* Data:-

⇒ Data is a collection of information
⇒ When data is Processed it becomes information

⇒ Data Can be number, Symbol, Character or any kind info.

⇒ The logical & Meaningful form of data is called information.

Two types of Data type

(1) Built in data type: int, float, Char, double which are defined by Programming language.

(2) User defined data type: User can be define his own data type.

eg:

```
typedef Struct Student
{
    int Roll;
    Char name;
}S;
S s1;
```

* Data object

→ is a set of elements such a Set may be finite or infinite.

Ex. Set of Students Studying in Second Year of Compeiter Eng. it is finite set,
Set of natural numbers is an infinite set.

* Data Structure:

→ The Combination which describe the set of elements together with the description of all legal operations is called data structure.

Ex:
The set of elements as integer, and the operations on integer are addition, subtraction, mul, division etc.

So, the data object - integer along with description of these

Def A da
domain
and Se
This trip
data Stru

* Abstract

The abst
Set of d
A - Axioms
to be e
how is

In ADT
dete

* Primitiv

→ Primitive
mental

int, f

Structure

Structure.

Def A data Structure is a Set of domain D, a set of function F and Set of axioms A. This triple (D, F, A) denotes the data Structure d. D is Domain.

* Abstract Data Type:-

The abstract data type is triple of D- Set of domain, F = Set of fun^n A - axioms in which only what is to be done is mentioned but how is to be done is not mentioned.

In ADT, all implementation details are hidden.

* Primitive and Non Primitive

→ Primitive data types are the fundamental data types.

int, float, double, char.

Page
Date

→ Non Primitive data types can be built using Primitive data types.

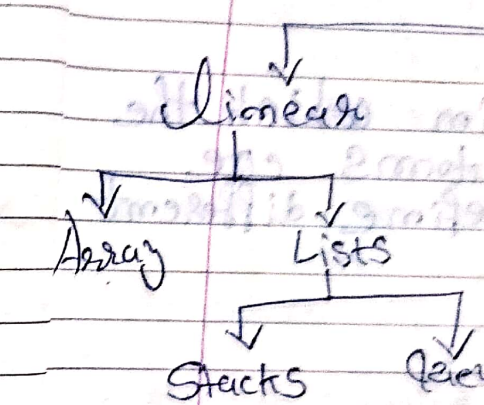
Ex:- Structure, Union.

Primitive Data Types

```
int count;
float average;
char choice;
```

Non Primitive Data Types

```
typedef struct student
{
    int roll_No;
    float marks;
};
```



→ Data St
modellin

→ We can
the data
Problem.

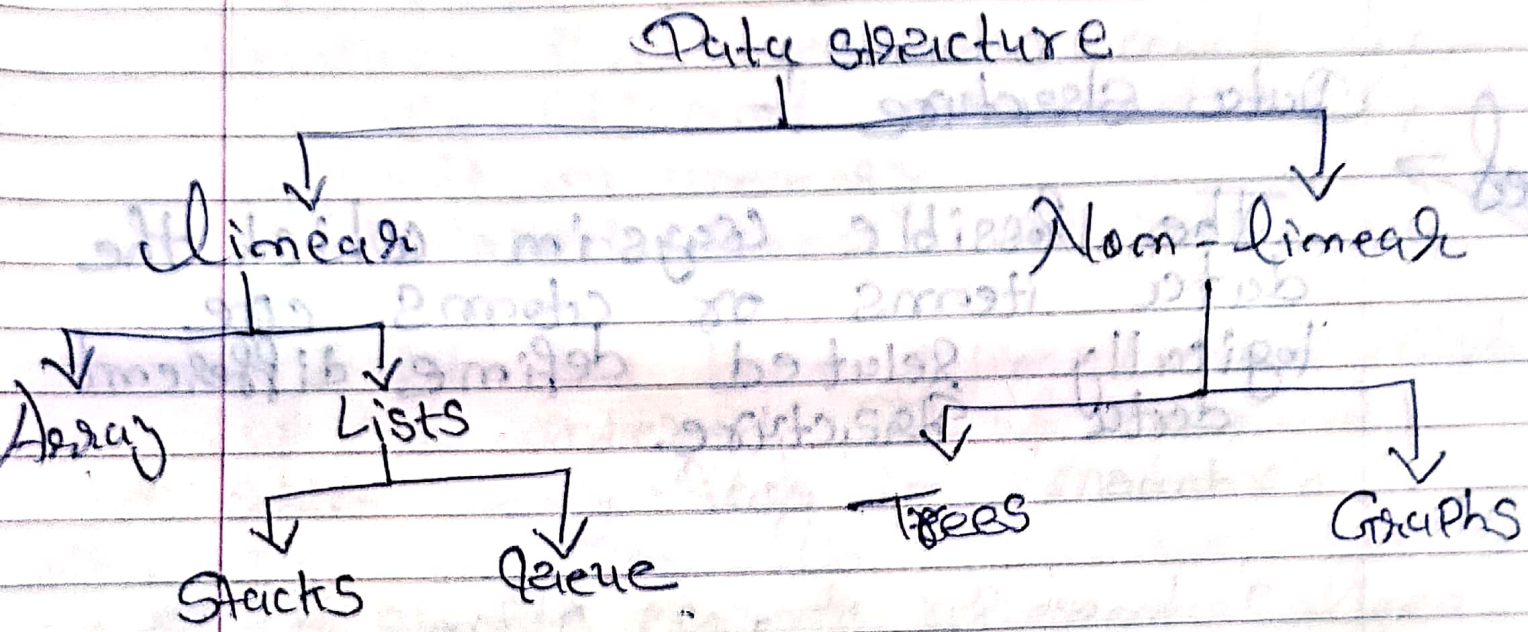
* Linear data

Structure
arranged in
sequence

* Non-linear

those in whi

* Types of Data Structure



→ Data Structure is used for modelling the Problem

→ We can Apply some algorithm on the data structre to Solve Particular Problem.

* Linear data Structures are the data Structure in which data may be arranged in a list or in a straight sequence Array, List

* Non-linear data Structure are those in which data may be arrange in

hierarchical manners.
trees, graphs

Data Structure :-

eg →

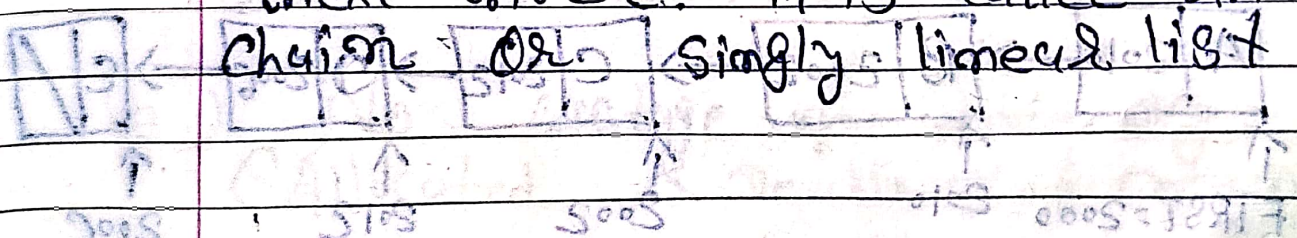
The possible ways in which the data items or atoms are logically related define different data Structure.

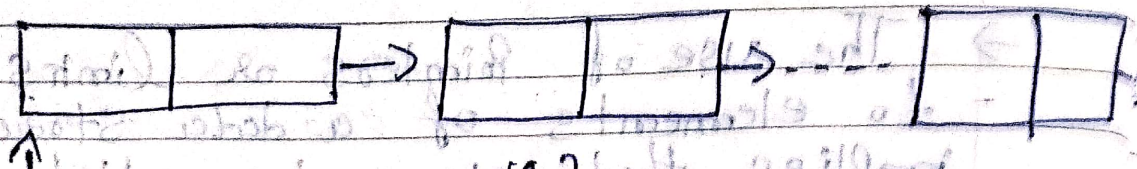
Link-List

→ The use of Pointers or links to refer to elements of a data structure implies that elements which are logically adjacent need not be physically adjacent in memory it is called linked allocation.

→ A list has been defined to consist of an ordered set of elements which may vary in number.

→ A simple way to represent a linear list to expand each node to contain a link or pointer to the next node. it is called one-way chain or singly linear list.



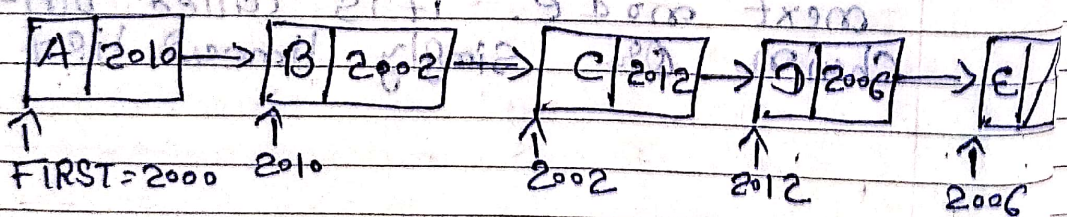


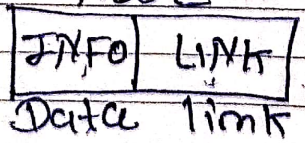
(a)

→ The variable FIRST contain an address or pointer which gives the location of first node of list.

→ Each node devided into two Part.

→ First Part represents the information of element, Second Part contain the address of the next node.





→ Available area of storage for this node structure consists of a linked stack of available node.

Def!-

→ Addres
is to be

The
nodes
two fi

* Advant

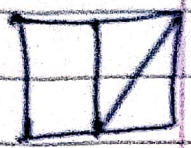
(1) Insert
elements

(2) No
Allocated

* Disadvan

(1) Does no
access

(2) Each
by link
node.



→ The Pointer Variable AVAIL Contains the address of the top node in the Stack.

→ Address of the next available node is to be Stored in the Variable NEXT.

Def:- The linked list is a collection of nodes. Each node consists of two field 'data' 'link'.

* **Advantages**

(1) Insertion and Deletion of elements can done efficiently.

(2) No wastage of memory. Allocated & Deallocated memory easily

* **Disadvantage**

(1) Does not Support Random or Direct access

(2) Each data field should be supported by link field to Point to next node.

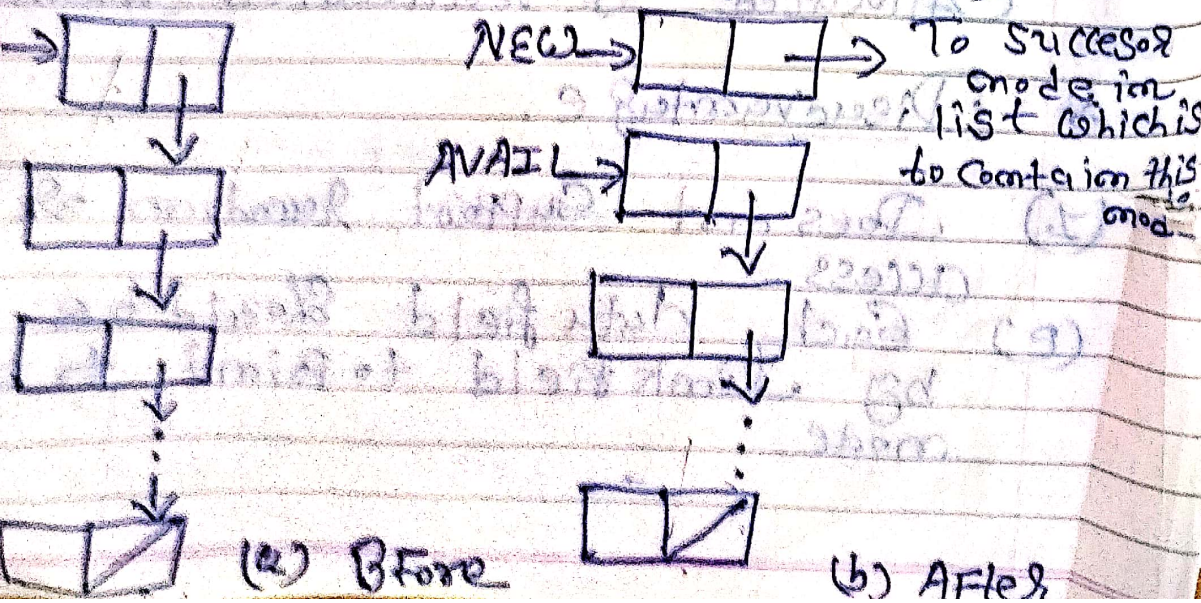
Simply linked list

Function INSERT (X, FIRST).

Given X, a new element
FIRST, a Pointer to the first element of a linked list
It contain INFO & link field.
element

AVAIL is a Pointer to the top elements of the availability Stack.

NEW is a temporary Pointer variable. (Availibility Stack)



* Insert the node at first Position

1. [Underflow?]
   if AVAIL = NULL
   then write ('Availibility stack Underflow')
   Return (FIRST)

2. [Obtain address of next free node]
   NEW ← AVAIL

3. [Remove free node from availability stack]
   AVAIL ← LINK (AVAIL)

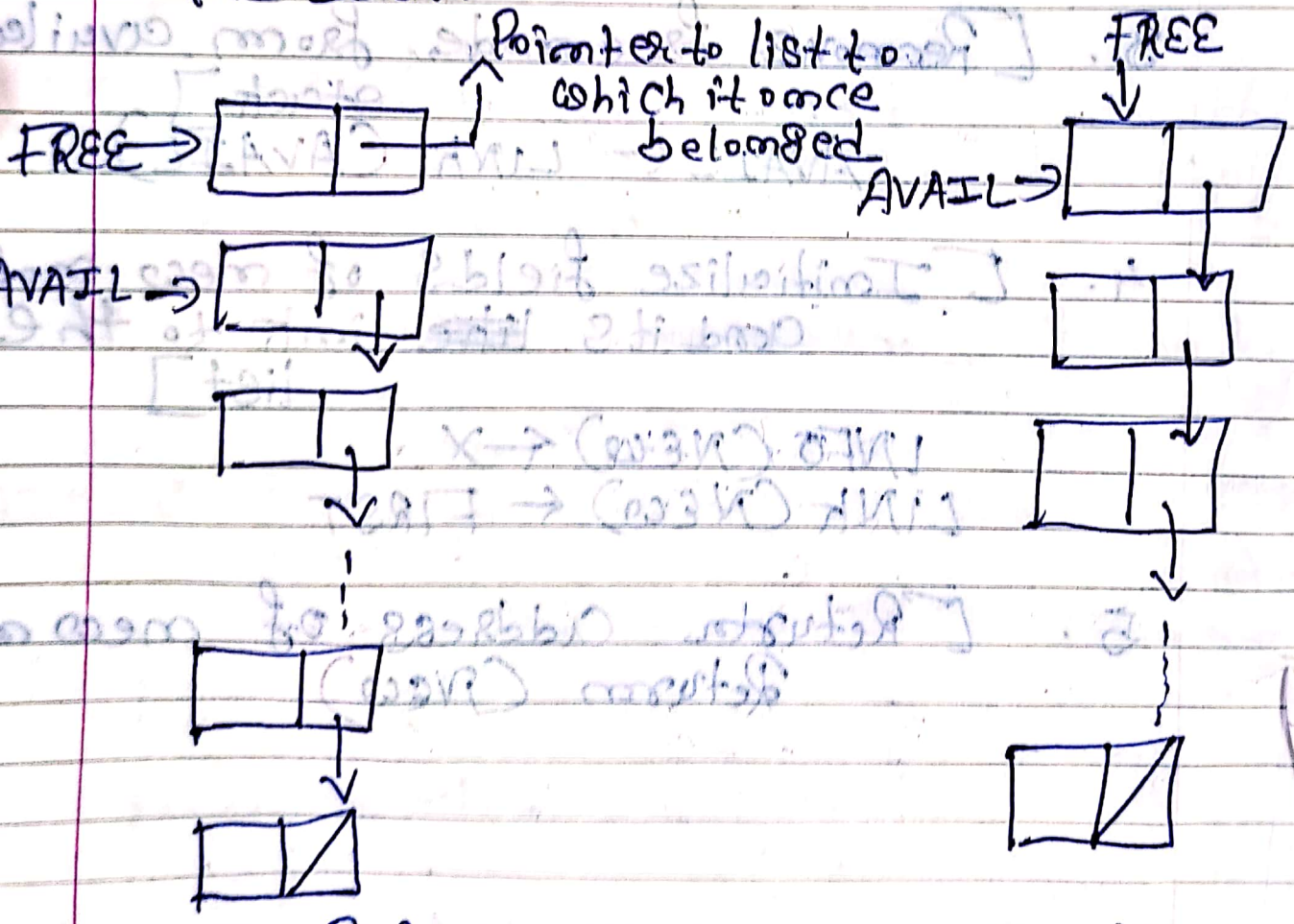
4. [Initialize fields of new node and its ~~three~~ link to the list]
   INFO (NEW) ← X
   LINK (NEW) ← FIRST

5. [Return address of new node]
   Return (NEW)

Insert the node at last Position.

function INSEND(X, FIRST)

→ NEW and SAVE are temporary Pointer variables. it is require that X be inserted at the end of the list.



