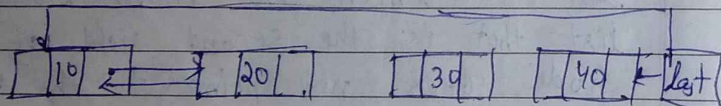
④ Header linked list :- In header linked list always contain a single node called header

node at the begining of the link list. This header node Contain vital information about the linked list, or whether the list is shorted or not
Such as no. of nodes in the list.

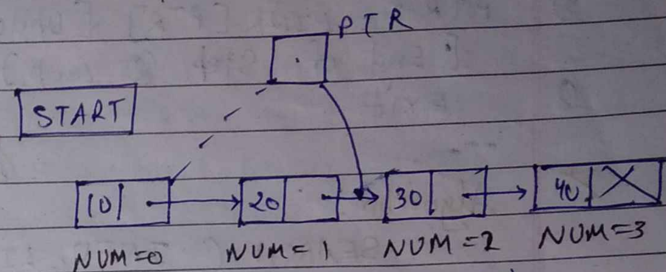


* Fig. Header link list.

(5) Circular doubley linked list :-
A circular doubley linked list is one which has both sucessor and predecesor pointer in circular manner. The aim of considering doubley linked list is to simplify the insertion and deletion operation performed on doubley linked list.



★ Transversing a linked list :-



Algorithm :- TRAVERSE (INFO, LINK, START)
PTR = LINK [PTR]

1) Set PTR = START [Initialises pointer PTR]
2) Repeat steps ③ and ④ while PTR ≠ NULL
3) Apply process to INFO [PTR]
4) set PTR = LINK [PTR] [PTR now points to next node].
5) [End of step ② Loop]
6) Exist.

Algorithm :-
PRINT (INFO, LINK, START)
1) Set PTR = START
2) Repeat Step ③ and ④ while PTR ≠ NULL
3) Write : INFO [PTR]
4) set PTR = LINK [PTR] [update pointer]
[End of step ② Loop].
5) EXIT

Algorithm :- COUNT (INFO, LINK, START, NUM).
1) Set NUM = 0 [Initializes Counter]
2) Set PTR = START [Initialies pointer]
3) Repeat steps ④ and ⑤ while PTR ≠ NULL

4) Set NUM = NUM + [Increase NUM by 1].
5) PTR = LINK [PTR] [Updates pointer)
[End of Step ② Loop]
6 Exit

★ Algorithm:-
SEARCH (INFO, LINK, START, ITEM, LOC):-
Initially, LOC = NULL.
1) Set PTR = START
2) Repeat steps ③ while PTR ≠ NULL
3) If ITEM = INFO [PTR], then
set LOC = PTR and EXIT
Else
set PTR = LINK [PTR] [PTR now points to the next node]
[End of If structure]
4) [Search ~~of If structure~~ is unsuccessfull]
Set LOC = NULL
5) EXIT

★ Inserting nodes into link list :-

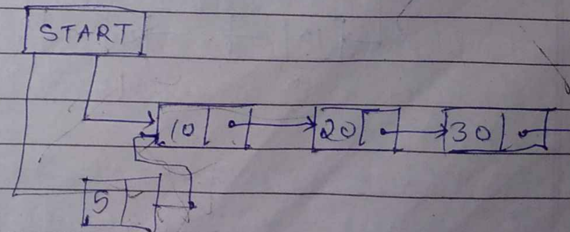
To insert an element in the link list following 3 things should be done.

① Allocating node
② Assiging the data
③ Adjusting the pointer

Inserting a new node into the link list as following three instances:-
1. Insertion at the begining of the list
2. Insertion at the end of the list
3. Insertion at the specified position within the list.

1. Insertion at the begining of the list :-

INSERT_FIRST (START, ITEM)
1) [Check for overflow?]
If PTR = NULL, then
Print 'overflow'
Exit
Else
PTR = (NODE*) malloc (size of (NODE))
[End If]
2) Set PTR → INFO = ITEM
3) Set PTR → NEXT = START
4) set START = PTR

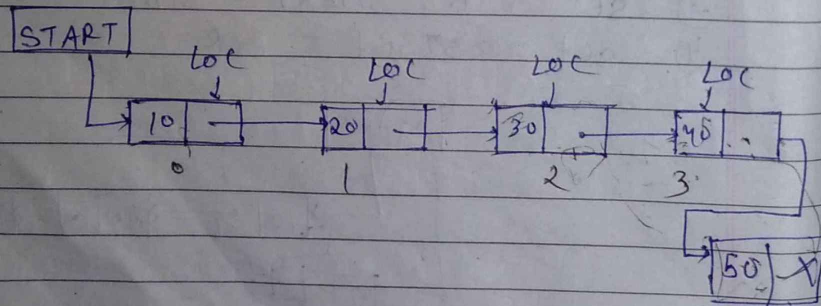


SHOT ON OPPO
By Ghanendra Kumar

2) Insertion at the End :-

INSERT LAST (START, ITEM).

1) [Check for Overflow?]
   If PTR = NULL, then
      Print 'overflow'
      Exit
Else
   PTR = (NODE *) malloc(size of (NODE))
      [Exit If]
2) Set PTR → INFO = ITEM
3) Set PTR → NEXT = NULL
4) If START = NULL, then
      set START = PTR
5) Set LOC = START
6) Repeat step ⑦ until
      LOC → Next ! = NULL
7) Set LOC = LOC → NEXT
8) set LOC → NEXT = PTR



3) Insertion at the specified position:-

INSERT_LOCATION (START, ITEM, LOC)

1) [Check for overflow?]
   If PTR = NULL, then
      print 'overflow'
      Exit
   Else
   PTR = (NODE *) malloc (size of (NODE))
   [End If]
2) Set PTR → INFO = ITEM
3) If START = NULL, then
      Set START = PTR
      Set PTR → NEXT = NULL
   [End If]
4) [Initialize the Counter (I) and pointers]
   NODE * TEMP
   Set I = 0
   Set TEMP = START
5) Repeat steps ⑥ and ⑦ until I <L
6) Set TEMP = TEMP → NEXT
7) Set I = I + 1
8) Set PTR → NEXT = TEMP → NEXT
9) set TEMP → NEXT = PTR [INFO]

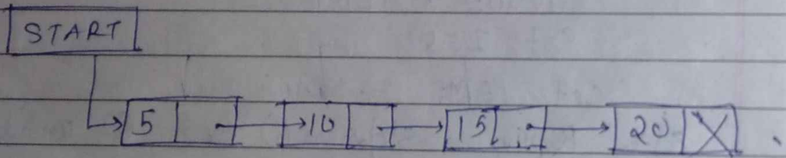
SHOT ON OPPO
By Ghanendra Kumar

PTR = NULL
START = NULL

Date / /
Page No.

★ Deletion:-
① Deleting the first node
② Deleting the last node
③ Deleting the node at specified position.

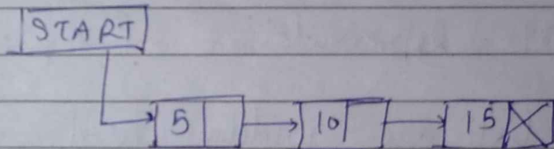
Algorithm:
→ Deleting the ~~last~~ first node:-
① DELETE_FIRST (START)
1) [check for under flow?]
If START = NULL, then
print 'Underflow', and
Exit
[End If]
2) Set PTR = START
3) Set START = START → NEXT
4) Print, Element deleted is, PTR → INFO
5) FREE (PTR).



②→ Deleting the last node!-
ALGORITHM
DELETE_LAST (START)
[Check for under flow?]
If START = NULL, then
print 'Underflow' and
Exit

Date / /
Page No.

[End If]
~~2) Set If PTR = START~~
2) If START → NEXT = NULL, then
Set PTR = START
Set START = NULL
print, Element deleted is PTR → INFO
free (PTR)
[End If]
3) Set PTR = START
4) Repeat step ⑤ and ⑥ Until PTR→NEXT != NULL
5) set LOC = PTR
6) Set PTR = PTR → NEXT
7) Set LOC → NEXT = NULL
8) free (PTR)

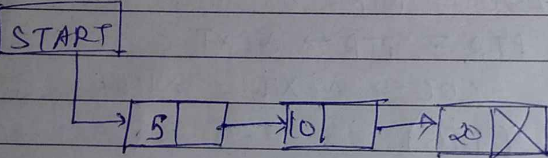


③ Deleting the node at specified position.
ALGORITHM:-
DELETE_LOCATION (START, LOC)
1) [Check for Under flow?]
If START = NULL, then
print 'Underflow' and
Exit
[End If]

SHOT ON OPPO

Date / /
Page No.

2) [Initialize the Counter(I) and pointers]
NODE * TEMP
Set I=0
Set PTR = START
3) Repeat Step (4) to (6) until I < LOC
4) Set TEMP = PTR
5) set PTR = PTR → NEXT
6) Set I = I+1
7) print, Element deleted is, PTR → INFO
8) Set TEMP → NEXT = PTR → NEXT
9) Free(PTR).



M. Imp

★ Linked list Representation of ploy polynomial :-

Linked list are widely used to represent and maniplulate polynomials. Polynomials are the expressions containing no. of terms with non-zero coefficients and exponents.

polynomial, $p(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n$
a_i are non-zero coefficient

for $i = (0, 1, 2, 3, 4, 5 \dots)$

Date / /
Page No.

In the linked list Reprn of polynomial each term is Considered as node and such a node contains.

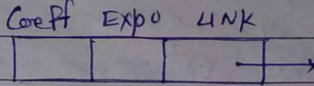
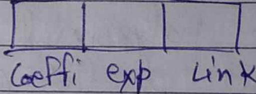


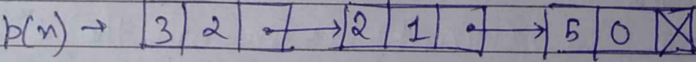
fig. A structure of a polynomial node.

(1) Coefficient field.
(2) Exponent field
(3) Link field.

exp:-



$p(x) = 3x^2 + 2x + 5$



★ Application of Linked List :-

(1) → Addition of two polynomial :-

The Steps involve in adding two polynomial are given below :-

(i) Read the no. of terms in the first polynomial, P
(ii) Read the coefficient and Exponent of the first polynomial
(iii) Read the no. of terms in the second polynomial, Q.
(iv) Read the coefficient and exponent of the

SHOT ON OPPO
By Ghanendra Kumar

second polynomial.

(v) Set the temperary pointer p and q to traverse the two polynomial respectively.

(vi) Compare the exponents of two polynomial Starting from the first node

(a) If both exponents are equal then add the Coefficient and Store it in the resultant linked list, R.

(b) If the exponent of the current term in the first polynomial P is less then the exponent of the current term of the second polynomial Q.

Then

When the current term of the 2nd polynomial is added to the resultant linked list and move the point q to the next node in the Second polynomial

(c) if the exp. of current term in the 1st polynomial is greater than the exponent of the current term in the 2nd polynomial Q. Than the current term of the first polynomial is added to the resultant linked list and move the pointer p to the next node in the first polynomial.