(a) Before (b) After

Availibility stack before and After a free node has been return.

1. [Underflow ?]

if AVAIL = NULL
then write ('Availability Stack Underflow')
Return(FIRST)

2. [Obtain address of next free node]

NEW ← AVAIL

3. [Remove free node from Availability Stack]

AVAIL ← LINK(AVAIL)

4. [Initialize fields of new node]

INFO(NEW) ← X
LINK (NEW) ← NULL

5. [Is the list empty ?]

if FIRST = NULL
then Return (NEW)

6. [Initiate Search for the last node]

SAVE ← FIRST

7. [Search for end of list]
Repeat while LINK (SAVE) ≠ NULL
SAVE ← LINK (SAVE)

8. [Set LINK field of last node to NEW]
LINK (SAVE) ← NEW

9. [Return first node Pointer]
Return (FIRST)

* Insert node Orderwise.

Function → INSORD (X, FIRST)

→ it is required that X be inserted so that it Preserves the ordering of the terms in increasing order of their INFO fields.

1. [Underflow?]

if AVAIL = NULL
then write ('Availability Stack Underflow')
Return (FIRST)

2. [Obtain address of next free node]
NEW ← AVAIL

3. [Remove node from the availability Stack]
AVAIL ← LINK (AVAIL)

4. [Copy information Content into new node]
INFO (NEW) ← X

5. [Is the list empty?]
if FIRST = NULL
then LINK (NEW) ← NULL
Return (NEW)

6. [Does the new node Precede all others in the list?]

if INFO (NEW) ≤ INFO (FIRST)
then LINK (NEW) ← FIRST

Page
Date

Return (NEW)

7. [Initialize temporary Pointer]
SAVE ← FIRST

8. [Search for Predecessor of new node]
Repeat while LINK(SAVE) ≠ NULL
and INFO(LINK(SAVE)) ≤ INFO(IW)
SAVE ← LINK(SAVE)
~~LINK (SAVE) ← NEW~~

9. [set link field of new node and its Predecessor]
LINK(NEW) ← LINK(SAVE)
LINK(SAVE) ← NEW

10. [Return First node Pointer]
Return (FIRST)

* Delete the node from linked-list.

DELETE(X, FIRST)

→ TEMP is used to find the desired node, and PRED keeps track of the Predecessor of TEMP.

1. [Empty list ?]
   if FIRST = NULL
   then Write('UNDERFLOW')
   Return

2. [Initialize Search for X]
   TEMP ← FIRST

3. [Find X]
   Repeat thru Step 5 while
   TEMP ≠ X and LINK(TEMP) ≠ NULL

4. [Update Predecessor marker]
   PRED ← TEMP

5. [Move to next node]
   TEMP ← LINK(TEMP)

6. [END of the list ?]
   if TEMP ≠ X

then : Write ('NODE NOT FOUND')

Return

7. [Delete X]

if X = FIRST (IS X the first node?)

then FIRST ← LINK (FIRST)

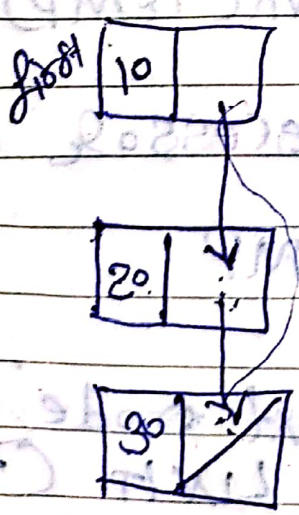
else LINK (PRED) ← LINK (X)

8. [Return node to Availibility area]

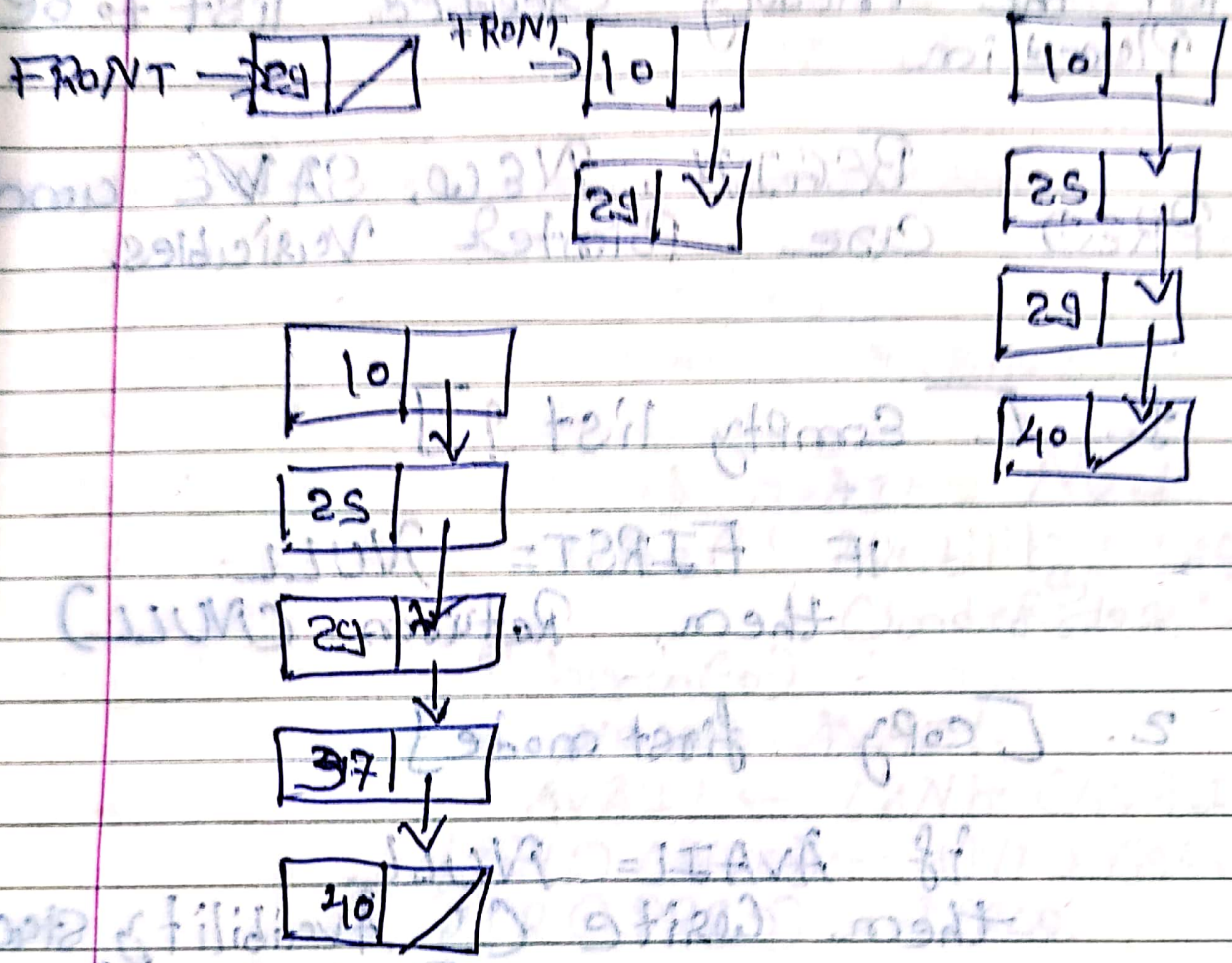
LINK(X) ← AVAIL

AVAIL ← X

Return



* Construction of an ordered linked list using fun. INSORD



* Copy Node in link list.

COPY (FIRST)

→ This fun creates a copy of this list.

→ The New list is to contain nodes whose in formation and Pointer field denoted by field and Ptr.

FIELD & PTR.

→ The address of the FIRST node in the newly created list to be placed in.

BEGIN, NEW, SAVE and PRED are pointer variables.

1. [Empty list ?]

IF FIRST = NULL
then Return (NULL)

2. [Copy first node]

if AVAIL = NULL
then Write ('Availability Stack Underflow')
Return(0)
else
NEW ← AVAIL
AVAIL ← LINK(AVAIL)
FIELD(NEW) ← INFO(FIRST)
BEGIN ← NEW.

3. [Initialize traversal]
SAVE ← FIRST

4. [Move to next node if not at end of list]

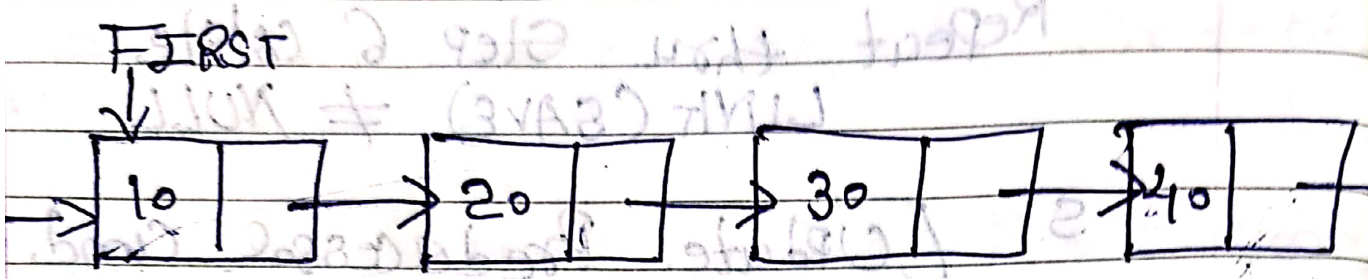
Repeat thru Step 6 while LINK(SAVE) ≠ NULL

5. [Update Predecessor and Save Pointers].
PRED ← NEW
SAVE ← LINK(SAVE)

6. [Copy node]
if AVAIL = NULL
then Write('Availability Stack Underflow')
Return(0)
else NEW ← AVAIL
AVAIL ← LINK(AVAIL)
INFO(NEW) ← INFO(SAVE)
PTR(PRED) ← NEW.

7. [Set link of last node and Return]
PTR(NEW) ← NULL
Return (BEGIN)

The Circular linked list



head node

Replacing the null pointer in the last node of a list with the address of its first node. Such a list is called a Circular linked linear list or Circular list.

Advantage

) Is Concerned with the accessibility of a node. In circular list every node is accessible from a given node.

) We can easily delete the node.

* Disadvantage

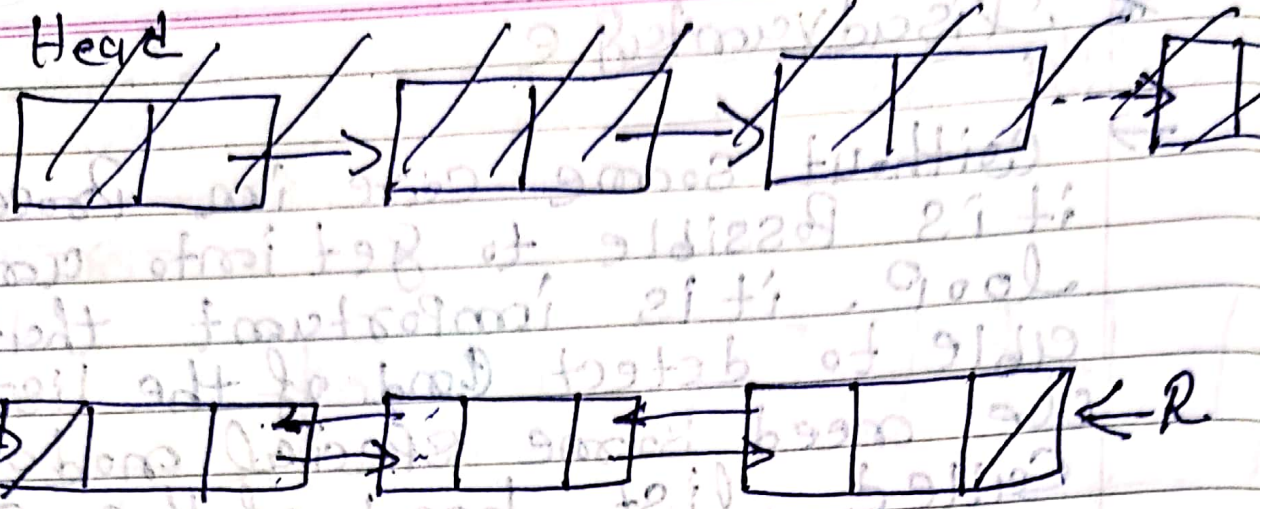
→ Without some care in Processing, it is Possible to get into an infinite loop. it is important that we are able to detect end of the list we need some special node is called list head of the circular list.

* Doubly linked linear list

→ The link are used to denote the Predecessor and successor of a node. The link denoting the Predecessor of node is called left link.

and denoting its successor its right link.

* A list Containing this types of node is called a doubly linked linear list or two-way Chain.



A doubly linked list.

→ Where L and R are Pointer variable denoting the left-most and right-most node in list.

→ The left-link of lef-most node and the right link of the right-most node are both NULL, indicating the end of the list for each direction.

→ The lef & Right links of a node are denoted by the variable LPTR and RPTR.

* Insertion in Doubly linked list.

* DOUBINS(L, R, M, X).

→ Where L and R are Pointer Variable denoting the left-most and Right most node in the list.

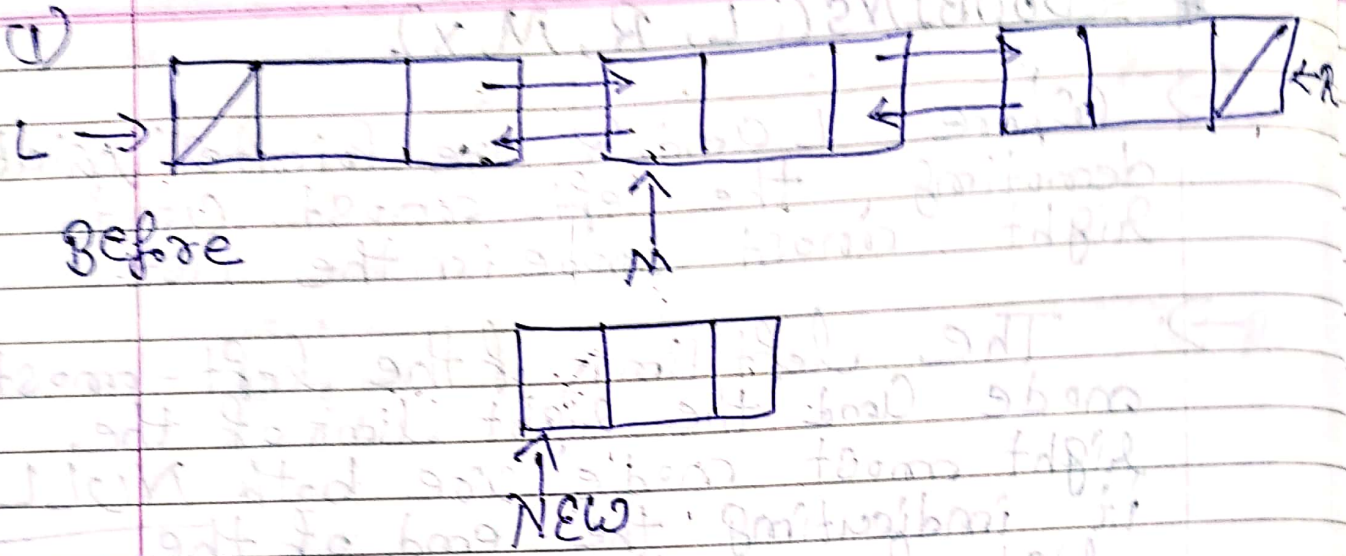
→ The left link of the left-most node and the right link of the right most node are both NULL it indicating the end of the list for each direction.

→ Left-link of node & Right link of node denoted by LPTR and RPTR.

→ Insert the node into a doubly linked list to the left of a specified node whose address is given by Variable M.

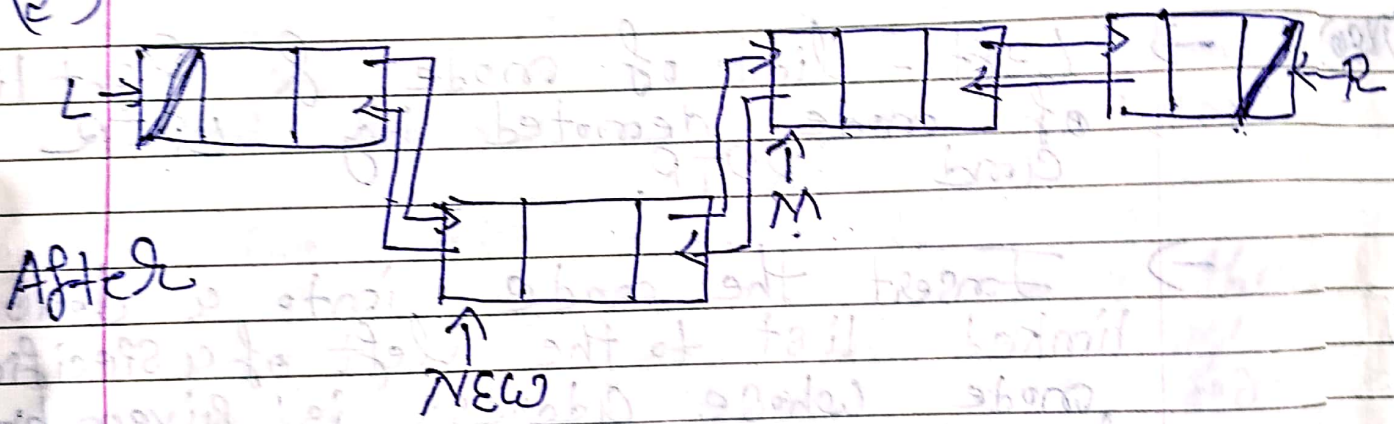
→ NEW is the address of the new node being inserted.

①



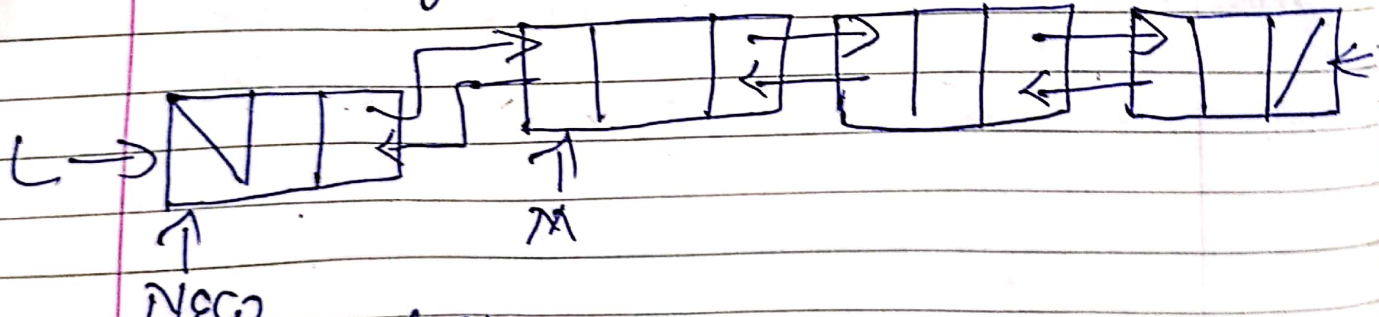
Before

②



After

Insertion in the middle of a doubly linked list.



left-most insertion.

1. [obtain new node from availibility Stack]

NEW ← NODE

2. [Copy information field]

INFO (NEW) ← X

3. [Insertion into an empty list]

if R = NULL

then LPTR (NEW) ← RPTR (NEW) ← NULL

L ← R ← NEW

Return.

4. [left-most insertion]

if M = L

then LPTR (NEW) ← NULL

RPTR (NEW) ← M

LPTR (M) ← NEW

L ← NEW

Return

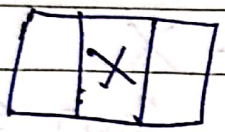
5. [Insert in middle]

LPTR (NEW) ← LPTR (M)

RPTR (NEW) ← M.

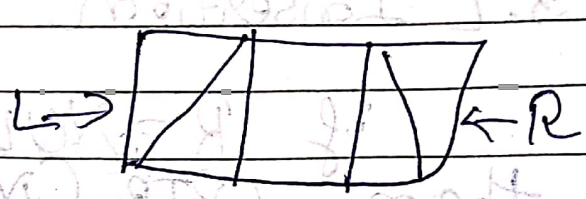
LPTR (M) ← NEW

RPTR(LPTR(NEW)) ← NEW
Return..



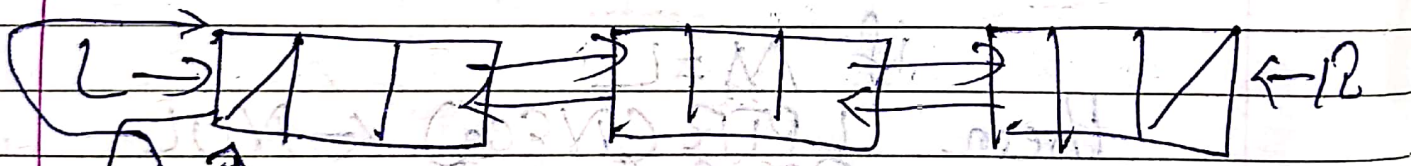
NEW

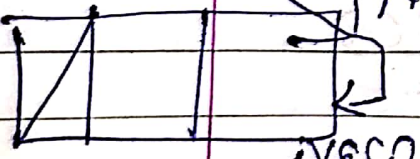
if no any node in link list



LPTR(NEW) ← RPTR(NE
L ← R ← NEW.

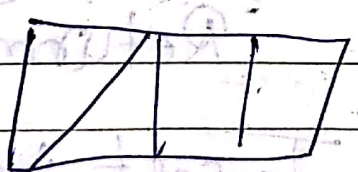
left- most insertion.





LPTR(M) ← NEW
RPTR(NEW) ← M

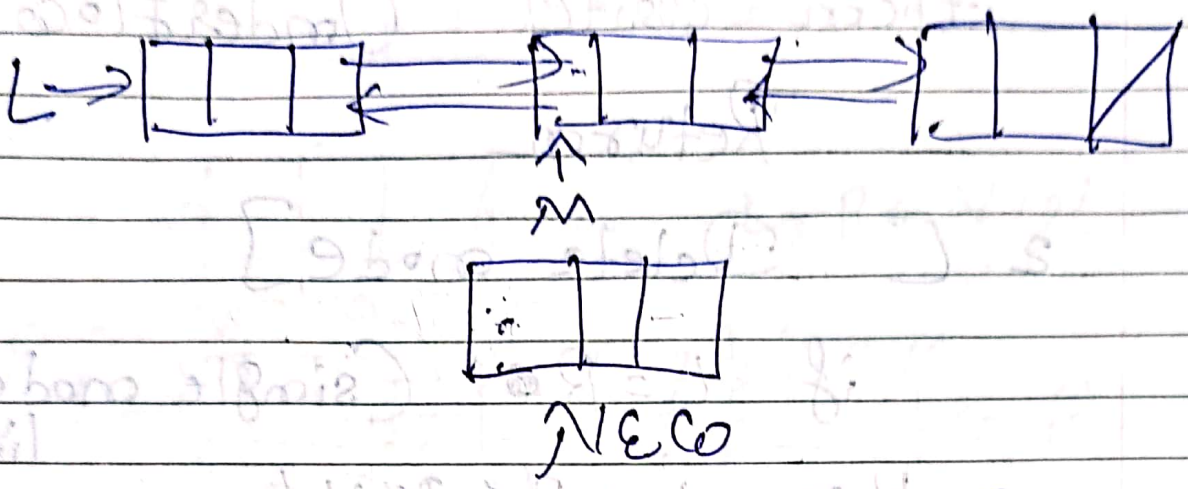
M = L



NEW

LPTR(NEW) ← M

In middle.



* Delete the node from Doubly linked list.

←NULL

* DOUBDEL (L, R, OLD)

→ Doubly linked list with the addresses of the left-most and right most nodes given by Pointer variable L and R.

→ it Require to delete the node whose address is contained in the variable OLD.

1. [Underflow?]

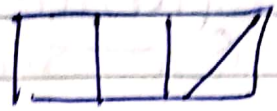
if R = NULL

then Write('Underflow')
Return

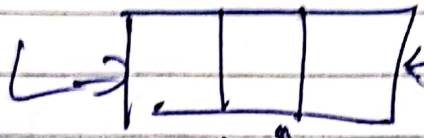
2. [Delete node]
if L=R (single node in list)
then L←R←NULL
else if OLD=L (left-most node being deleted)
then L←RPTR(L)
LPTR(L)←NULL
else if OLD=R (Right-most node being deleted)
then R←LPTR(R)
RPTR(R)←NULL
else RPTR(LPTR(OLD))←RPTR(
LPTR(RPTR(OLD))←LPTR

3. [Return deleted node]
Restore(OLD)
Return.

(1)

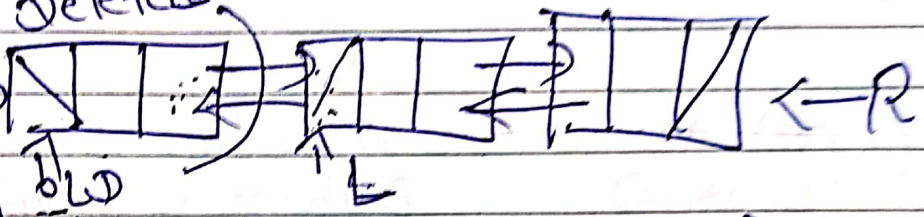


← R = NULL



L → R ← R ← NULL

Node Deleted

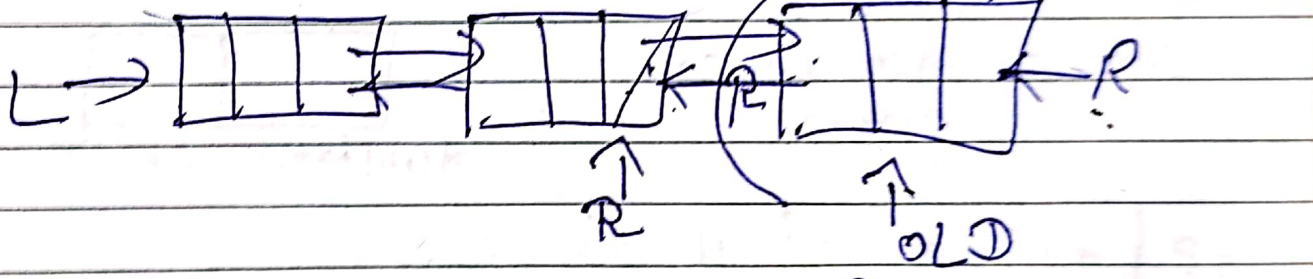


if OLD = L

RPTR(L) → L

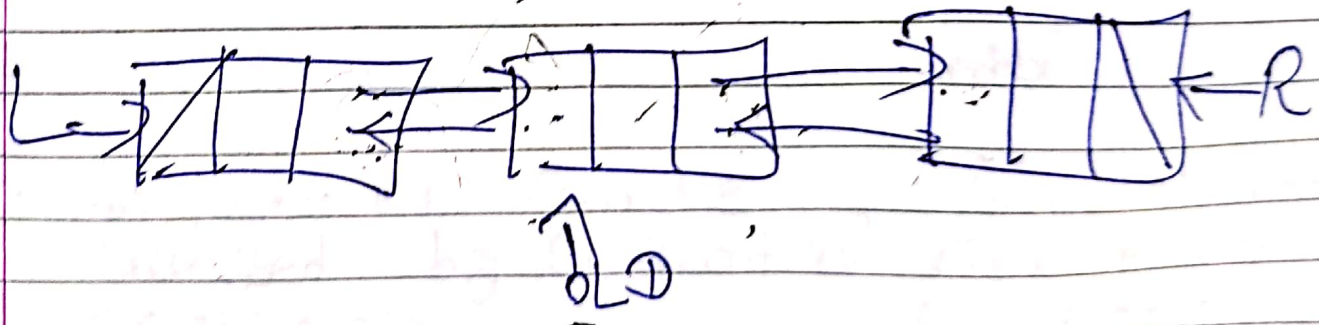
LPTR(L) ← NULL

Node deleted



OLD = R

if middle

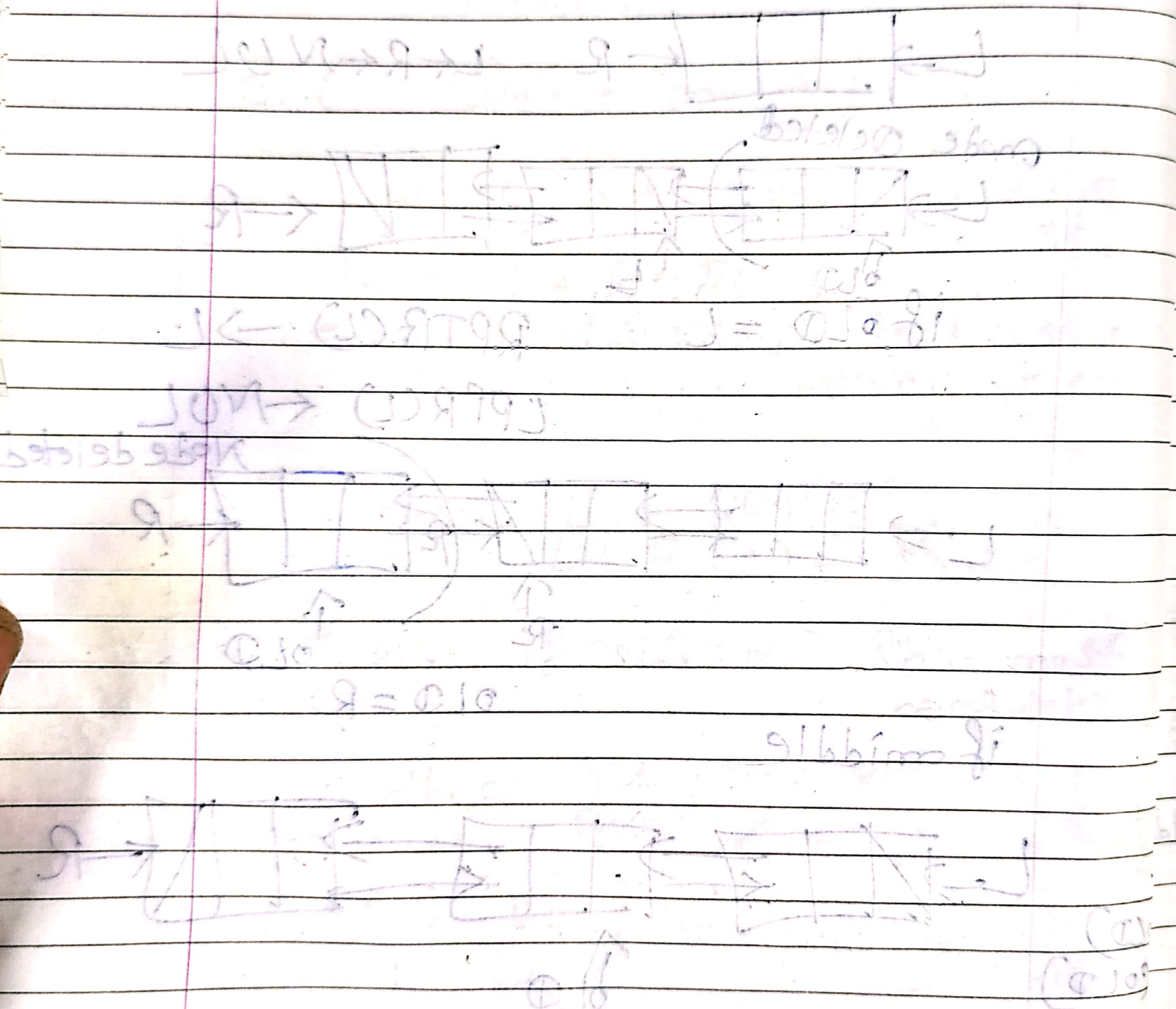


OLD)

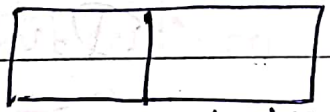
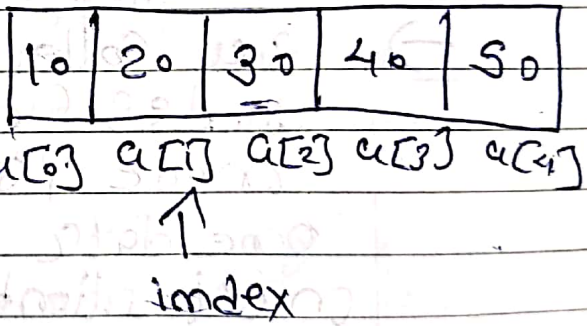
(OLD)

Page
Date

Circular - linked list.