(d) Append the remaining node of Either of the polynomials to the resultant linked list.

exp:- $P = 3x^2 + 2x + 7$

$Q = 5x^3 + 2x^2 + x$

Step ①
P → [3|2] → [2|1] → [7|0|X]
Q → [5|3] → [2|2] → [1|1|X]
R → [|] → [|] → [| |X]

Step ②
P → [3|2] → [2|1] → [7|0|X]
Q → [5|3] → [2|2] → [1|1|X]
R → [5|3] → [|] → [| |X]

Step ③
P → [3|2] → [2|1] → [7|0|X]
Q → [5|3] → [2|2] → [1|1|X]
R → [5|3] → [5|2] → [| |X]

Step ④
P → [3|2] → [2|1] → [7|0|X]
Q → [5|3] → [2|2] → [1|1|X]
R → [5|3] → [5|2] → [3|1|X]

Step ⑤
P → [3|2] → [2|1] → [7|0|X]
Q → [5|3] → [2|2] → [1|1|X]
R → [5|3] → [5|2] → [3|1|X] → [7|0]

$A = 5x^3 + 5x^2 + 3x + 7$

Date / /
Page No.

Subtraction of two polynomials:

Compare the exponents of two polynomials starting from the first node

(a) If both exponent of two polynomial then subtract the coefficient and stored it in the resultant linked list.

(b) If the exponent of the current term in the first polynomial P is less than the exponent of the current term of the 2nd polynomial Q. Then the current term of the second polynomial is added to the resultant linked list and move the pointer Q to point to the next node in the 2nd polynomial.

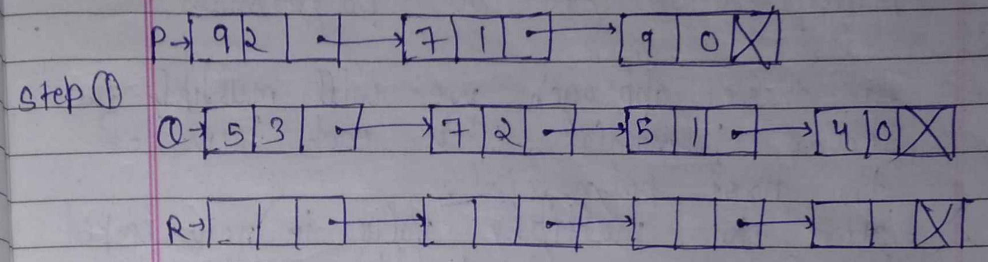
(c) Give the exponent of the term in the first polynomial is greater than exponent of the current term the second polynomial Q. Then the current term of the first polynomial resultant linked list and move the pointer P to the next node.

(d) Up and the remaining nodes of either of the polynomials to the resultant linklist.

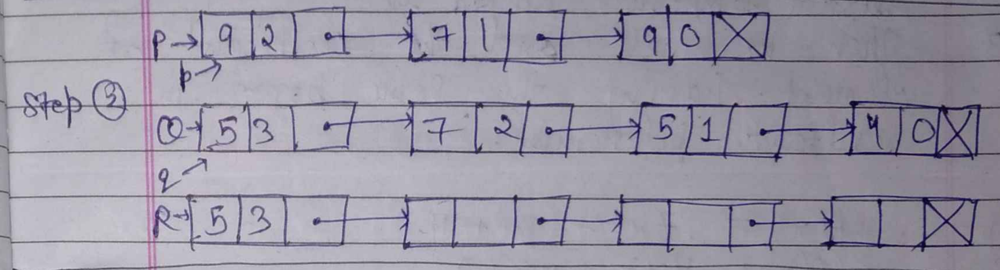
Date / /
Page No.

Exp: $P(x) = 9x^2 + 7x + 9$
$Q(x) = 5x^3 + 7x^2 + 5x + 4$

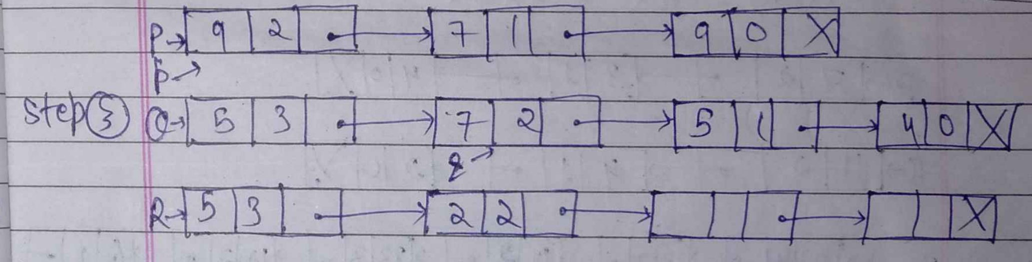
Step ①



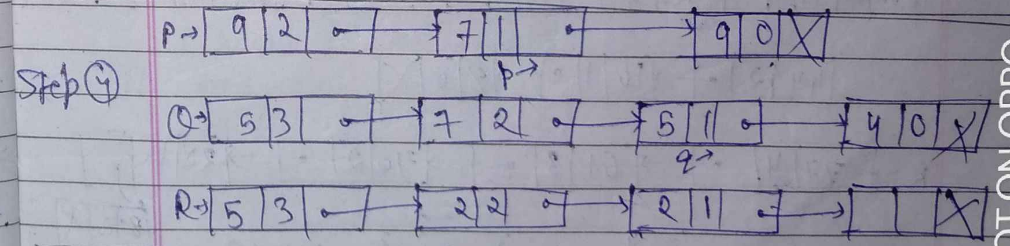
Step ②



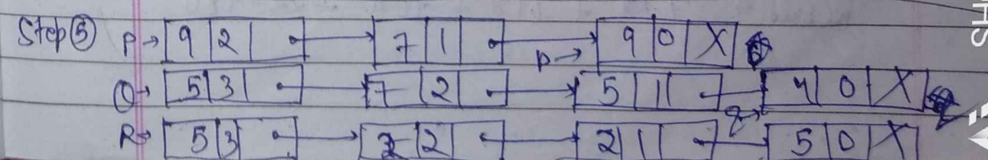
Step ③



Step ④



Step ⑤



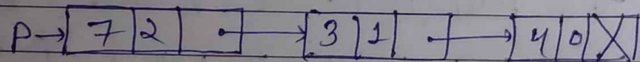
SHOT ON OPPO
By Ghanendra Kumar

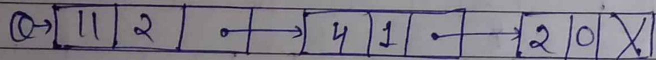
Result:- $5x^3 + 2x^2 + 2x + 5$

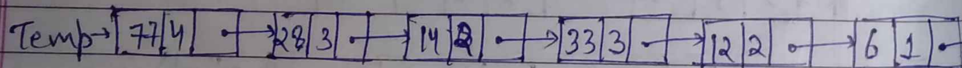
★ Multiplication of two polynomials:-

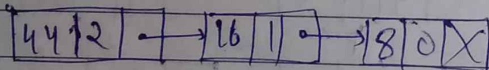
1. In these approach we will multiply the second polynomial with each term of the first polynomial.
2. Store the multiply value in new linked list (temperary linked list).
3. Then we will and the coefficient of element having the same power in resultant polynomial.

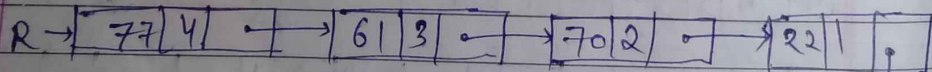
exp:-
$P(x) = 7x^2 + 3x + 4$
$Q(x) = 11x^2 + 4x + 2$











Result → $77x^4 + 61x^3 + 70x^2 + 22x + 80$

Unit - 5

Tree

→ Tree is non-primitive, non-linear data structure in which items are arranged in a sorted sequence.

→ It is used to represent hierarchical relationship among several data items.

→ There is no more than one edge b/w any pair of nodes.

→ A Tree contains no loops and no cycles.

→ If A tree contains 'n' nodes than it contains 'n-1' edges.

→ It is a finite set of 1 or more data items (nodes). such that

a) There is a special data item called the root of the tree.

b) And its remaining data items are partitioned into no. of disjoint subsets, Each of which is itself a tree and they are called sub-trees.

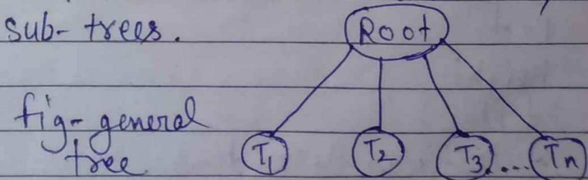
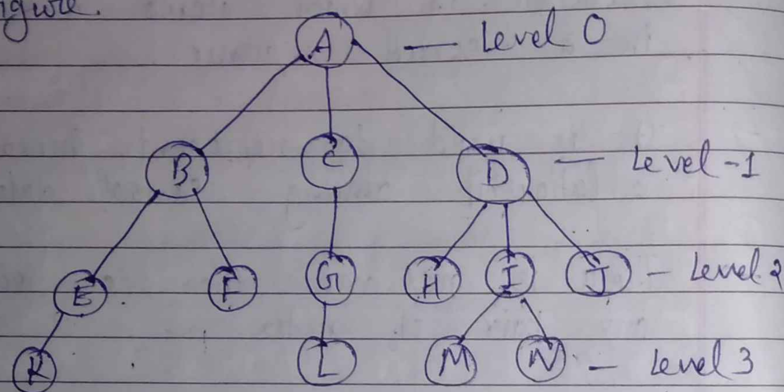


fig- general tree

SHOT ON OPPO
By Ghanendra Kumar

☆ Basic Tree terminology:-

There are no. of terms associated with the tree which can be explained with the help of the tree which shown in figure.



→ Node
→ Root
→ Parent
→ Child
→ Siblings
→ Degree of node
→ Degree of tree
→ Terminal node
→ Non-Terminal node
→ Level
→ Edge
→ Path
→ Height
→ depth
→ Ancestor or descendent
→ Forest

1. Node:- Each element of a tree is called Node. It is a basic structure in a tree. It specify the info. and links to other data items.
   There are 14 nodes in this tree

2. Root:- It is the first node in the hierarchical arrangement of data items. In this tree 'A' is the root node.

3. Parent:- Parent of a Node is the 'Immediate predecessor' of a node.
   Exp:- 'G' is the parent of L

4. Child:- Each immediate successor of a node is known as child.
   Exp:- 'L' is the child of G.

5. Siblings:- The child nodes of a given parent node are called siblings.
   Exp:- 'E and F' are the siblings. ~~of B~~

6. Degree of node:- The no. of sub trees of a node in a given tree is called degree of that node.
   Exp:- The degree of node 'D' is 3.

7. Degree of tree:- The maximum degree of any node in a given tree

is called the degree of the tree.
In the given tree the maximum degree of node are A & D. So degree of tree is 3.

8. Terminal nodes:- A node with degree '0' is called a terminal node or a leaf.
Exp:- K, F, L, H, M, N, J are the terminal nodes.

9. Non-Terminal nodes:- Any node (except the root node) whose degree is not zero called Non-terminal node.
Exp:- B, E, C, D, G, and I are the Non terminal node.

10. Level:- In general if a node at level 'n' then its children will be at level 'n+1'.
In the given tree there are four levels 0, 1, 2, and 3.