* Difference betn linked-list and Array.

Link-list	Array
→ The link-list is a collection of nodes and each node is having one data field and next link field 	The array is a collection of similar datatypes elements. the data is always stored at some index of the array. 
→ Any element can be accessed by sequential access only.	Any element can be accessed randomly.
→ Physically the data can be deleted	only logical deletion of data is possible.

→ Insertion & deletion of data is easy.

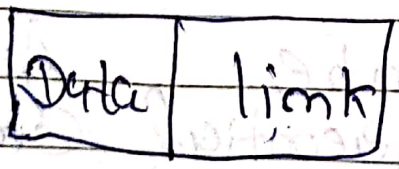
Insertion & deletion of data is difficult.

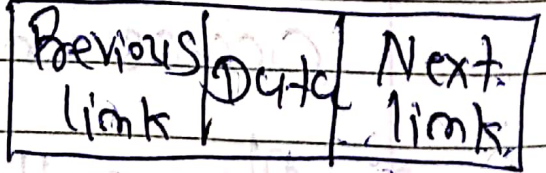
→ Memory allocation is dynamic.

Memory allocation is static.

Diff. bet^{n} Singly & Doubly linked-list.

Singly linked-list	Doubly linked list
→ is a collection of nodes and each node is having one data field and next link field.	is a collection of node and each node having one data field, one Previous link field and one next link field.





→ The element can be accessed using next link.	The elements can be accessed using both link
→ No extra field is required. node take less memory.	One field is required to store previous link node take more memory.
→ less efficient to access to elements.	More efficient access to elements.

* Application of linked-list.

→ Representation of Polynomial and Performing various operation such as Addition, mul and Evaluation on it.

→ Performing addition of long Positive integers.

→ Representing non integer and non homogeneous list.

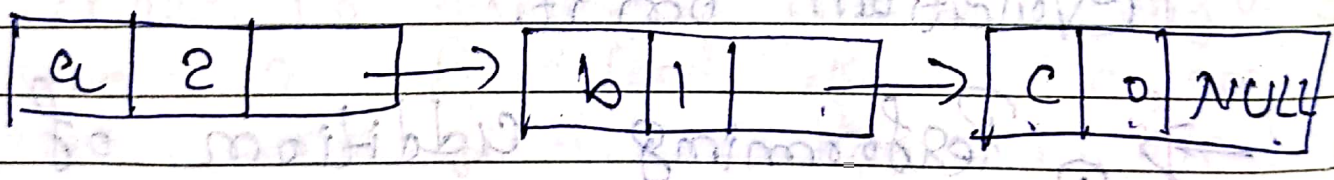
* link list Application for quadratic equation addition.

→ link-list used datast. for representing quadratic eqn.

```
Struct Quad_eq
{
    int coeff;
    int exp;
    Struct Quad_eq * next;
};
```

```
Struct Quad_eq *eq1, *eq2, *eq3;
```

Graphically any quadratic eqn can be represented using linked list.



$$ax^2+bx+c$$

Consider eqn

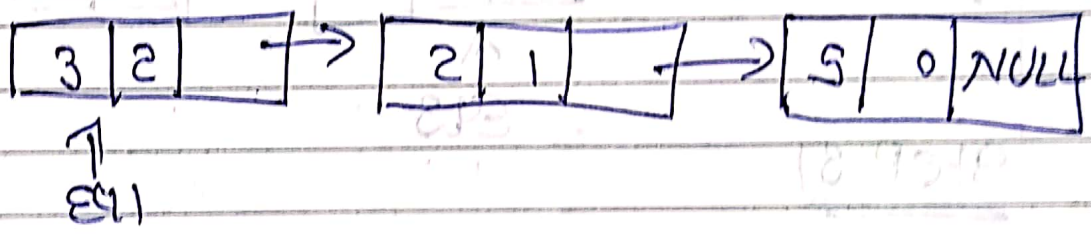
$$eq_1 = 3x^2+2x+5$$
$$eq_2 = 7x^2+3x+8$$

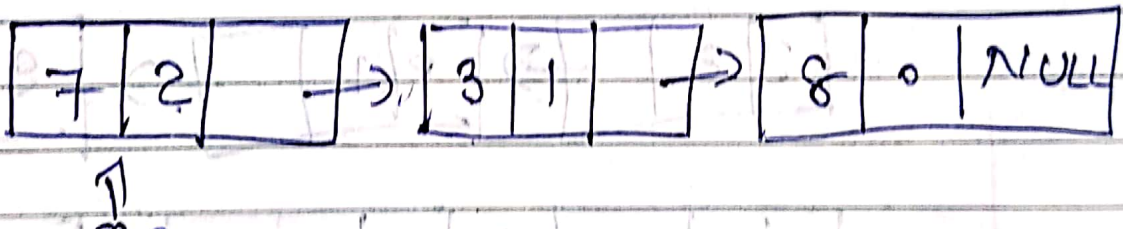
Here two quadratic eq^{n}. we can Perform their addition as

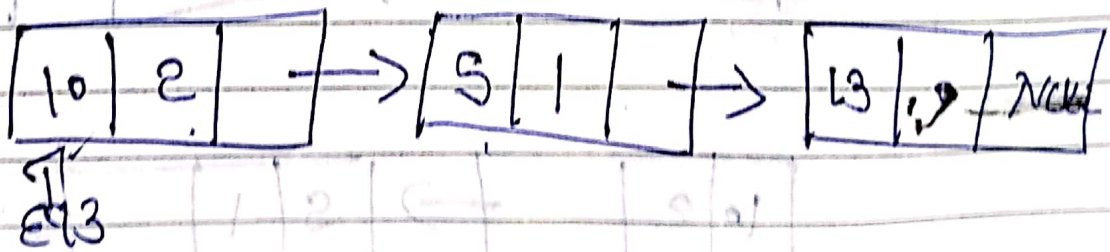
$$3x^2 + 2x + 5$$
$$+ 7x^2 + 3x + 8$$
$$\overline{10x^2 + 5x + 13}$$

Perform using linked list.

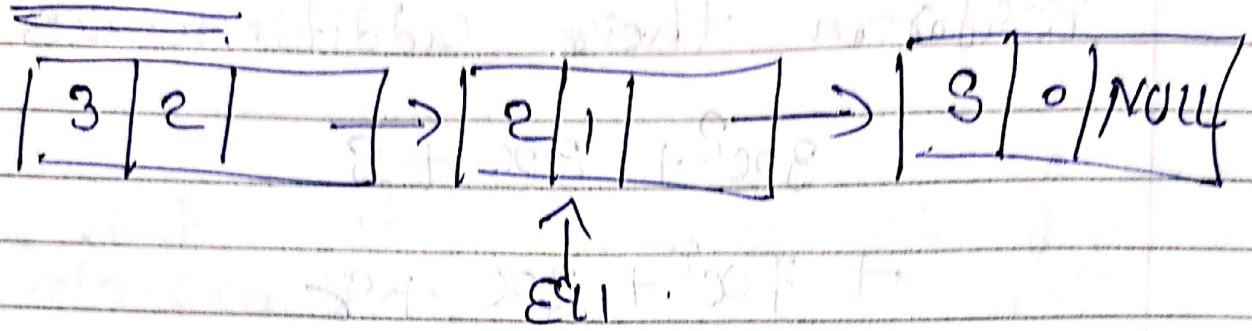
Step 1

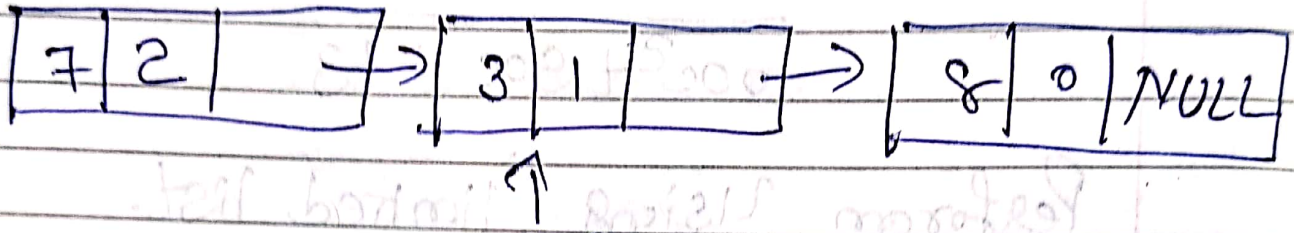


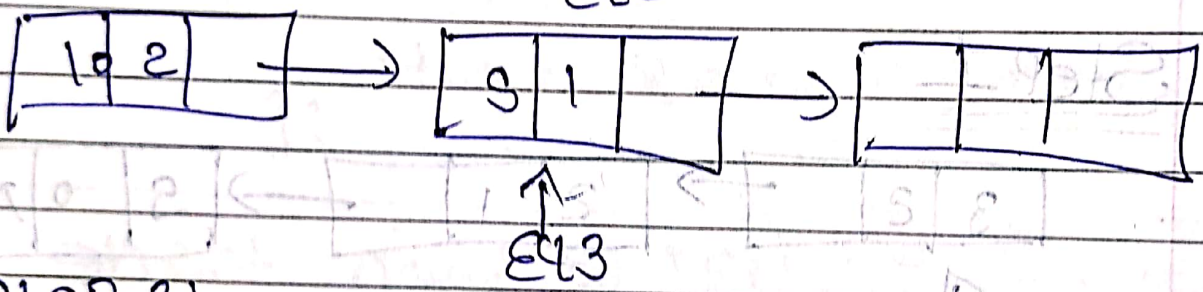




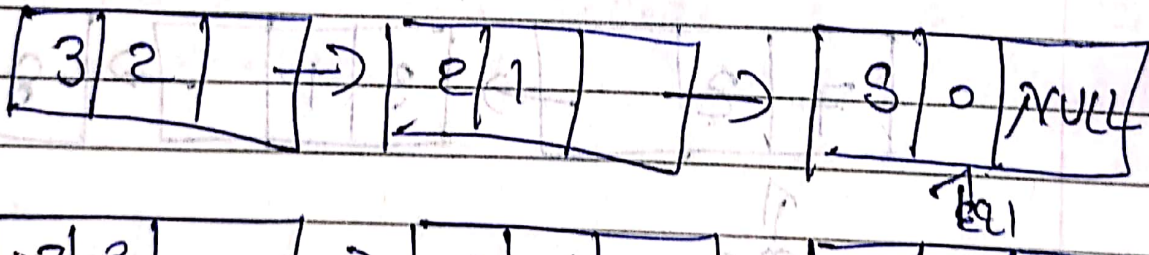
Step 2

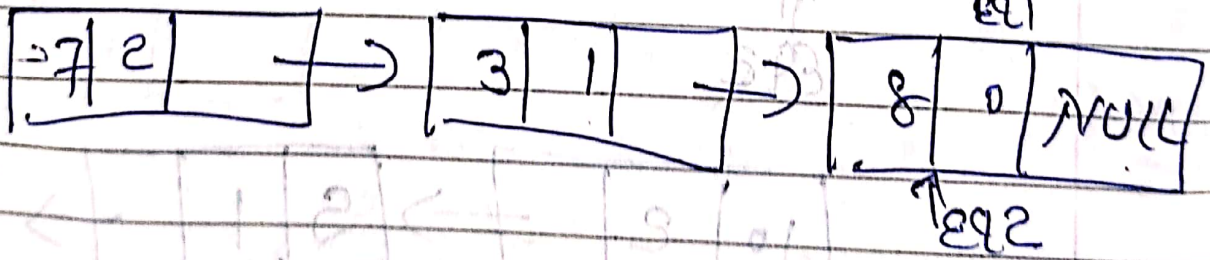


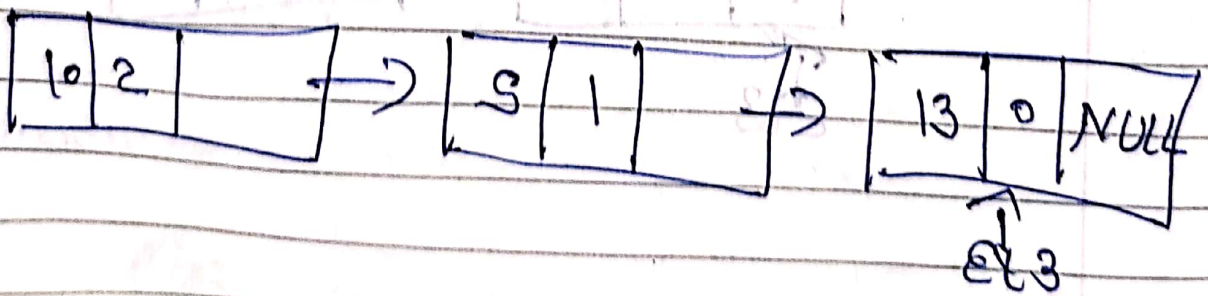




Step 3:







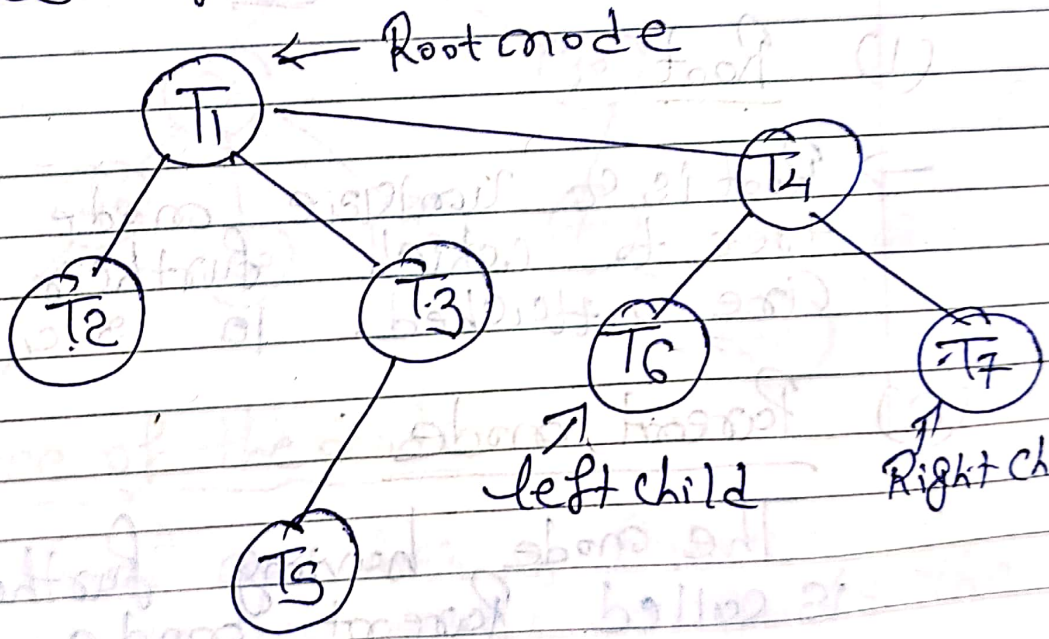
Trees

Definition and concepts

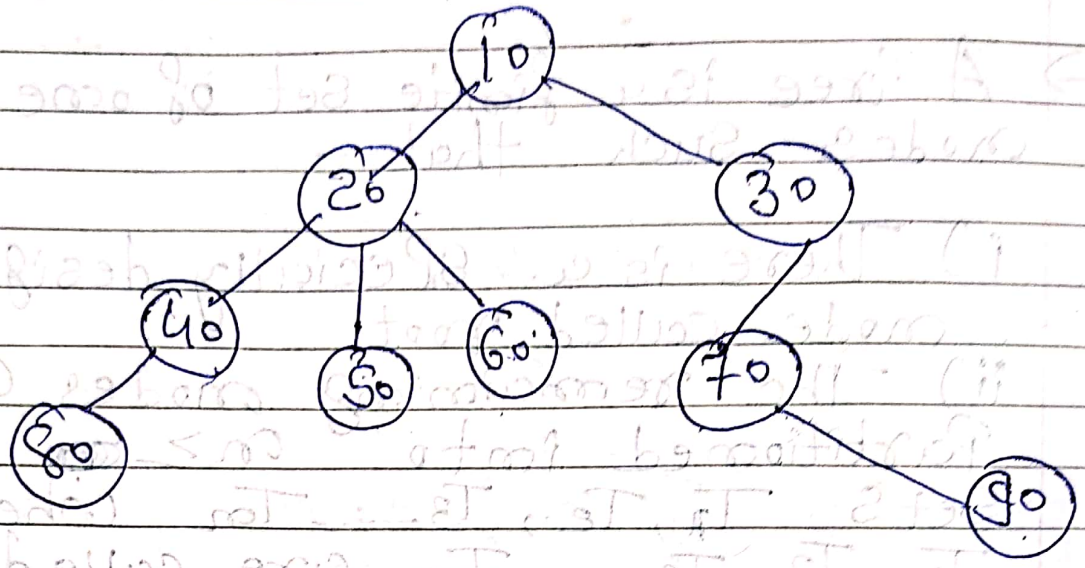
→ A tree is a finite set of one or more nodes such that —

i) There is a specially designed node called root.

ii) The remaining nodes are partitioned into $n \geq 0$ disjoint sets $T_1, T_2, T_3 \ldots T_n$ where $T_1, T_2, T_3 \ldots T_n$ are called the subtrees of the root.



Basic Defination



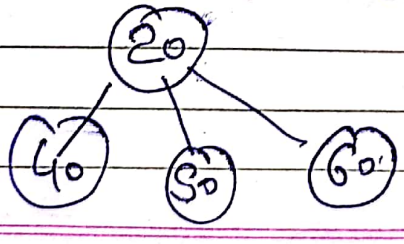
Tree

(1) Root

→ Root is a unique node in the tree to which further subtrees are attached. 10 is a root node.

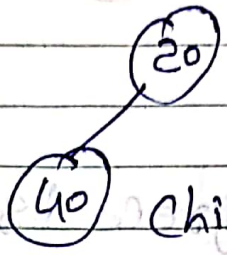
(2) Parent node

The node having further subbranches is called Parent node.



(3) Child nodes

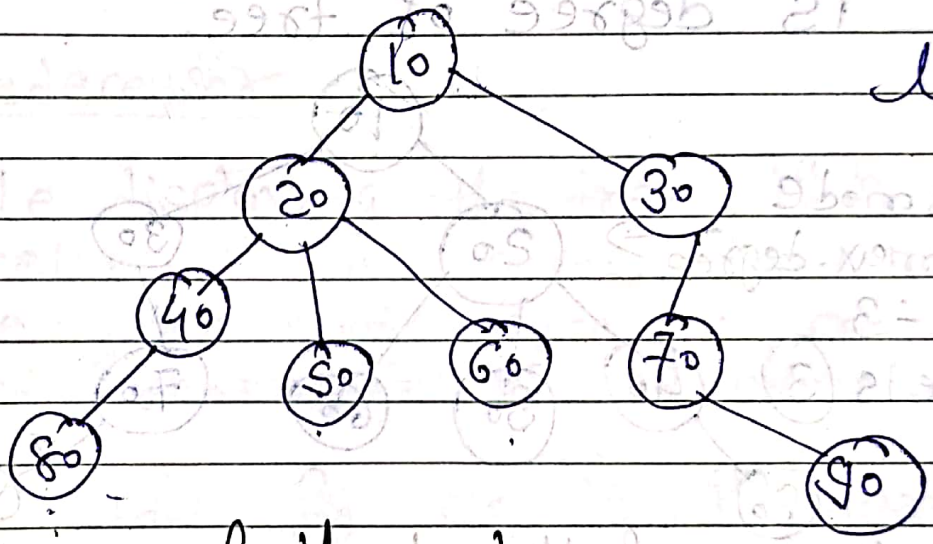
→ The child nodes in above given tree are marked as follow



Child node.

(4) Leaves

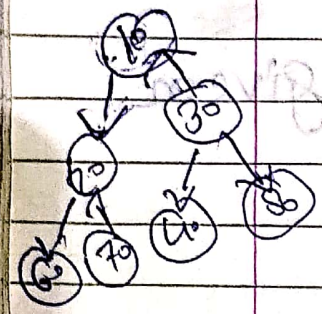
These are the terminal nodes of the tree.

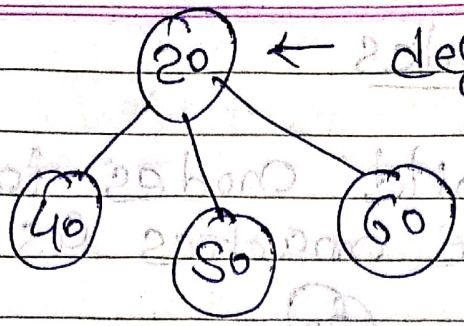


leaf

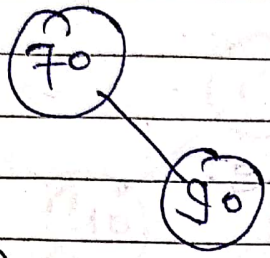
(5) Degree of the node

The total number of subtrees attached to that node is called the degree of a node.





← degree of this node is 3.



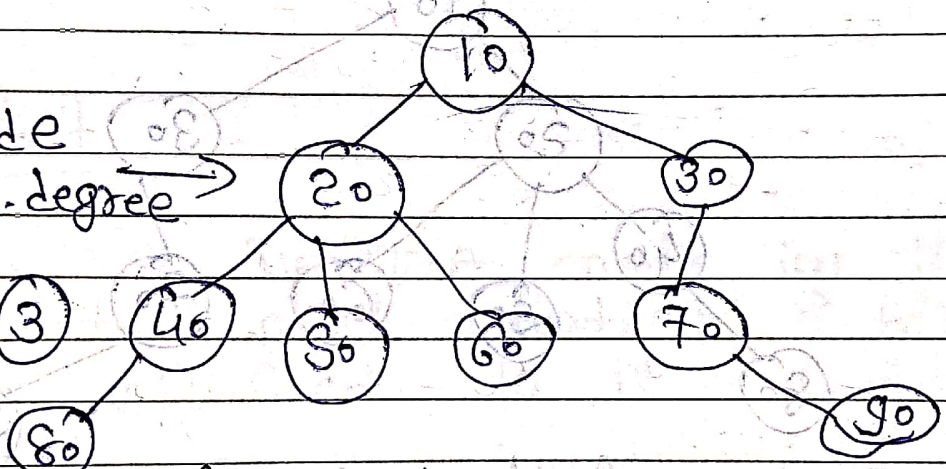
Degree is 1.

(6) Degree of tree

The maximum degree in the tree is degree of tree.

This node with max. degree is -3

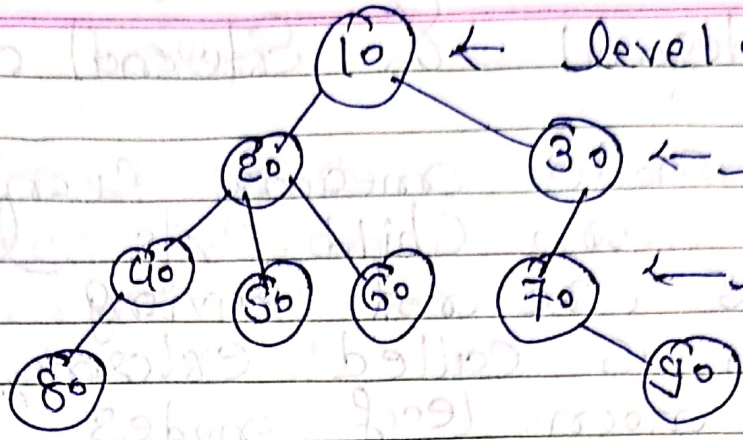
So degree is 3



(7) Level of the tree

The root node is always considered at level zero.

The adjacent nodes to root are supposed to be at level 1 and so on.



(8) Height of the tree

The maximum level is the height of the tree. Also called (depth) of tree. In above fig. hight is 3.

(9) Predecessor

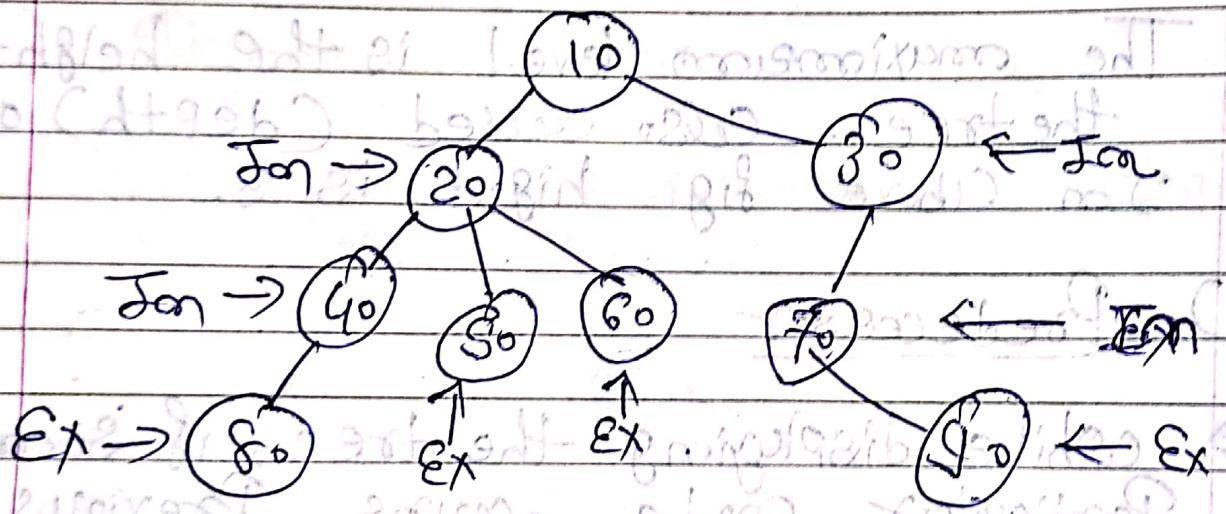
→ while displying the tree, if some Particular node occurs Previous to Some other node then that node is called Predecessor of the other node.

In above fig. 20 is Predecessor of 40.

(10) Successor is called next node. if we read node 60 after reding 20 then 60 is called Successor of 20.

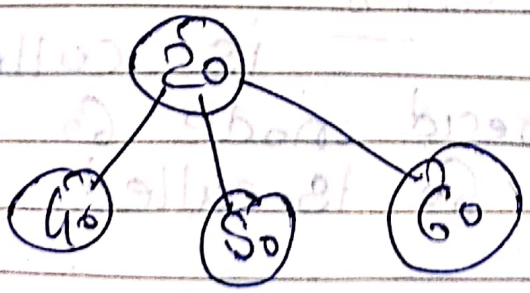
(11) Internal & External nodes.

⇒ leaf node means a node having no child. As leaf nodes are not having further links, is called external nodes and non-leaf nodes are called internal node.



(12) Sibling

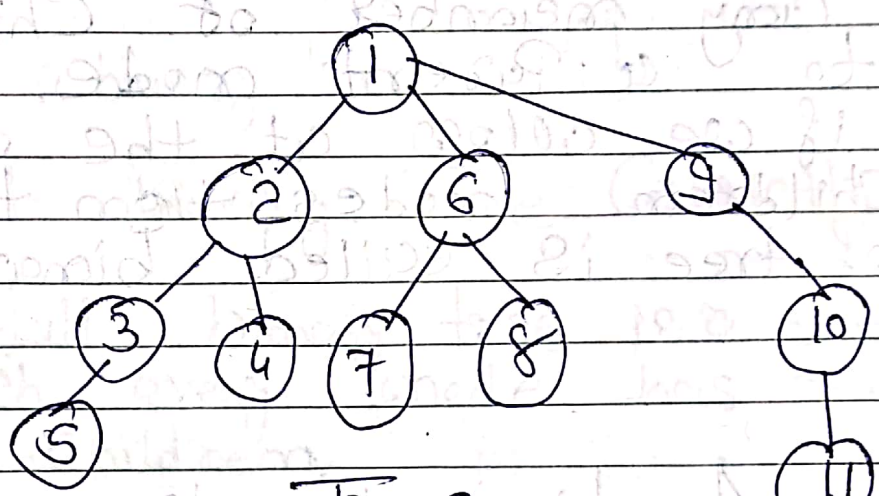
The nodes with common Parent are called Siblings or brothers



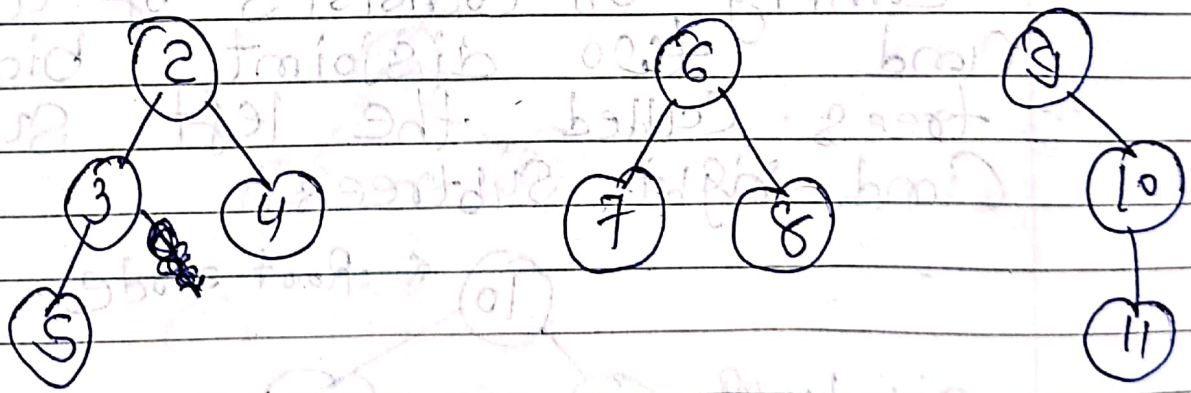
40, 50 & 60 are sibling node

(13) Forest

→ Forest is a collection of disjoint trees. From a given tree if we remove its root then we get a forest.



Tree



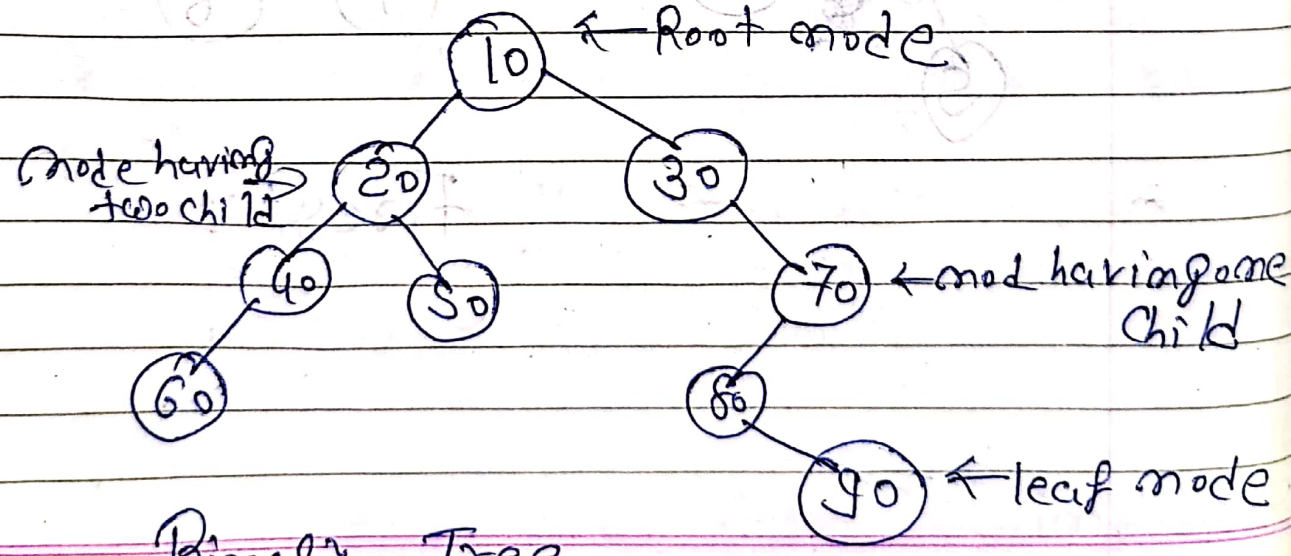
Forest.