11. Edge:- Edge is a connecting lines of two nodes.
exp:- AB, AC, AD are the edges.

12. Path:-
from the source node to the destination node. In the given tree the path b/w A and M is given by nodes pair or edges, { {A, D}, {D, I}, {I, M} }

13. Depth:- The depth of node is a length of the unique path from the root to node. The root is at depth '0' in the given tree, the depth of node E is 'two'. And Depth of B is 'one'.

14. Height:- The height of a node is the length of the longest path from leaf to a node 'ni'. Thus all leafs are at height '0'. The height of the tree is equal to the height of the root. In the given tree the height of node B is 'two'. And the height of node D is 'two'.

15. Ancestor and descendent:- If there is a path from node n_1 to n_2 than n_1 is an Ansestor of n_2. And n_2 is a descendent of n_1.
Exp:- In the Given tree L is the descendent of G and G is the Ancestor of L.

16. Forest:- It is a set of disjoint trees, In a given tree if you remove its root then it becomes a forest. In the given tree there is a forest with three trees.

Date / /
Page No.

★ Binary tree :- A tree in which every node can have maximum of two children is called as binary tree.
In a binary tree every node can have either '0' children or 1 or two children But not more than 2 children.
Exp ÷

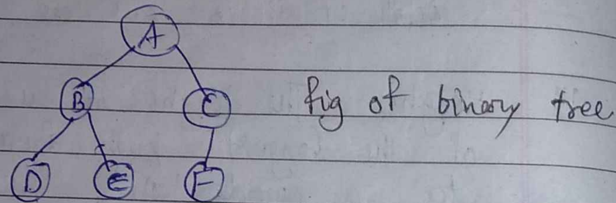


fig of binary tree

There are different types of binary trees which are as follows :

① Strictly Binary Tree :- A binary tree in which every node has either '0' or '2' no. of children is called strictly binary tree.
Other names of strictly Binary trees are 'full Binary tree' or 'proper binary tree' or '2-Tree'
Exp ÷

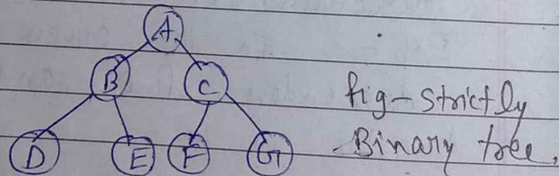
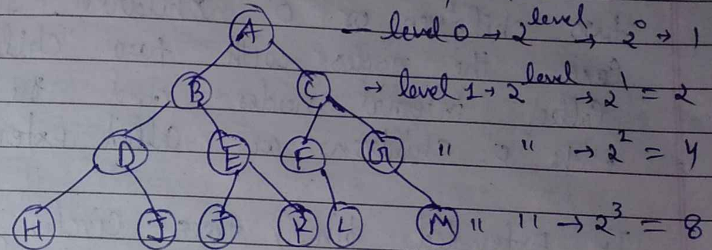


fig - strictly Binary tree.

② Complete Binary Tree :- A binary tree in which every internal node has exactly '2' children and all leaf node are at same level is called Complete binary tree.

Date / /
Page No.

Every level of complete binary tree there must be (2^{level}) no. of node, It is also known as perfect binary tree
Exp ÷



level 0 → 2^{level} → 2^0 → 1
→ level 1 → 2^{level} → 2^1 = 2
" " → 2^2 = 4
" " → 2^3 = 8

fig - Complete Binary tree.

Total no. of nodes in a complete binary tree of height 'H' $= 2^0 + 2^1 + 2^2 + 2^3 + \dots + 2^h$

$= 1 + [2^1 + 2^2 + 2^3 + \dots + 2^h]$

$= 1 + \left[\frac{2 \cdot (2^h - 1)}{2 - 1}\right]$ $\quad \frac{r^n}{r-1}$

$= 1 + 2^{h+1} - 2$

$n = 2^{h+1} - 1$

$n + 1 = 2^{h+1}$

$\log(n+1) = (h+1)\log 2$

$(h+1) = \frac{\log(n+1)}{\log 2}$

$h + 1 = \log_2(n+1)$

$\boxed{h = \log_2(n+1) - 1}$

SHOT ON OPPO
By Ghanendra Kumar

Date 10/10/22

③ ~~External~~ Extended Binary Tree:-

A binary tree (T) is said to be extended binary tree if each node 'n' has either two children or '0' children. In such a case the nodes with two children are called internal nodes and the nodes with '0' children are called external nodes.

In Extended binary trees circles for internal nodes and squares for External nodes. It is obtained by adding dummy nodes to a binary Tree.

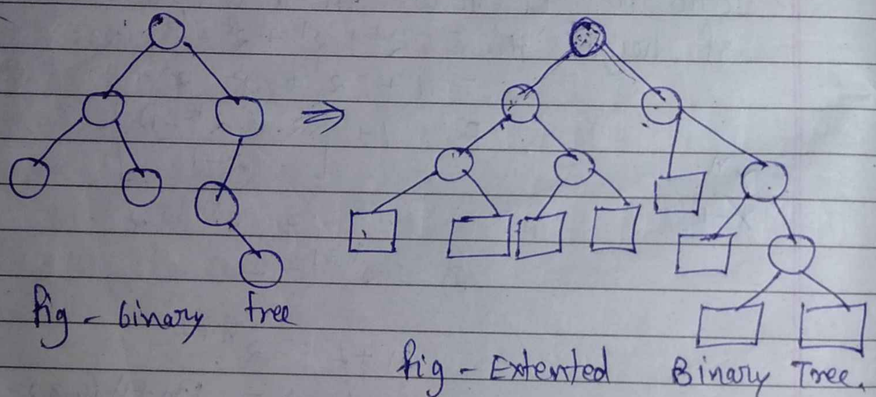


fig - Binary tree

fig - Extented Binary Tree.

★ Binary Tree Representation:-

There are two common methods for representing a binary tree.

① Linear (or sequential) rep^n using Array
② Linked list Rep^n using pointer.

1. Linear rep^n using Array:-

This rep^n uses only a single linear Array tree[] as follows:

1) The root of the tree is stored in tree[0].
2) If a node occupies tree[i] then its left child is stored in tree[2*i+1], its right chilid is stored in tree[2*i+2]. And the parent is stored in tree[(i-1)/2].

exp:-

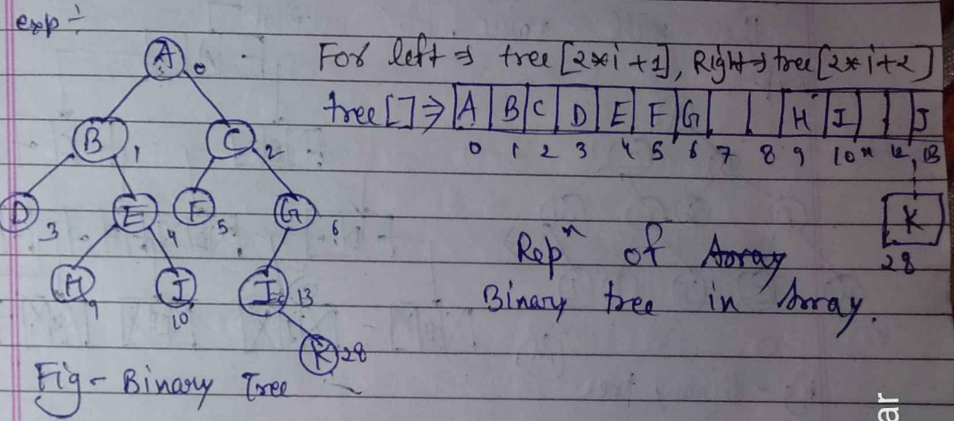


Fig - Binary Tree

Rep^n of ~~Array~~ Binary tree in Array.

2. Linked List Representation of Binary tree using pointer:-

In the linked list Representation of binary tree, a node is divided into three field Left, Info, Right,

Left	Info	Right

1) Left:-
Left pointer field which is used to store the address of the left child.

SHOT ON OPPO
By Ghanendra Kumar

2. Info:- Info is used to store the data item.

3. Right:- Right pointer field which is used to store the address of the right child.

Root:- It will contain the location of the root node. If any sub-tree is empty then the corresponding pointer will contain the null value. If the tree itself is empty then the root will contain the null value.

exp:-

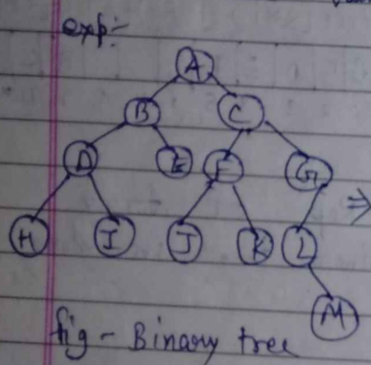


fig - Binary tree

⇒

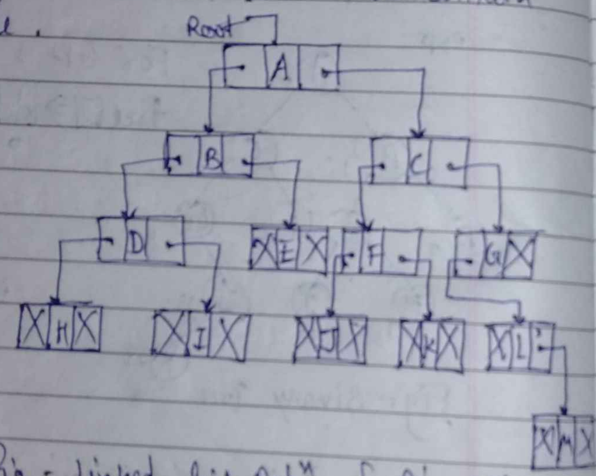


fig - Linked list Repⁿ of Binary Tree.

Imp

Traversal of a Binary tree:-

Tree traversal is one of the most common common operation performed on tree data structure. It is a way in which each node in the tree is visited exactly once in a systematic manner.

There are three popular ways of binary tree traversal. They are:-

1. Preorder Traversal (NLR)
2. Inorder Traversal (LNR)
3. Postorder Travesal (LRN)

Exp:-

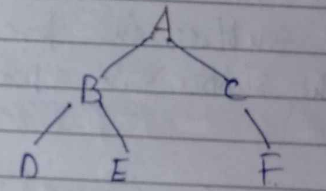


fig. of binary tree

(i) Preorder Traversal (NLR)
ABDECF

(ii) Inorder Traversal (LNR)
DBEAFC

(iii) Postorder Traversal (LRN)
~~DBECFA~~ ~~DEBCFA~~ DEBFCA

SHOT ON OPPO
By Ghanendra Kumar

exp:-

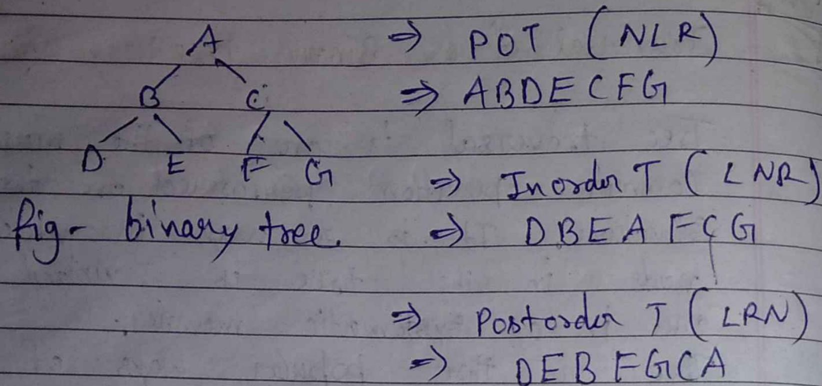


fig- binary tree.