* Concept of Binary Trees.

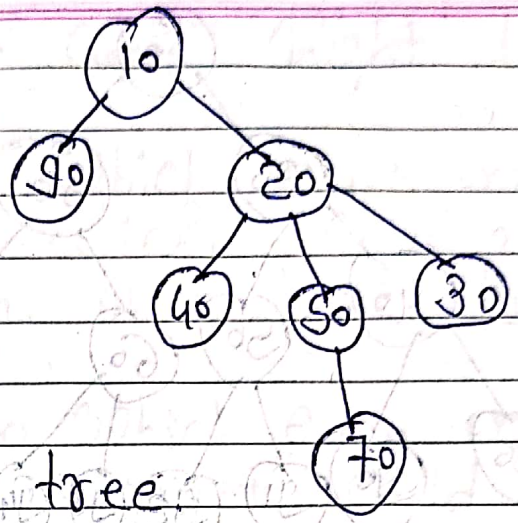
GTU 3 Marks.

3 → In this trees, we can have any number of Child nodes to a Parent node.
→ if we allocate at the most two Children nodes then that type of tree is called binary tree.

Def A binary tree is a finite set of nodes which is either empty or consists of a root and two disjoint binary trees called the left subtree and right subtrees.



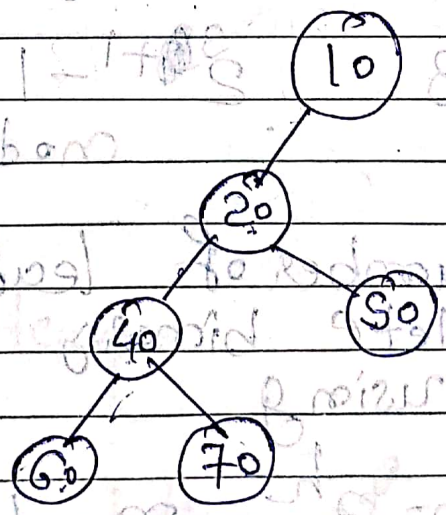
Binary Tree



Not a binary tree.

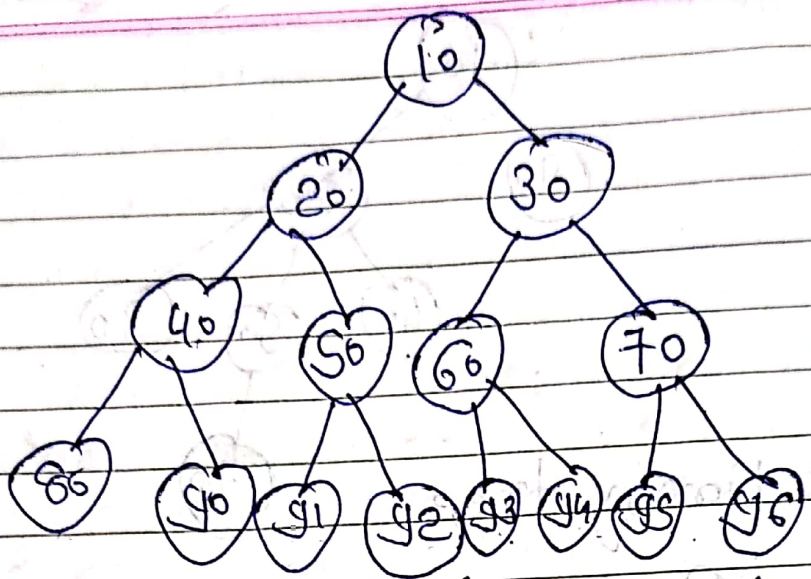
2 Marks * Full Binary Tree (Strictly Binary Tree)

→ A full binary tree is a tree in which every node has zero or two children.



3 Marks * Complete Binary Tree.

A complete binary tree is full binary tree in which all leaves are at the same depth.



→ The Total number of nodes in Complete binary tree are $2^{h+1}-1$ where h is a height of the tree. In above fig.

h is 3 $2^{3+1}-1 = 2^4-1 = 15$ nodes in the tree

→ The number of leaf nodes in a Complete binary tree can be found using

$n = 2^h$ n total No. of leaf node

h is height of binary tree.

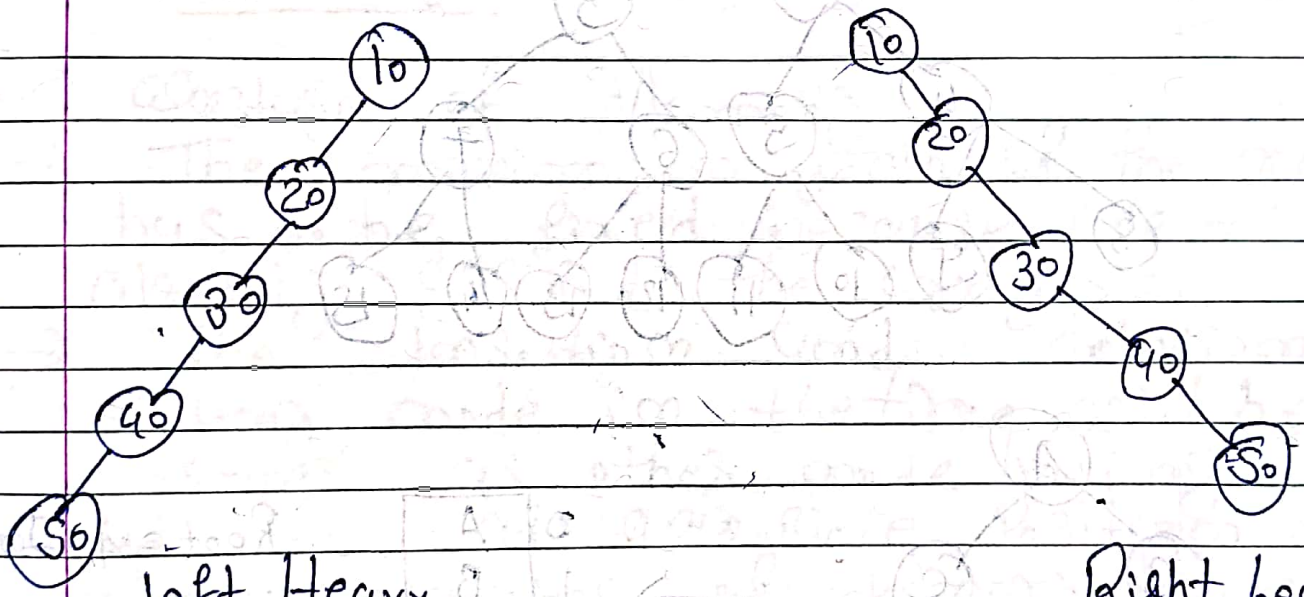
h=3

$2^3 = 8$ node is leaf node.

* Left Heavy and Right Heavy Trees.

→ The tree in which each node is attached as a left child of Parent node then it is left heavy tree.

→ The tree in which each node is attached as a right child of Parent node then it is called right heavy tree.



Left Heavy Right heavy.

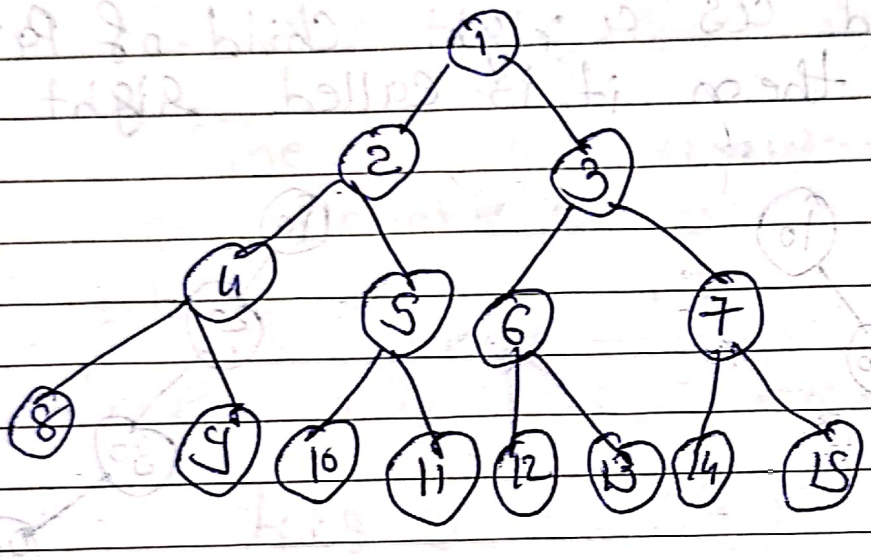
* Representation of Binary Tree:-

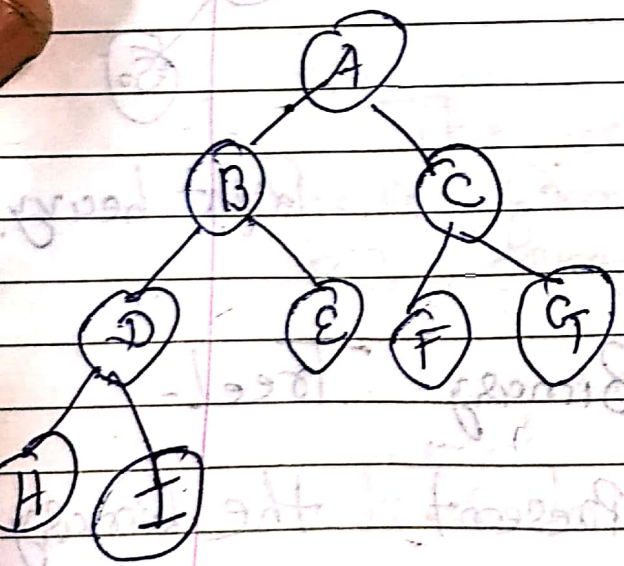
→ Two ways to represent the binary tree

(1) Sequential Representation.
(2) Linked ".

(1) Sequential or Array Representation of tree.

→ Each node is sequentially arranged from top to bottom and from left to right.





0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H
8	I

Root = A = Index 0

a[2]

Advantages

→ The direct access to any node can be possible and finding the Parent,

→ left or right children of any Particular node is fast because of the random access.

Disadvantages

(1)→ wastage of Memory.

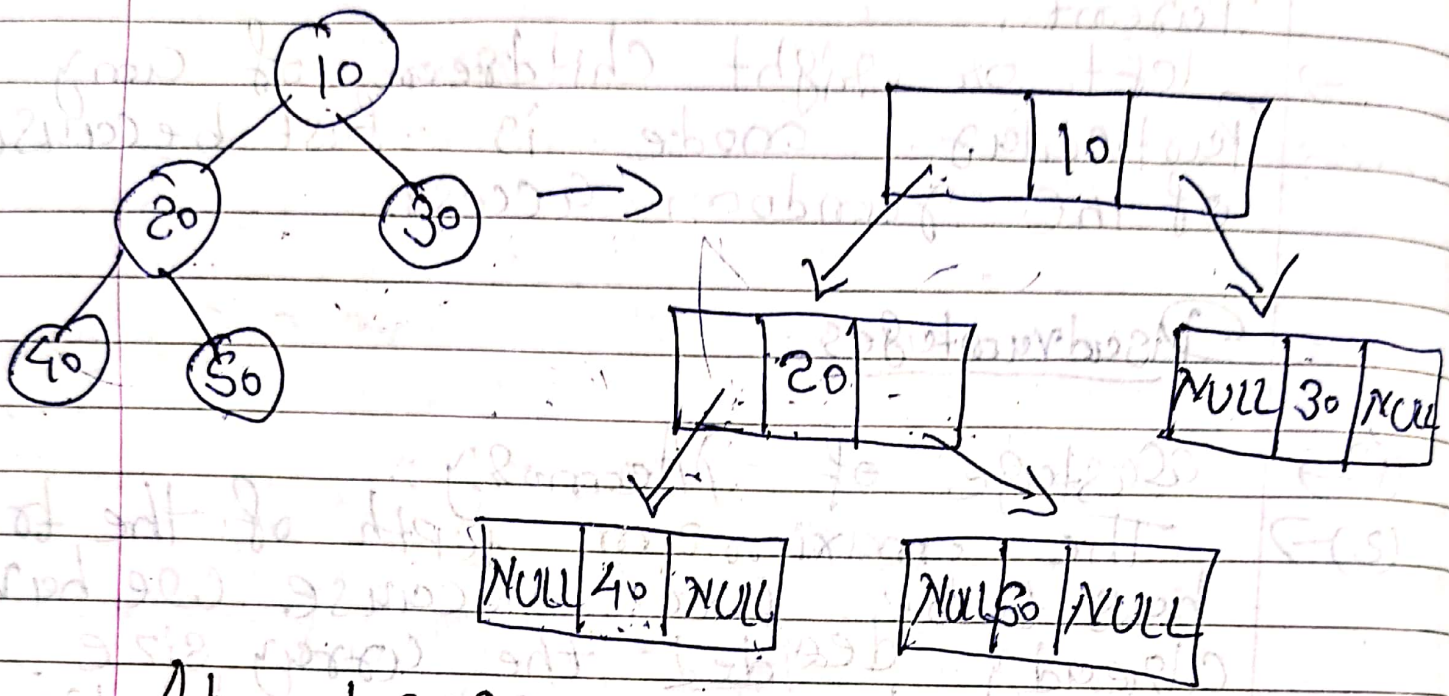
(2)→ The maximum depth of the tree has to be fixed because we have already decided the array size.

(3)→ The insertion and deletion of any node in the tree will be Costlier as other node having to be adjusted at appropriate position so that meaning of binary tree can be preserved.

* Linked Representation or node Representation of binary trees.

A binary tree each node will have left child, right child and data field.

left child	Data	Right child



Advantages

→ No wastage of memory.

→ Insertion and deletions which are the most common operations can be done without moving the other node.

Disadvantage

→ This representation does not provide direct access to a node and special algorithm are required.

⇒ This representation needs additional space in each node for storing the left and right subtrees.

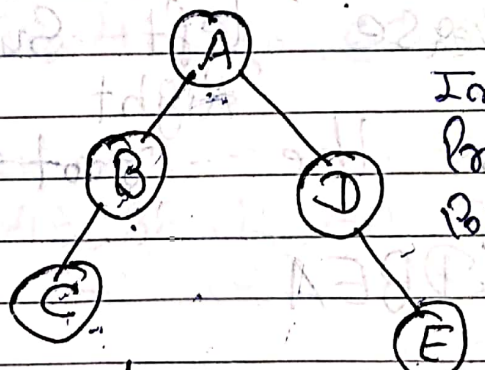
Binary Tree- Traversal

GTU 7 marks.

⇒ Traversing the tree means visiting each node exactly once.

L means move to the left child
R " " " " Right "
D " the root / Parent node.

(1) Inorder traversal



In: CBADE (LRoR)
Pre: ABCDE (RoLR)
Pst: CBEDA (LRRo)

Inorder traversal of binary tree is given by.