⇒ POT (NLR)
⇒ ABDECFG

⇒ Inorder T (LNR)
⇒ DBEAFCG

⇒ Postorder T (LRN)
⇒ DEBFGCA

(1) Preorder Traversal (NLR) :-

The preorder traversal of a non-empty binary tree is defined as

Algorithm:-

(a) visit the root node (N)
(b) Traverse the left subtree in pre order
(c) Traverse the right subtree in pre order.

C- function :-

(a) For Preorder Traversal (NLR) :-

```
void preorder (node * P)
  {
    if (tree != NULL)
    {
      printf ("%d\n", P→num);
      preorder (P→ left);
      preorder ( P→ Right);
    }
  }
```

(b) For Inorder Traversal (LNR) :-

```
void Inorder (node * P)
  {
    if (tree != NULL)
    {
      Inorder (P → left);
      Printf ("%d\n", P→num);
      Inorder (P→ Right);
    }
  }
```

(c) For Post order Traversal (LRN) :-

```
void Postorder (node * P)
  {
    if (tree != NULL)
    {
      Postorder (P→ Left);
      Postorder ( P → Right);
      Printf ("%d\n", P→nu
    }
  }
```

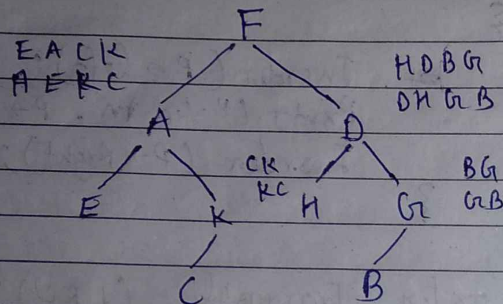
SHOT ON OPPO
By Ghanendra Kumar

Pre/Post Main see
In main do

Date / /
Page No.

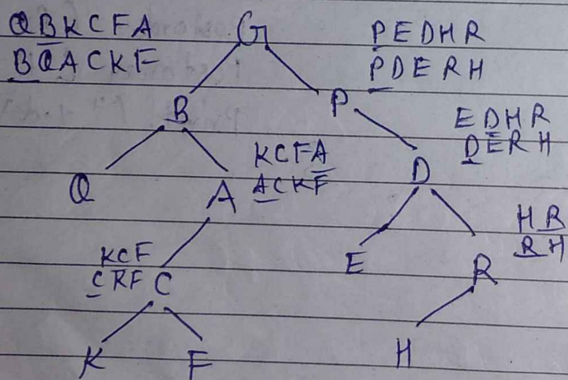
Q. Construct the binary tree for the following

Inorder :- E A C K F H D B G
Preorder :- F A E K C D H G B



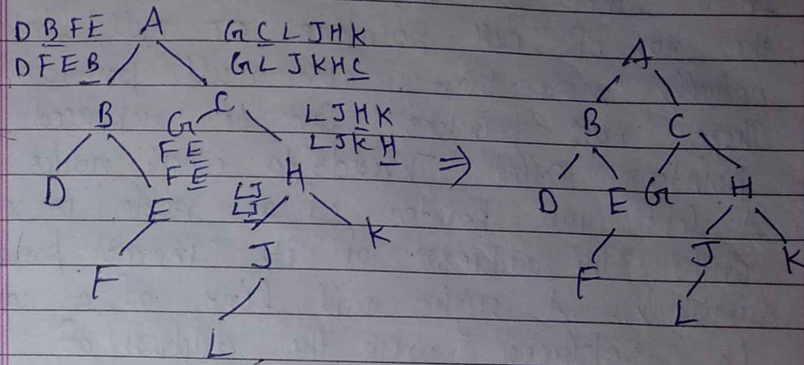
Q.

Inorder :- Q B K C F A G P E D H R
Preorder :- G B Q A C K F P D E R H

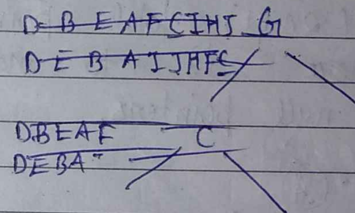


Date / /
Page No.

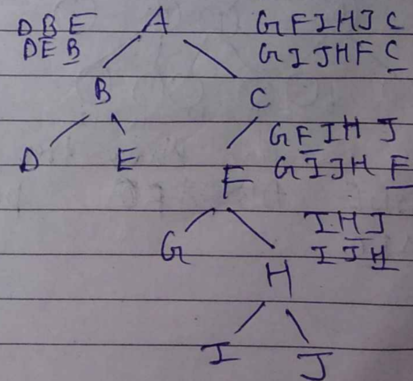
Q. Inorder :- D B F E A G C L J H K
Postorder :- D F E B G L J K H C A



Q. Inorder :- D B E A F G I H J G C
Postorder :- D E B A I J H F C G



Q. In :- D B E A G F I H J C
Post :- D E B G I J H F C A



SHOT ON OPPO
By Ghanendra Kumar

Thread :- light weight process.

☆ Threaded Binary tree :-

In a linked list representation of a binary tree, the no. of null pointers or null links are actually more than non-null pointers.
These null pointers can be replaced by pointers called Threads to other nodes.
A left null pointer of a node is replaced with the address of its inorder predecessor.
Similarly, A right null link of a node is replaced with the address of its inorder successor.
We can generalize it for any binary tree with n nodes there will be (n+1) null pointers. And (2n+1) total pointer the objective here is to make effective use of these null pointers.

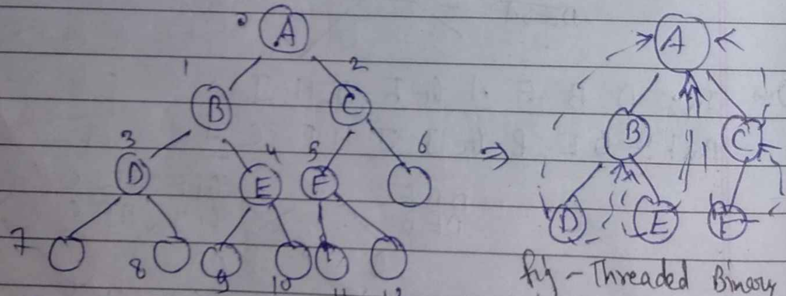


fig - Binary tree with null pointer

fig - Threaded Binary tree.

Inorder
~~ABDE~~ ~~DBEAFC~~
DBEAFC

☆ Linked list Repⁿ of Threaded Binary Tree :-

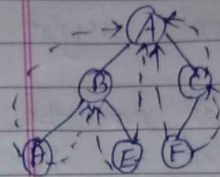


fig :- Threaded binary tree

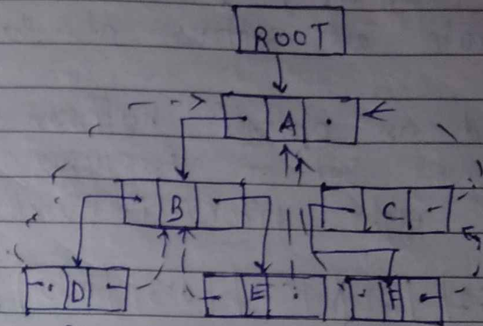
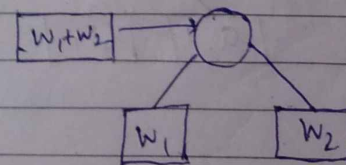


fig - linked list Repⁿ of Threaded Binary tree

☆ Huffman Algorithm :-

It is a method for building and extended binary tree with a minimum weighted path length from a set of given weight.
Huffman coding that make is based on variable length code size.
Huffman codes are binary used and very effective technique for compressing the data.

(i) Suppose there are 'n' weights $w_1, w_2, w_3 \ldots w_n$
(ii) Taking two minimum weights, among the 'n' given weight. Suppose w_1 and w_2 are first two minimum weight then sub tree will be



SHOT ON OPPO
By Ghanendra Kumar

(iii) Now the remaining weights will we
$w_1 + w_2, w_3, w_4, \ldots w_n$.

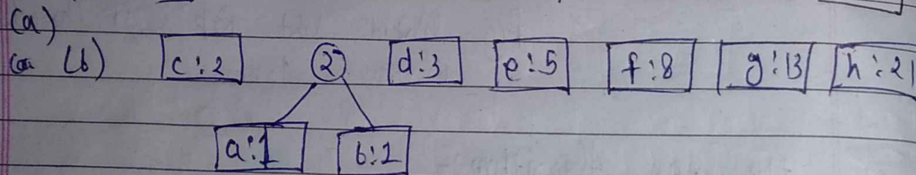
(iv) Create all subtrees at the least weight.

Q. Find an optimal huffman code for the following set of frequency based on the 1st eight fibonacci numbers.

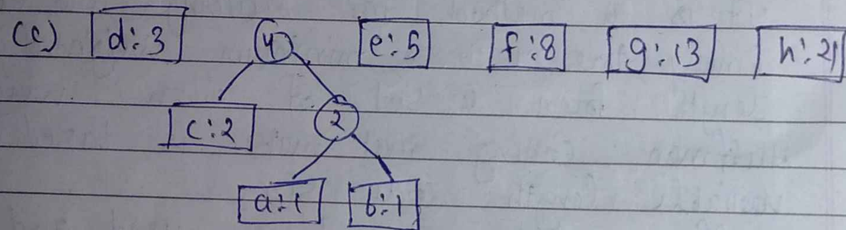
a:1 b:1 c:2 d:3 e:5 f:8 g:13 h:21

(a)

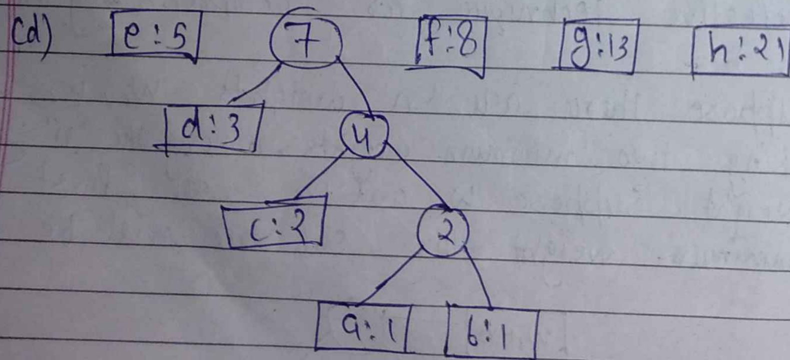
(b)



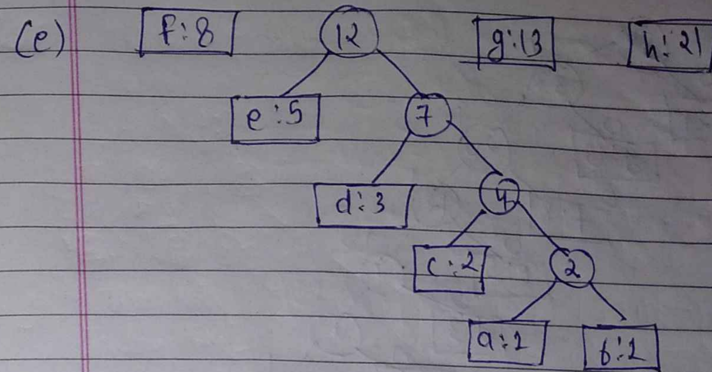
(c)



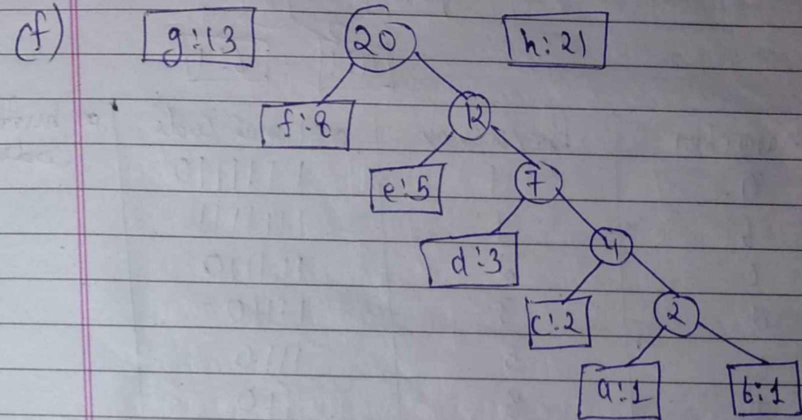
(d)



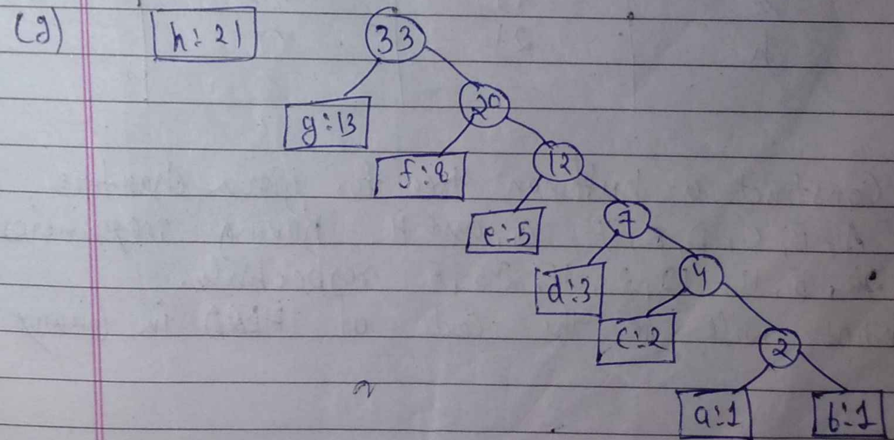
(e)



(f)

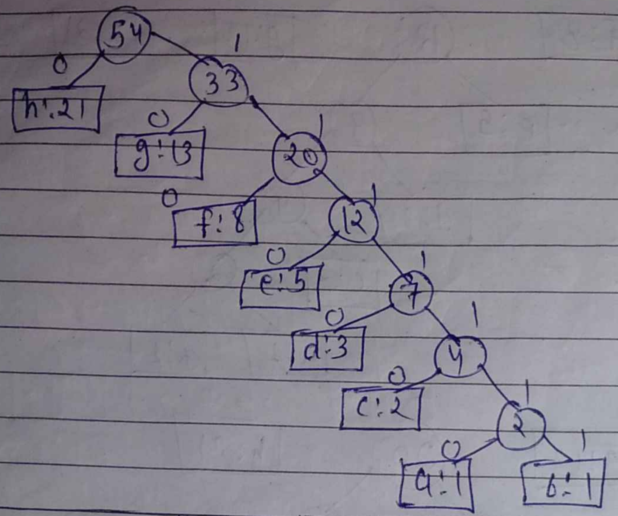


(g)



SHOT ON OPPO
By Ghanendra Kumar

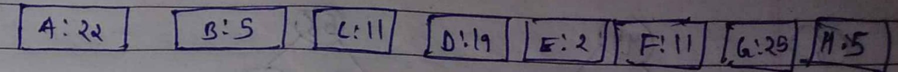
(h)

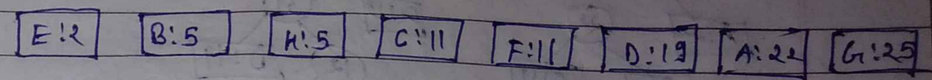


Character	Frequency	optimal Code	or huffman Code
a	1	1111110	
b	1	1111111	
c	2	111110	
d	3	11110	
e	5	1110	
f	8	110	
g	13	10	
h	21	0	

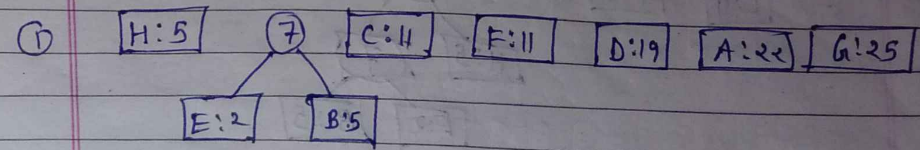
Q. Construct a huffman tree for given charters A, B, C, D, E, F, G, and H having frequencies 22, 5, 11, 19, 2, 11, 25, 5 respectively. What will be the code of HEAD in binary

Sol:

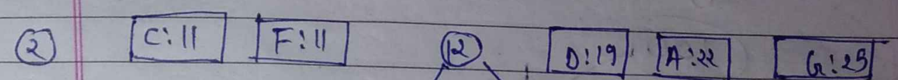


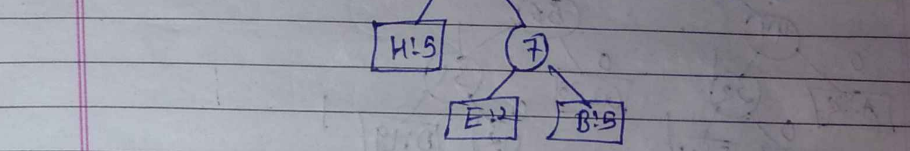


(1)

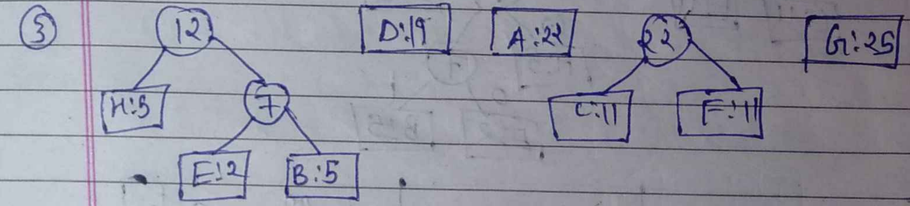


(2)

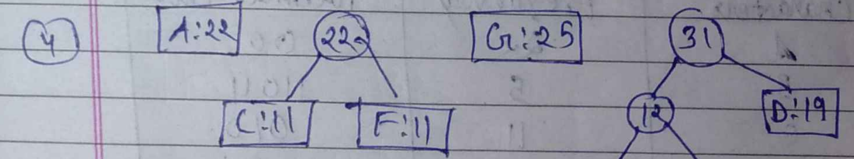


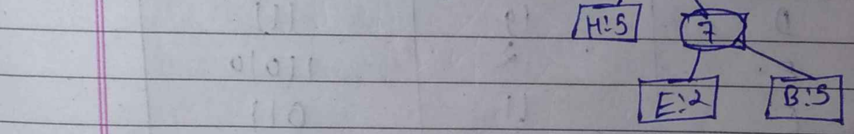


(3)

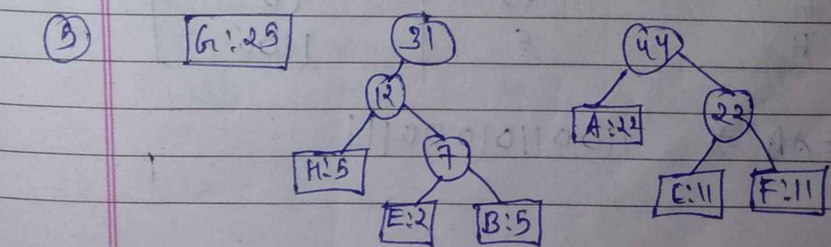


(4)



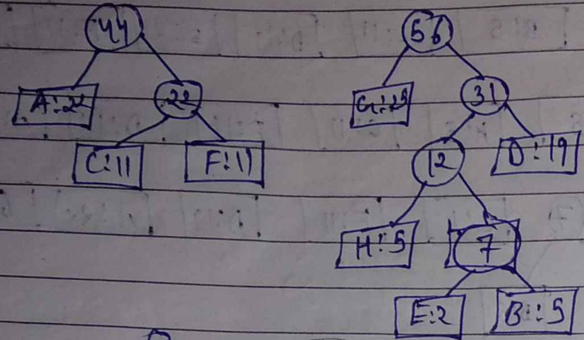


(5)

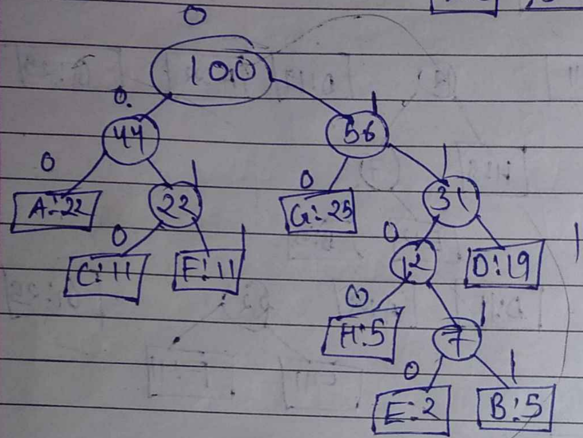


Date / /
Page No.

(6)



(7)



Characters	Frequency	Huffman Code
A	22	00
B	5	1101
C	11	010
D	19	111
E	2	11010
F	11	011
G	25	10
H	5	1100

HEAD ⇒ 1100110100 0111

Date / /
Page No.

Imp ☆

Binary Search Tree (BST) :-

A binary search tree is a binary tree, Such a tree can be represented by a linked list data structure in which each node is an object

→ Binary Search tree properties:-

A binary search tree may be empty. If it is not empty then it satisfies the following properties.

1) Every element has a key i.e key of (x) for element (x) and no two element have the same key. The keys in the left subtrees are smaller then the key in the Root.

2) The key in the right sub-trees are larger then the key in the Root.

3) The left and right Sub-tree are also binary search trees.

☆ Maximum and Minimum key element in (BST):

Tree Minimum :-

1) while left[x] ≠ NULL
2) do x ← left[x]
3) return x

SHOT ON OPPO
By Ghanendra Kumar

Date / /
Page No.

☆ Tree Maximum :-

1) while Right [x] ≠ NULL
2) do x ← Right [x]
3) return x.