1. Traverse the left subtree in tree
2. Process the root node.
3. Traverse the right subtree.

So, Inorder is

LRoR

CBADE

(2) Preorder traversal of binary tree

(1) Process the root node
(2) Traverse the left Subtree
(3) Traverse the right "

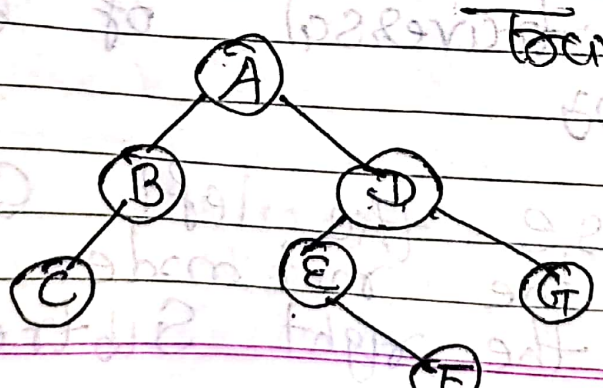
ABCDE (RLR)

(3) Postorder traversal

(1) Traverse left Subtree
(2) " Right " (LRRo)
(3) Process the root node.

CDBEA

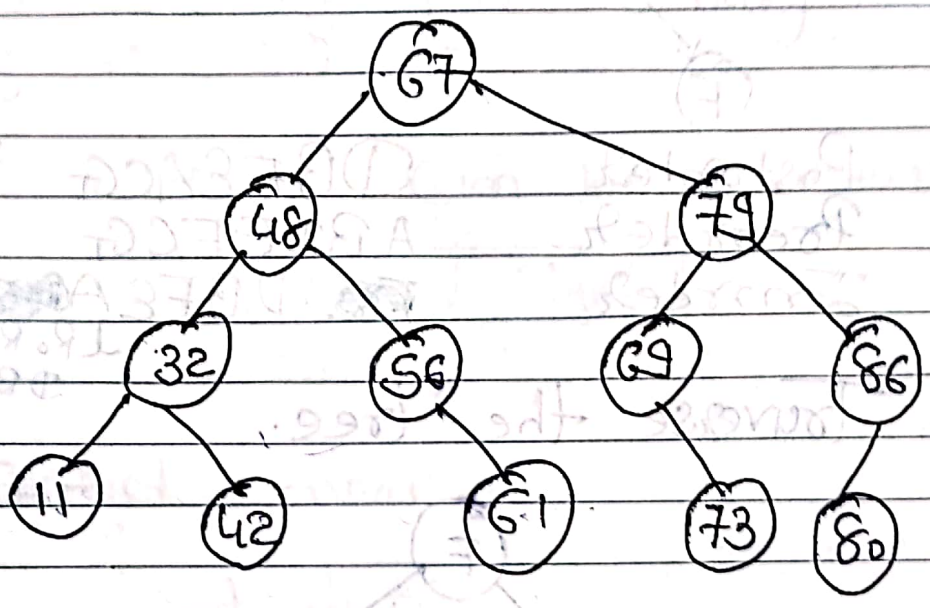
Ex1. Traverse the tree.



Preorder - ABCDEFG
Inorder - CBAEFDG
Postorder - CBFEGDA

Ex1: GTU 3Marks

Traverse the tree.



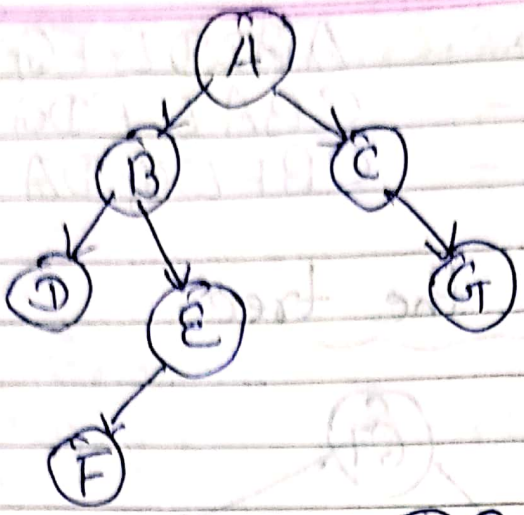
Inorder
11, 32, 42, 48, 56, 61, 69, 73, 79, 80, 86

Preorder
67, 48, 32, 11, 42, 56, 61, 79, 69, 73, 86, 80

Postorder
11, 42, 32, 61, 56, 48, 73, 69, 80, 86, 79, 67

Ex1: 3Marks

Traverse the tree.



LRRo
DFEBGCA

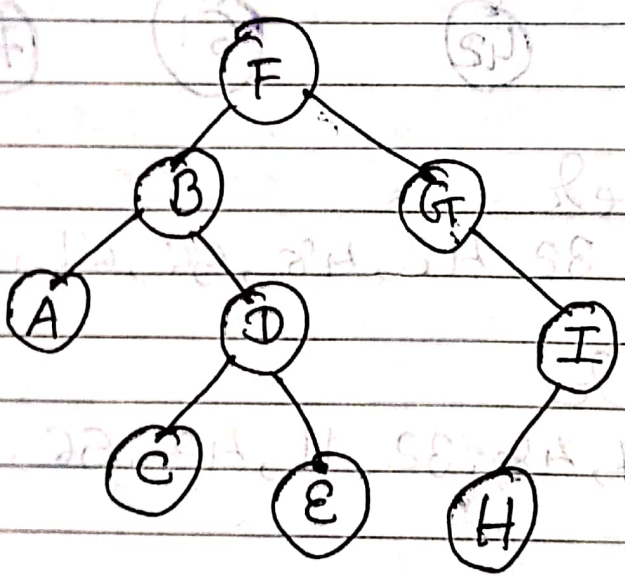
Postorder — DBFEACG
Preorder — ABDEFCG
Inorder — DBFEACG

LRoR

LRoR
DBFEACG

Traverse the tree.

Ex1



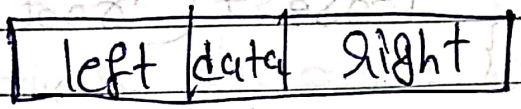
RoLR Pre: F, B, A, D, C, E, G, I, H
LRoR In: A, B, C, D, E, F, G, H, I

Page
Date

Binary Tree Operations.

(1) Creation of binary tree
(2) Insertion of a node in the tree
(3) Display of a binary tree.

node can be represent like this



node

```
Struct node
  {
   int data
   Struct node *left;
   Struct node *right;
};
Void Insert (node *root,
             node *New
{
  Char ch;
Print f (" where to insert left/
                    right
                    of %d"
", root->data)
```

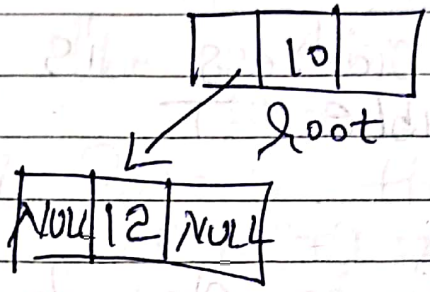
```
ch = getche();
if ((ch=='r')||(ch=='R'))
{
    if(root->right==NULL)
    {
        root->right=Neco;
    }
    else
        insert(root->right,Neco);
}
else
{
    if(root->left==NULL)
    {
        root->left=Neco;
    }
    else
        insert(root->left,Neco);
}
}
```

⇒ Initially root = NULL

NULL	10	NULL

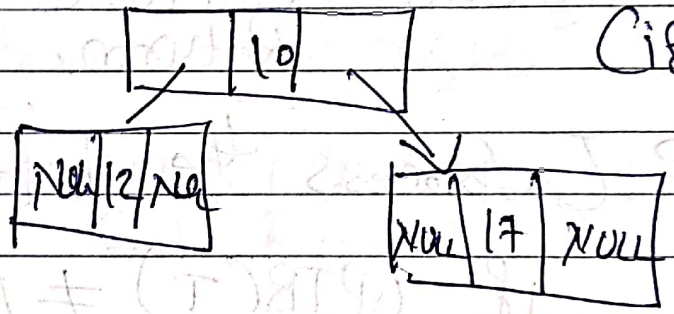
AS Root = NULL , We will call NEW node as Root.

i) Insertion of a node as a left child.



(if Root->left == NULL
Root->left = New

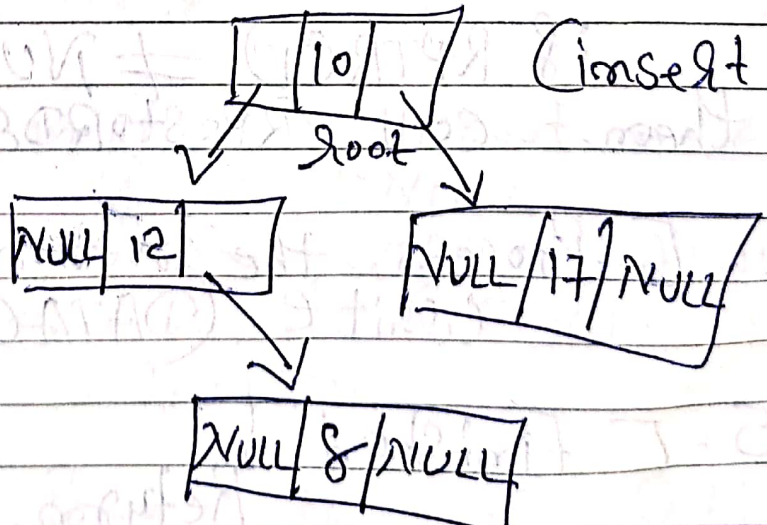
ii) Insertion of a node as a right child



(if Root->right == NULL)
Root->right = New

New

Now if we want to insert node 8 as a right child of node 12 then



(insert (Root->left, New)

New

Recursive Traversal

(1) RPOSTORDER(T)

→ Root node address is given by Pointer variable T.

1. [Check for empty tree]
   if T=NULL
   then Write ('EMPTY TREE')
   Return.

2. [Process the left Subtree]
   if LPTR(T) ≠ NULL
   then call RPOSTORDER(LPTR(T))

3. [Process the right Subtree]
   if RPTR(T) ≠ NULL
   then call RPOSTORDER(RPTR(T))

4. [Process the root node]
   Write (DATA(T))

5. [Finished]
   Return.

(2) RINORDER (T)

1. [check for empty tree]
   if T = NULL
   then Write ('EMPTY TREE')
   Return
2. [Process the left Subtree]
   if LPTR(T) ≠ NULL
   then Call RINORDER (LPTR(T))
3. [Process the root node]
   Write (DATA (T))
4. [Process the right Subtree]
   if RPTR(T) ≠ NULL
   then Call RINORDER (RPTR(T))
5. [finished]
   Return.

(3) RPREORDER (T)

1. [Process the root node]
   if T ≠ NULL
   then Write (DATA(T))
   else Write ('EMPTY TREE')
   Return
2. [Process the left Subtree]

if LPTR(T) ≠ NULL
then Call RPREORDER (LPTR(T))

3. [Process the right Subtree]

if RPTR(T) ≠ NULL
then Call RPREORDER (RPTR(T))

4. [finished]
Return.

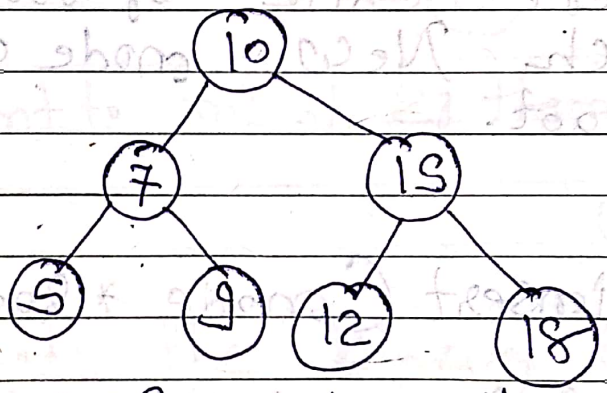
* Binary Search Tree :-

GTU 7 Marks

→ To make the Searching algorithm faster in a binary tree we will go for building the binary Search tree.

→ It is based on binary Search algorithm.

→ Value of left Subtree < Root node value < Right Subtree value.



* Various Operation that can be performed on binary Search tree are:

(1) Insertion of a node in BST.
(2) Deletion " " " "
(3) Searching Elements " "
(4) Display of binary tree.

(1) Insertion of a node in BST.

(1) Read the value for the node which is to be created and store it in a node called Neco.

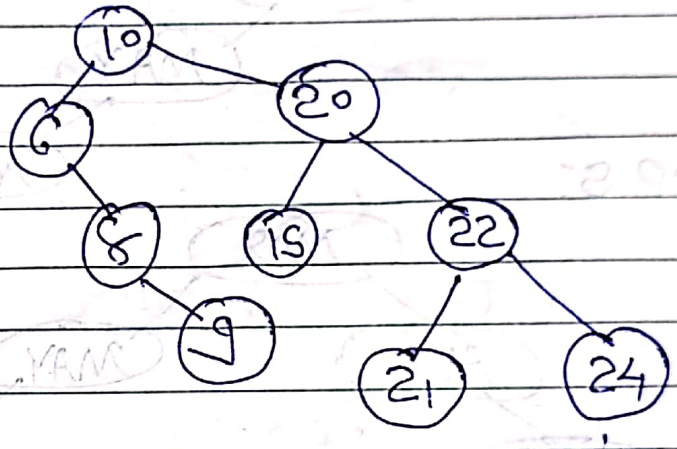
(2) if (root != NULL) then root=Neco;

(3) Again read the next value of node created in Neco.

(4) If (Neco → value < root → value) then attach Neco node as a left child of root otherwise attach Neco node as right child of root.

```
Void insert (node *root, node *Neco)
{
  if (Neco→data < root→data)
  {
    if (root→left == NULL)
      root→left = Neco;
    else
      insert (root→left, Neco);
  }
  if (Neco→data > root→data)
```

```
{
    if(root -> right == NULL)
        root -> right = New;
    else
        insert (root -> right, New);
}
```



if we want to insert 23 then

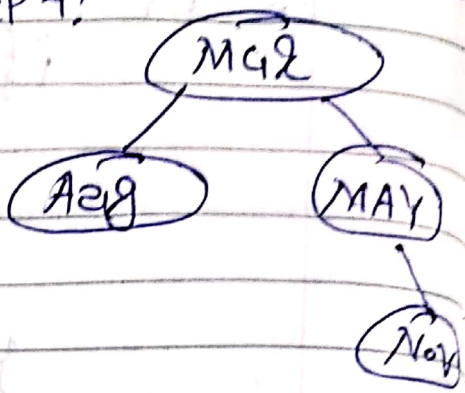
Ex: Construct the binary search tree from the following set of strings.

Mar, May, Nov, Aug, Apr, Jan, Dec, Jul, Feb, Jun, Oct and Sep.