☆ Searching in BST :-

TREE - SEARCH (x, k)

1) If x = NIL or k = key [x]
2) then return x
3) If k < key [x]
then return TREE - SEARCH (Left [x], k)
4) Else:
return TREE - SEARCH (Right [x], k)

☆ Insertion of data into a 'BST' :-

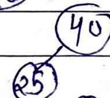
The sequence of steps to be followed in performing the insertion operation on a binary search tree are as follows:

1) Start from the root node
2) If the data to be inserted it is 'w', Compare this with the value of the root node
(i) If they are equal just stop because this value already exist.
(ii) If they are different and if the data to be inserted is less than the value of the root node choose the left subtree.
(iii) If the data to be inserted is greater than the value of the root node.
3) Repeat step ② until the leaf node is encountered where the data has to be inserted.

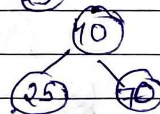
Q→ Insert the following numbers in empty BST 40, 25, 70, 22, 35, 60, 80, 90, 10, 30. Draw tree.

Sol→ 1) Insert 40 :

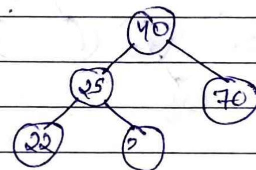
2) Insert 25 :



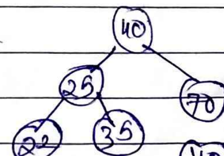
3) Insert 70 :



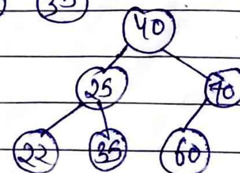
4) Insert 22 :



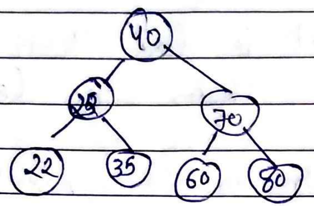
5) Insert 35 :



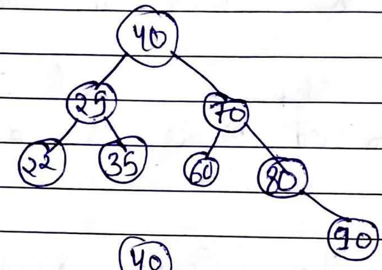
6) Insert 60 :



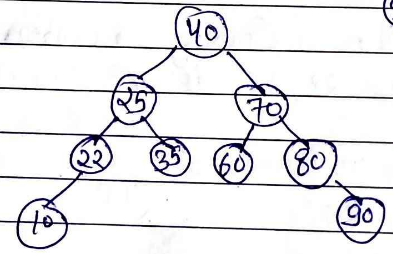
Insert 80:



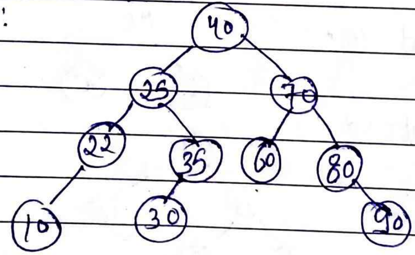
Insert 90:



Insert 10:



Insert 30:

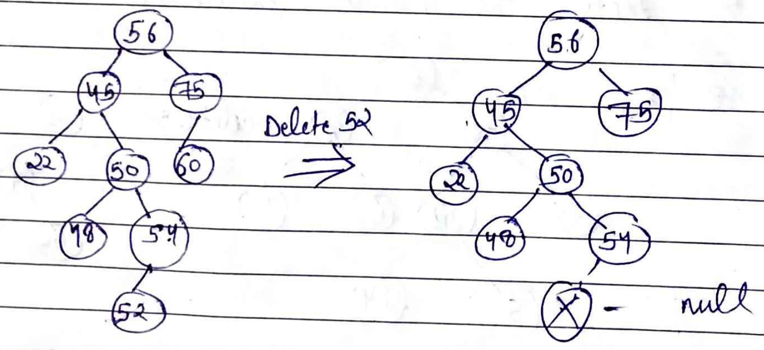


☆ Delete a node from a BST :-

Deletion of a node from a BST depends on the no. of its children.
Suppose delete a node with key = k from BST
There are three cases that can occur.

Case 1:- k has no child (i.e. leaf node).

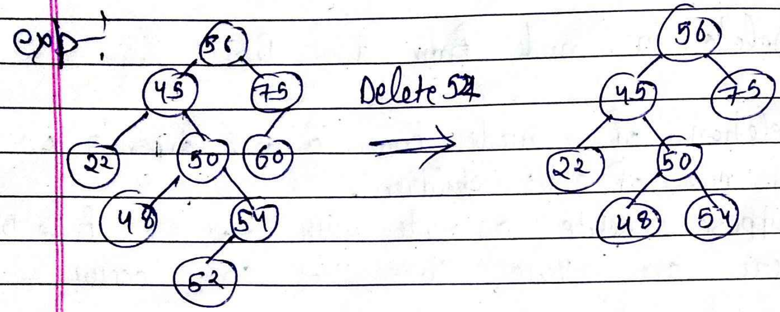
If the node to be deleted is the leaf node. simply make the pointer field of the parent node to null.



Case 2:- k has exactly 1 child

If the node to be deleted has 'one' child then simply set the pointer field of the parent node of k to point to the only child node of k.

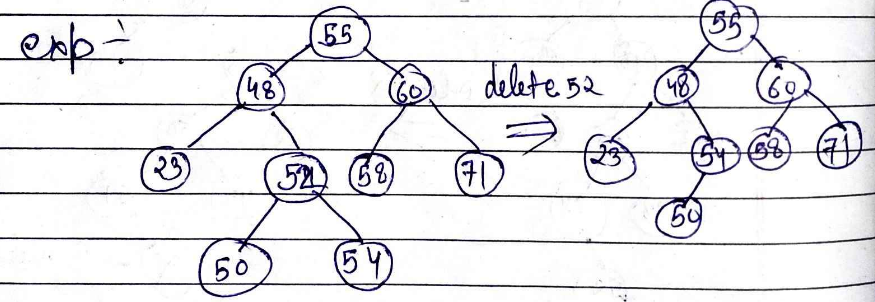
exp:-



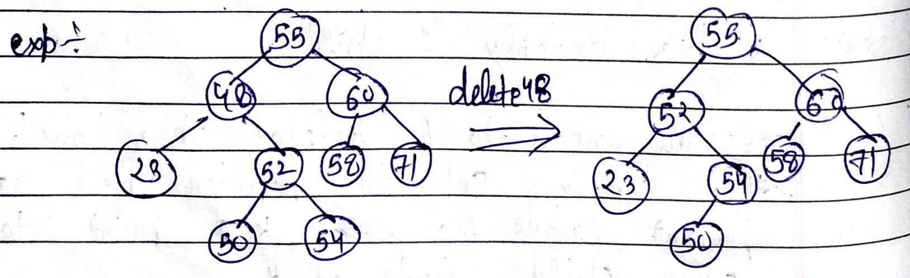
Case 3:- K has two children:-

If the node to be deleted K has two children Find the inorder successor on the node to be deleted K. place it in the node K then delete the inorder successor node

exp:-



exp:-



☆ B-Tree :-

A B-tree of order $M \geq 2$ is a M-way Search tree with the following properties.

1. The root can have 1 to M-1 keys
2. All nodes (except the root) have between $\lceil \frac{M}{2} \rceil$ and M children.

$$\text{Min. no. of keys} = \lceil \frac{M}{2} \rceil - 1$$

$$\text{Max no. of keys} = M-1$$

3. The tree is balanced i.e. All leafs are at the same depth, which is the tree's hieght.

If a node has 't' no. of children then it must have t-1 no. of keys

☆ For degree

There are lower and upper bound on the no. of keys a node can contain. These bounds are expressed in terms of a fixed integer $t \geq 2$ called the min. degree of the B-tree.

a) Lower bound:- Every node other than the root must have atleast t-1 keys. Every internal node other than the root has atleast 't' children. If the tree is non-empty, the root must have atleast 'one' key.

Sorted ⇒ ascending order

b) Upper bound :- Every node can contain atmost 2t-1 keys. Therefore an internal node can have atmost ~~two key~~ '2t' children. We say that a node is full if it contains exactly (2t-1) keys.

★ Insertion of a key into a B-Tree :-

To insert a value 'x' into a B-tree, there are three steps.

1) Find the correct leaf node to which x should be added.
2) Add 'x' to the node at the appropriate place. Sorted measure of keys inside the node should be preserved after insertion.
3) If there are (M-1) or fewer keys in the node after 'x' is added then the finish.

Q. Explain the steps to build a B-Tree of order 5 for the following data,
78, 21, 14, 11, 97, 85, 74, 63, 45, 42, 57, 20, 16, 19, 32, 30, 31

Sol: Insert 78 : [78]
Insert 21 : [21 78]
Insert 14 : [14 21 78]

Insert 11 : [11 14 21 78]

Insert 97 : [11 14 21 78 97] ⇒ overflow, split → root [21], children [11 14] [78 97]

Insert 85 : root [21], children [11 14] [78 85 97]

Insert 74 : root [21], children [11 14] [74 78 85 97]

Insert 63 : root [21], children [11 14] [63 74 78 85 97] ⇒ overflow, split

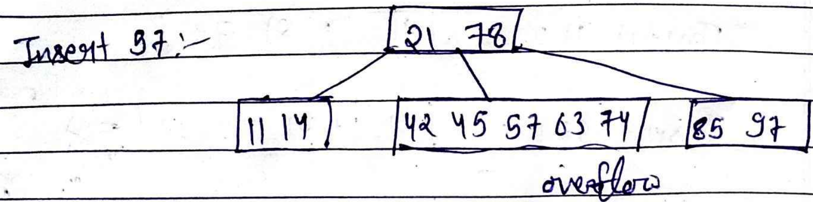
⇓

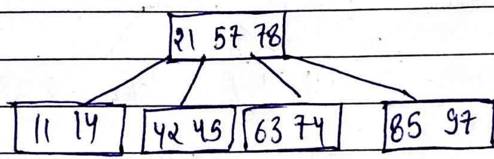
root [21 78], children [11 14] [63 74] [~~78~~ 97]

Insert 45 : root [21 78], children [11 14] [45 63 74] [85 97]

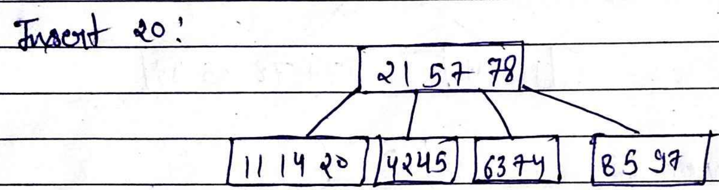
Insert 42 : root [21 78], children [11 14] [42 45 63 74] [85 97]

Insert 97 :-

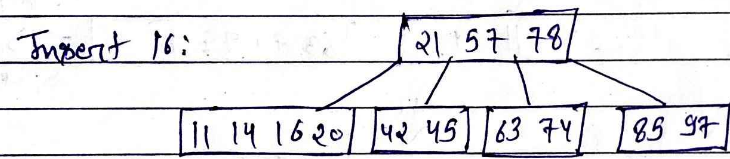




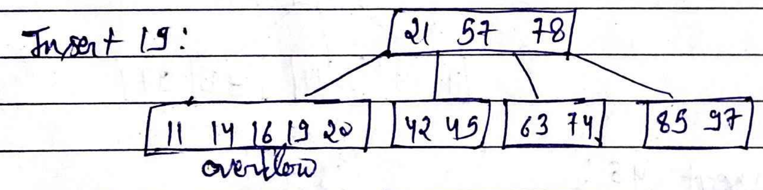
Insert 20 :



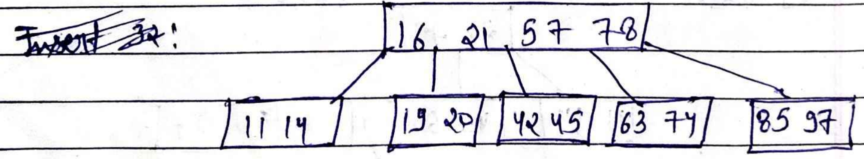
Insert 16 :



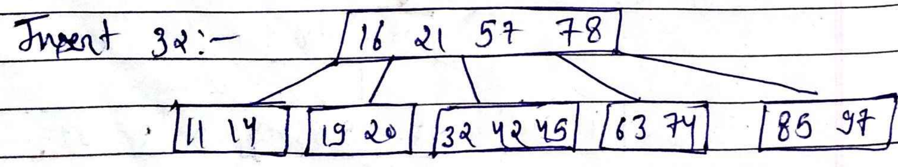
Insert 19 :



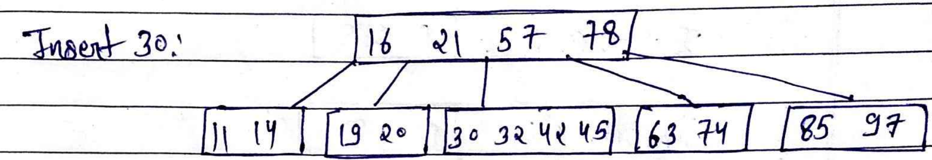
~~Insert 32~~ :



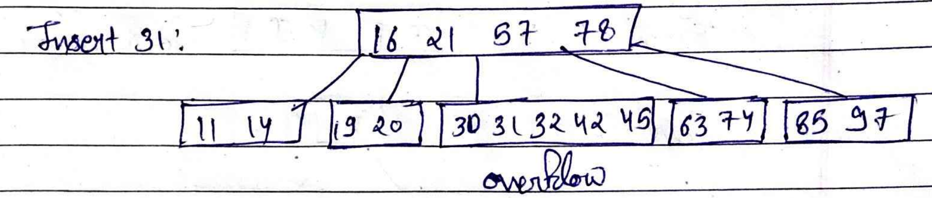
Insert 32 :-

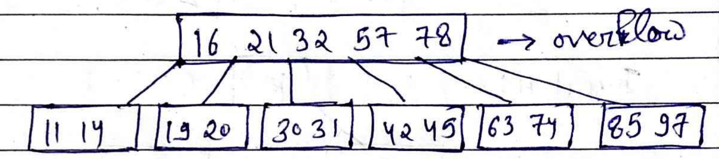


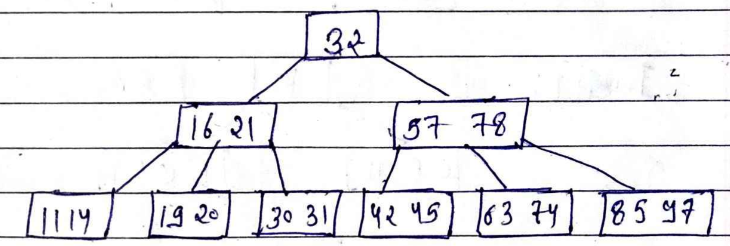
Insert 30 :



Insert 31 :







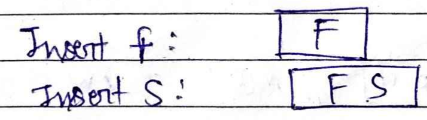
Q. Inserting the keys F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B, X, Y, D, Z, E, G, I, Inorder into an empty B Tree only use $t = 3$ (degree).

Sol. $t = 3$ (degree)

Max no. of keys $= 2t - 1 = 2 \times 3 - 1 = 5$

Insert F :

Insert S :

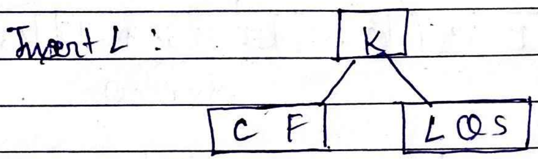


Insert Q : FQS

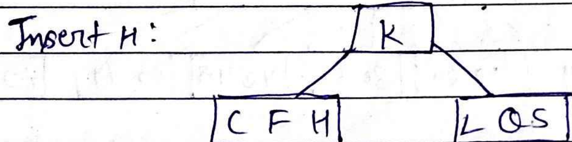
Insert K : FKQS

Insert C : CF KQS overflow

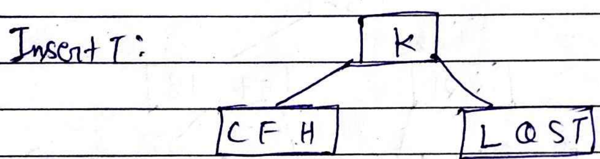
Insert L :



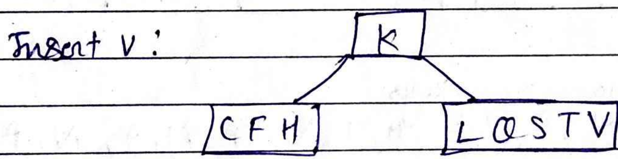
Insert H :



Insert T :

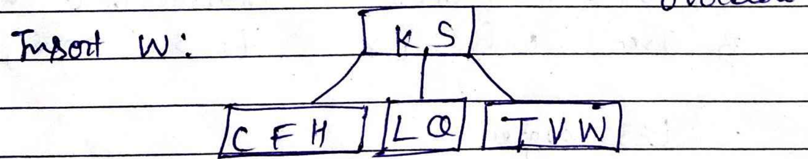


Insert V :

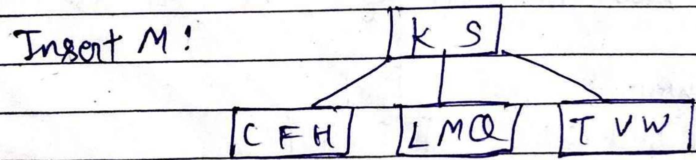


overflow

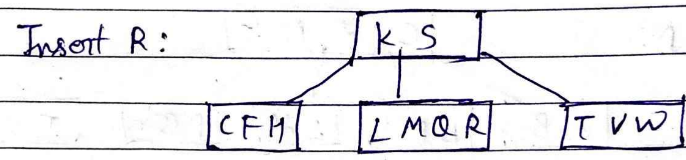
Insert W :



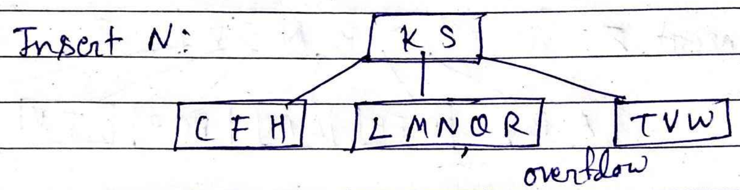
Insert M :



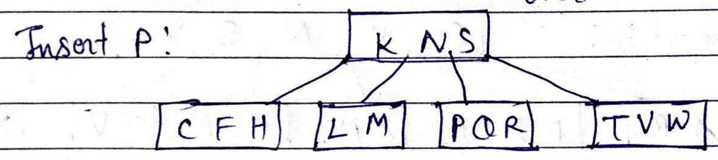
Insert R :



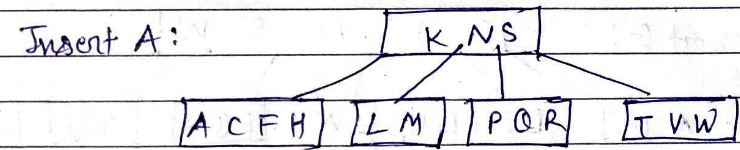
Insert N :



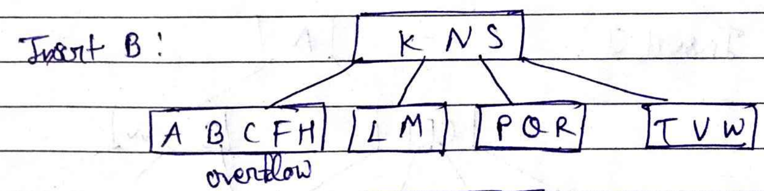
Insert P :



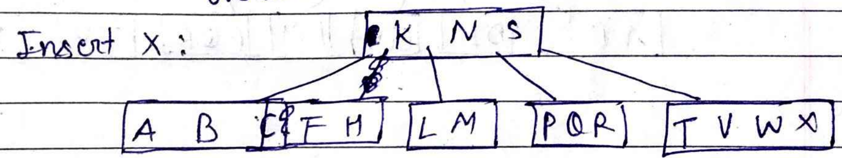
Insert A :



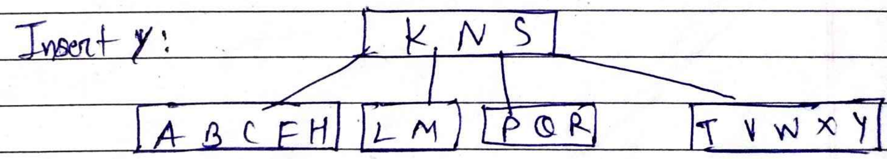
Insert B :



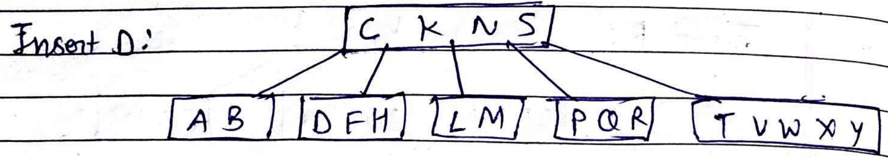
Insert X :



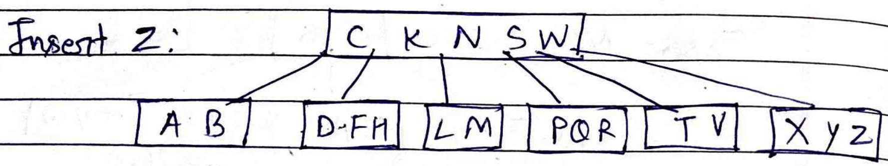
Insert Y :



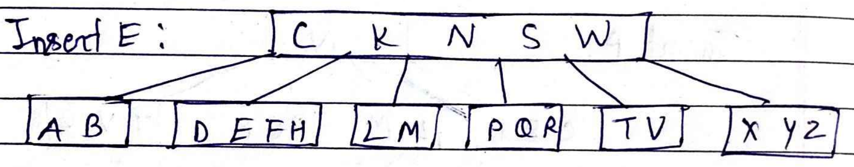
Insert D:



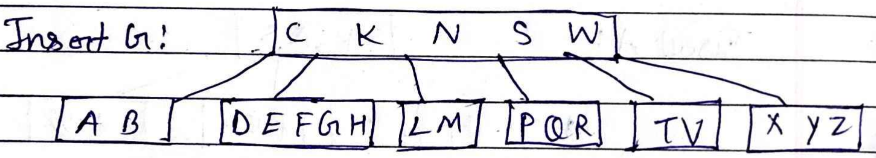
Insert Z:



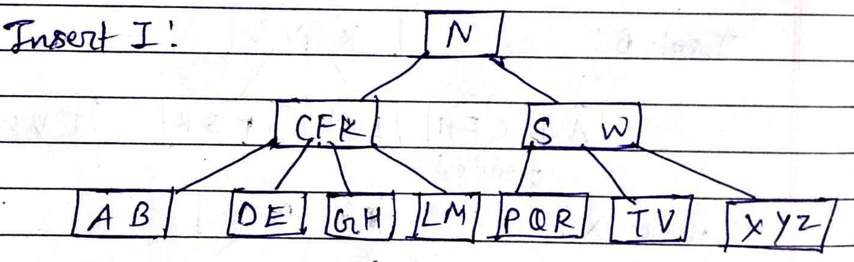
Insert E:



Insert G:



Insert I:



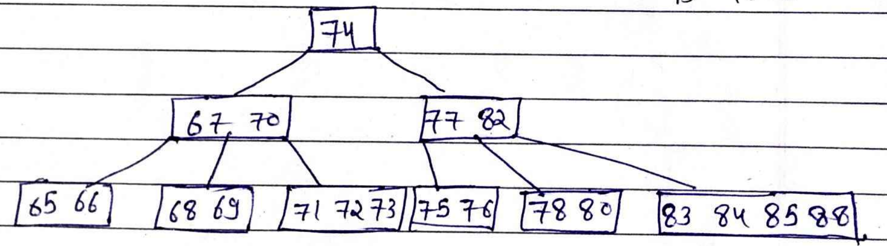
(A) Deletion from a leaf node:-

If the node to be deleted is on leaf then it is simply deleted.

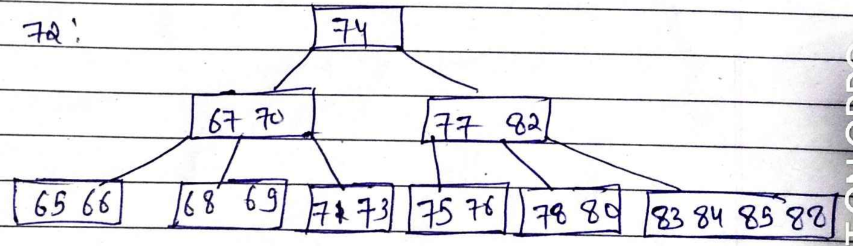
☆ Deletion from a non-leaf node:-

If it is not a leaf node, it must be replaced by its inorder successor or predecessor

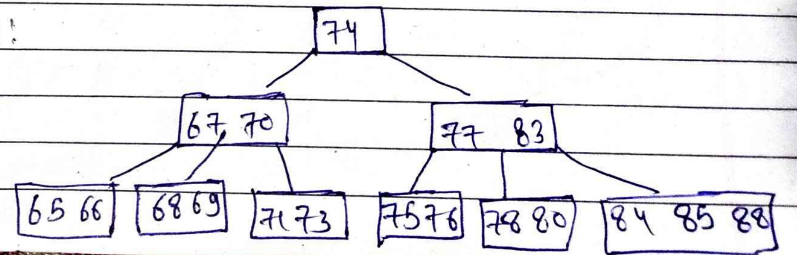
Q→ Delete these node 72, 82, 80 and 68 from the following B-Tree.



Sol: Delete 72:

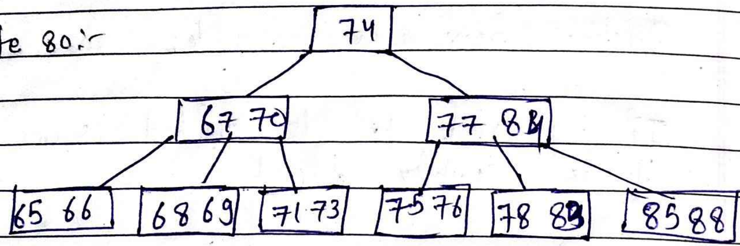


Delete 82:

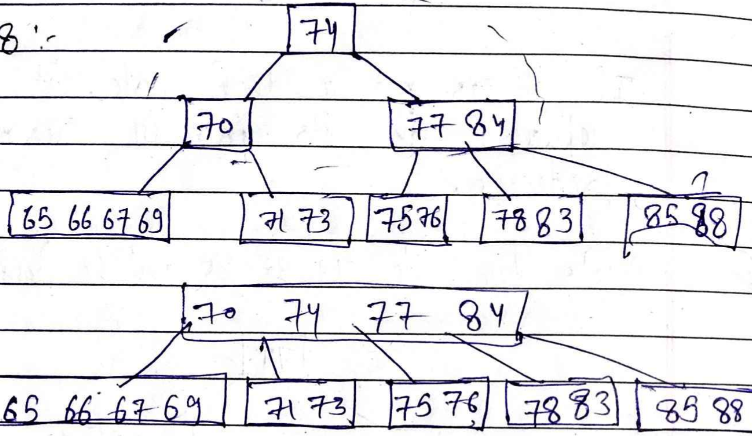


SHOT ON OPPO

Delete 80 :-



Delete 68 :-



☆ AVL tree :-

An AVL tree is a hieght valence tree. These trees are binary search trees in which the height of two siblings are not permitted to differ by more than 1.

| Height of left subtree − Height of right subtree | ≤ 1

$$|H_L - H_R| \leq 1$$

Defination :- An empty tree is height balanced is 'T' if 'T' is a non-empty binary tree with T_L and T_R as its left and right sub-trees. Then T is height balanced if and only if

1) T_L and T_R are height balanced
2) $|H_L - H_R| \leq 1$

Balance factor (BF) = BF(T) = $(H_L - H_R)$

For any node tree is an AVL tree if BF(T) = −1, 0 or 1

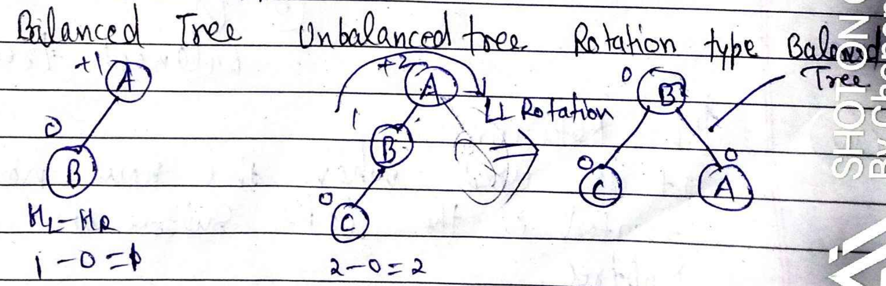
LL → Right rotate
RR → Left Rotation

There are four types of rotation :-

1. LL Rotation
2. RR "
3. LR "
4. RL "

① LL Rotation :-
It is used when the new node is inserted in the left subtree of left subtree of node 'A'.

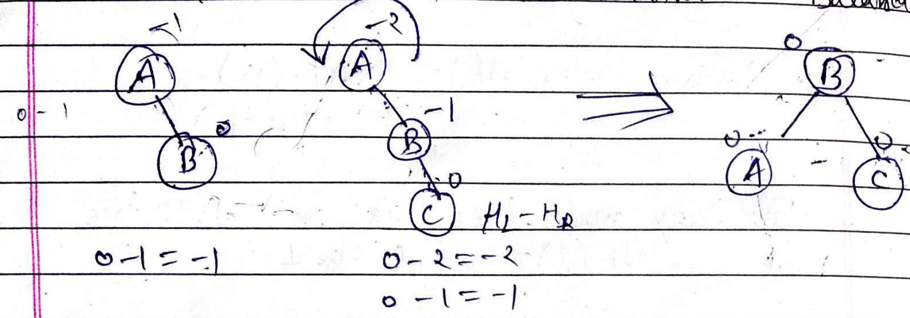
Balanced Tree | Unbalanced tree | Rotation type | Balanced Tree



(ii) RR - Rotation:-

When the new node is inserted in the right subtree of right subtree of node 'A'.

exp: Balanced Unbalanced RR-Rotation Balanced.



0-1=-1

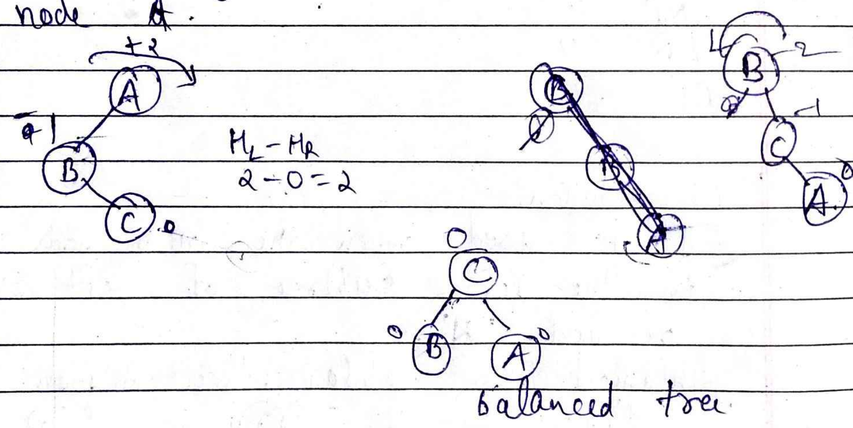
0-2=-2

0-1=-1

(iii) LR - Rotation:-

It is used when the new node is inserted in the right subtree of left subtree node A.

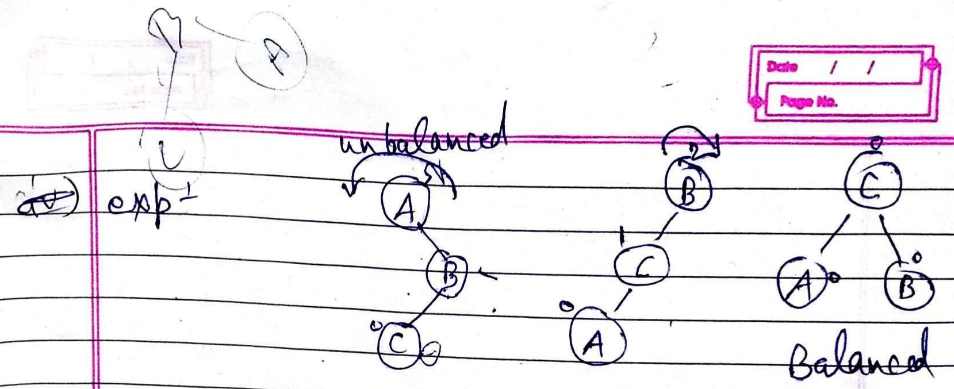
exp:

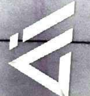


(iv) RL - Rotation:-

It is used when the new node is inserted in the left subtree of Right subtree.

(iv) exp:-





SHOT ON OPPO
By Ghanendra Kumar