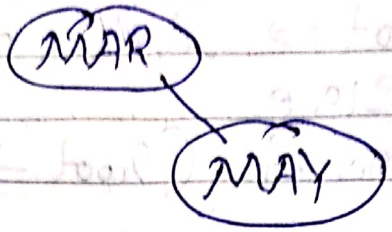
Step 1:

Page
Date

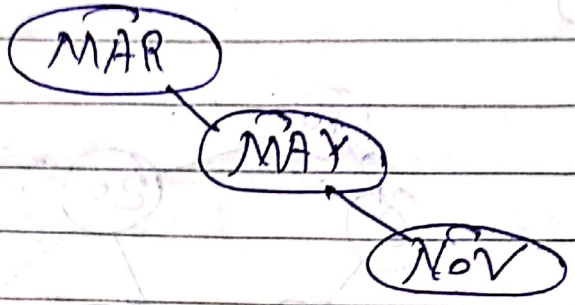
Step 1:



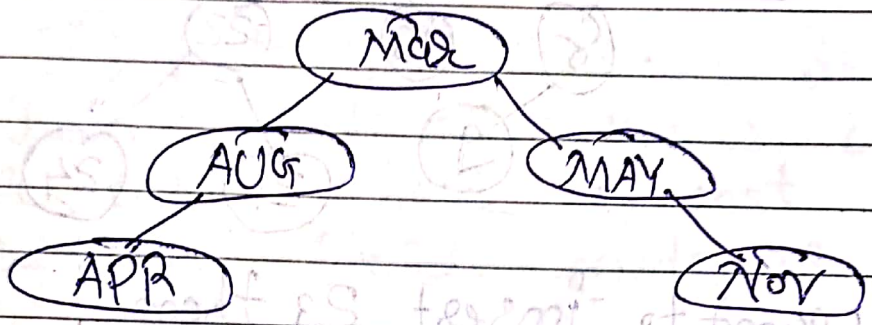
Step 2:



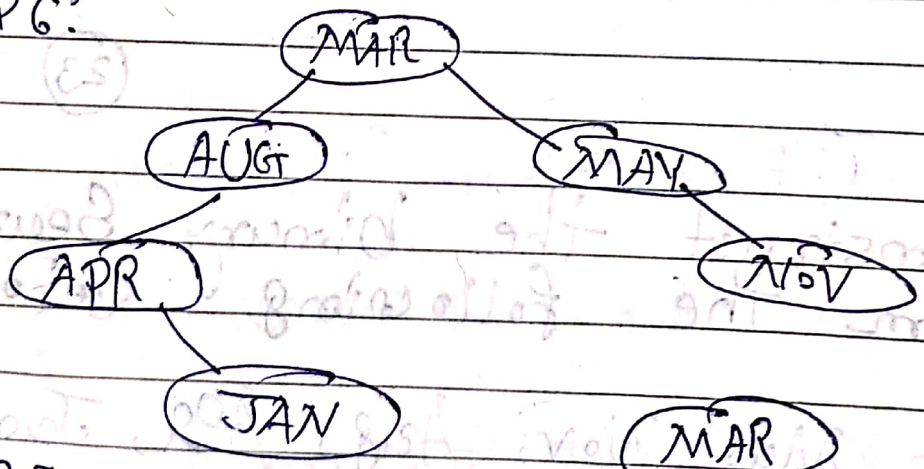
Step 3:



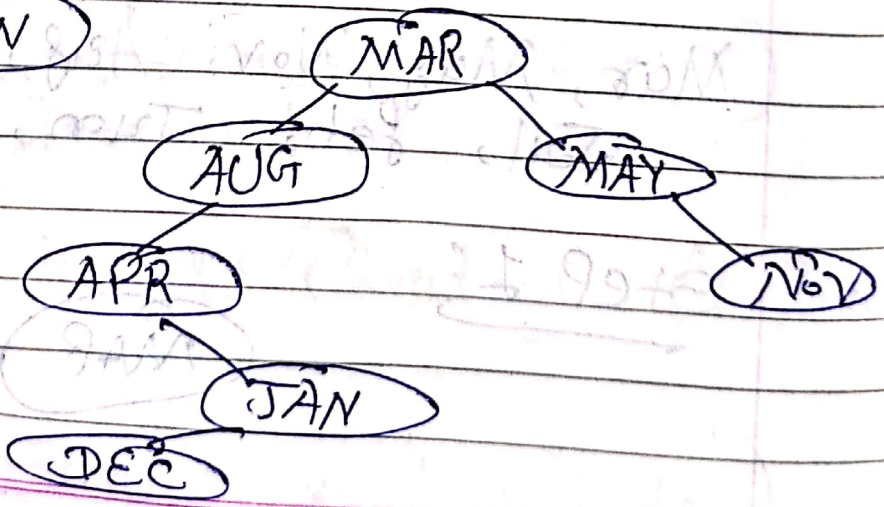
Step 5:



Step 6:

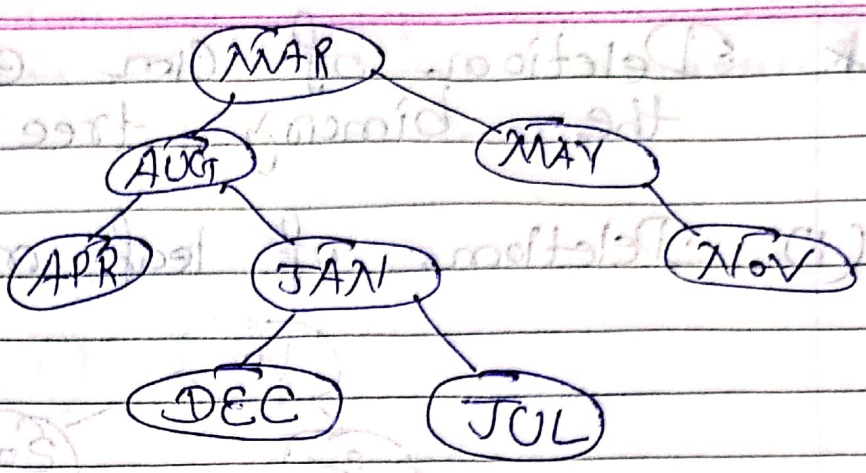


Step 7

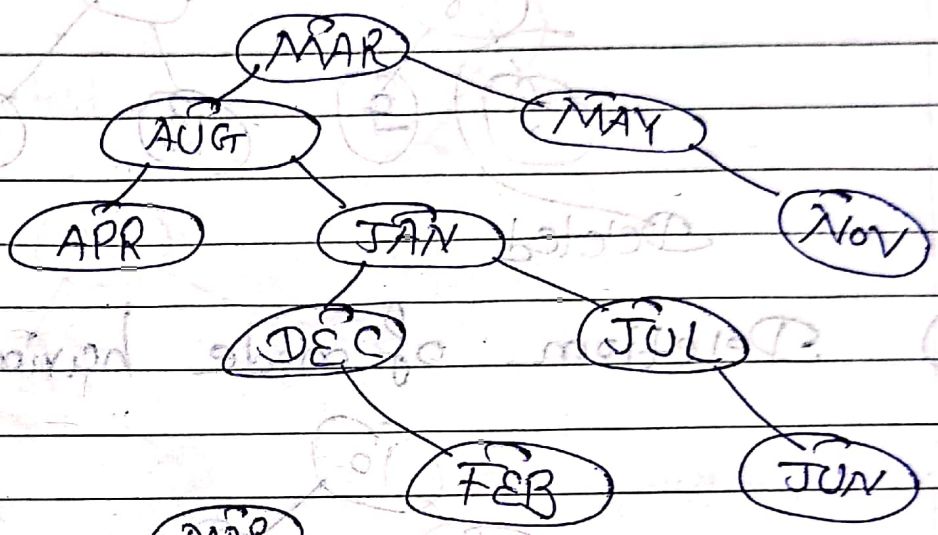


Page
Date

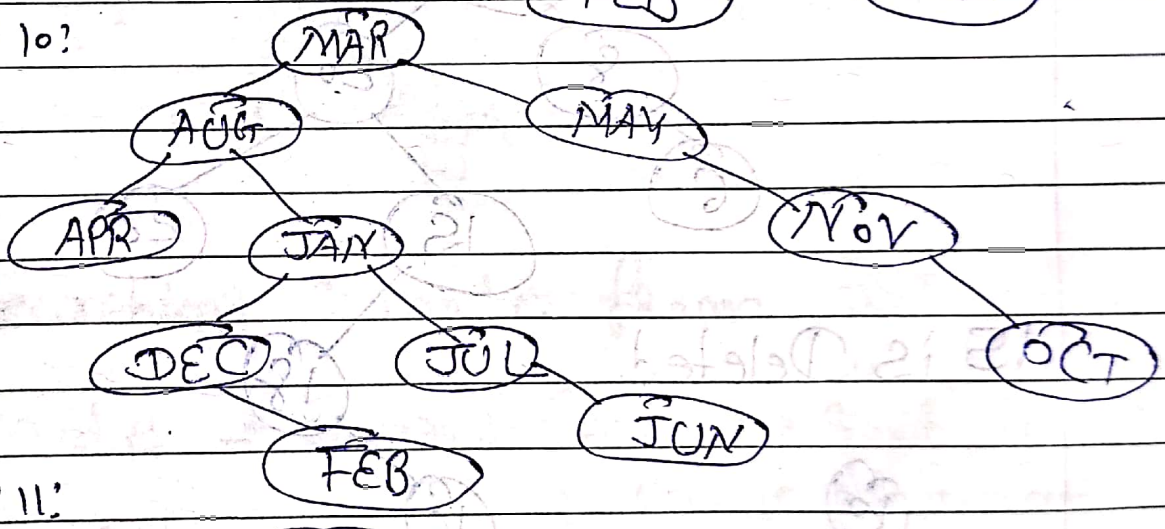
Step 8:



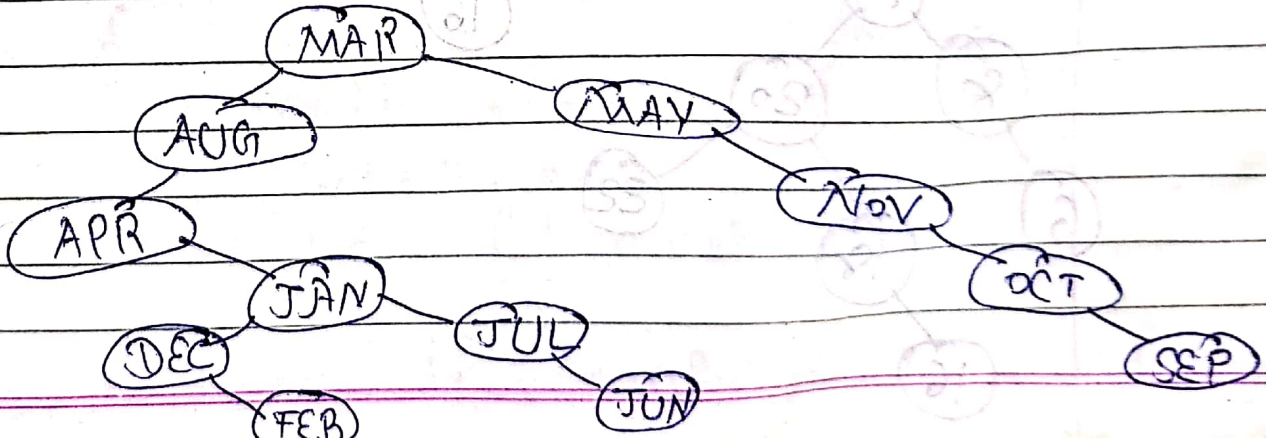
Step 9:



Step 10:

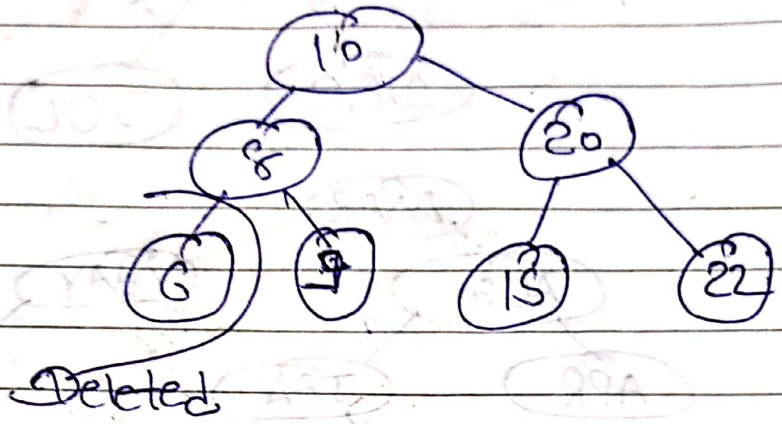


Step 11:



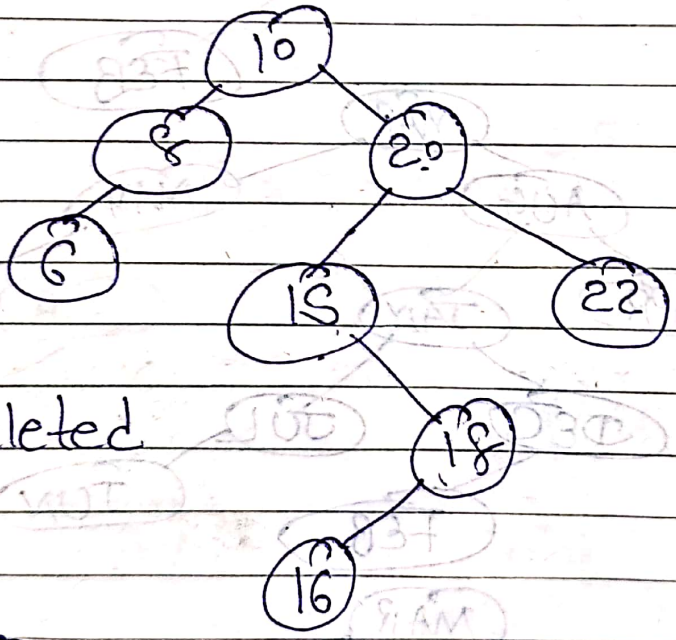
* Deletion of an element from the binary tree:

(1) Deletion of leaf node

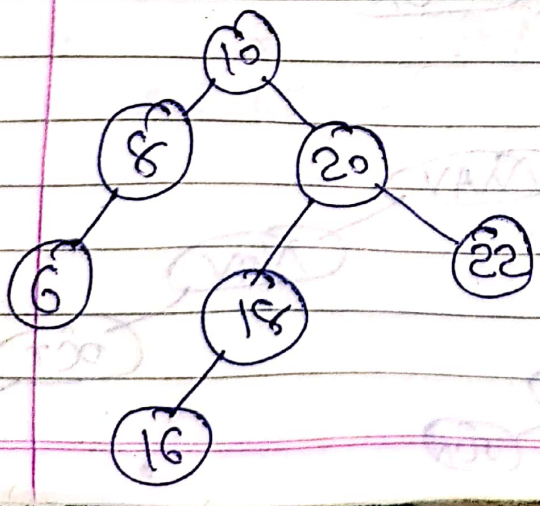


6 is Deleted

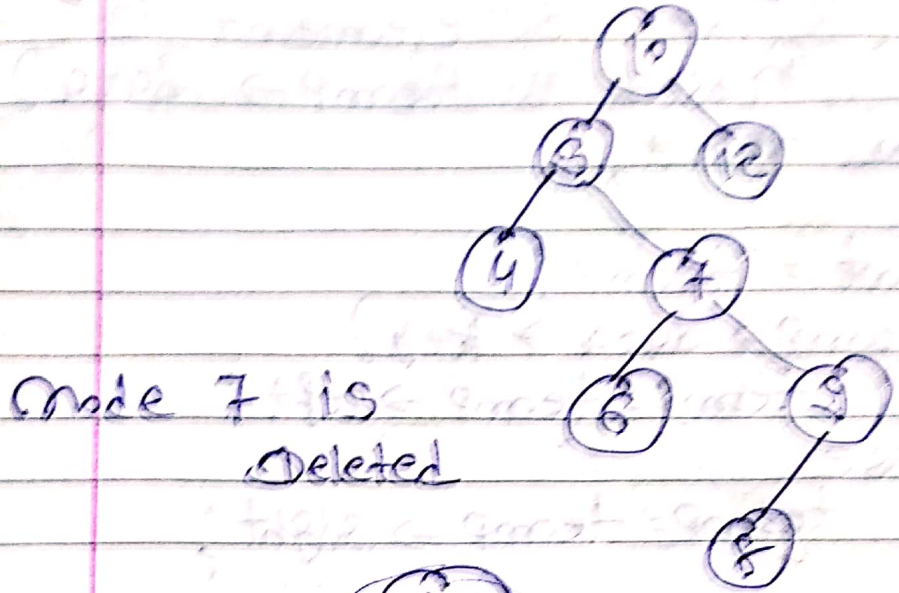
(2) Deletion of node having one child.



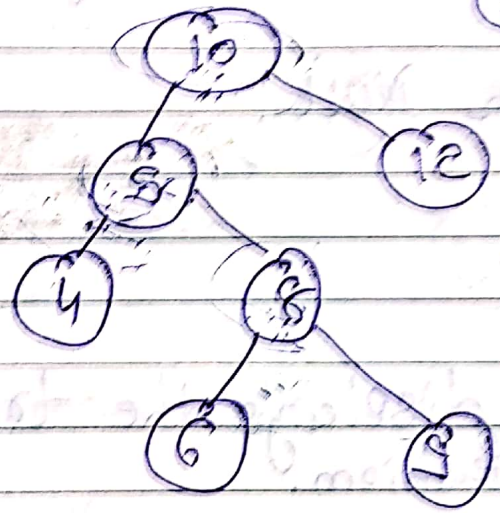
15 is Deleted



(3) The node having two Children.



Node 7 is Deleted



* Searching a node from BST.

```
node * Search (node *root, int key,
                node **Parent)
{
    node *temp;
    temp = root;
    while (temp != NULL)
    {
        if (temp -> data == key)
```

```
{
  Printf("The %d element is
          Present", temp→data);
  return temp;
}
```

```
* Parent = temp;
  if (temp→data > key)
        temp = temp→left;
  else
        temp = temp→right;
  }
  return NULL;
}
```

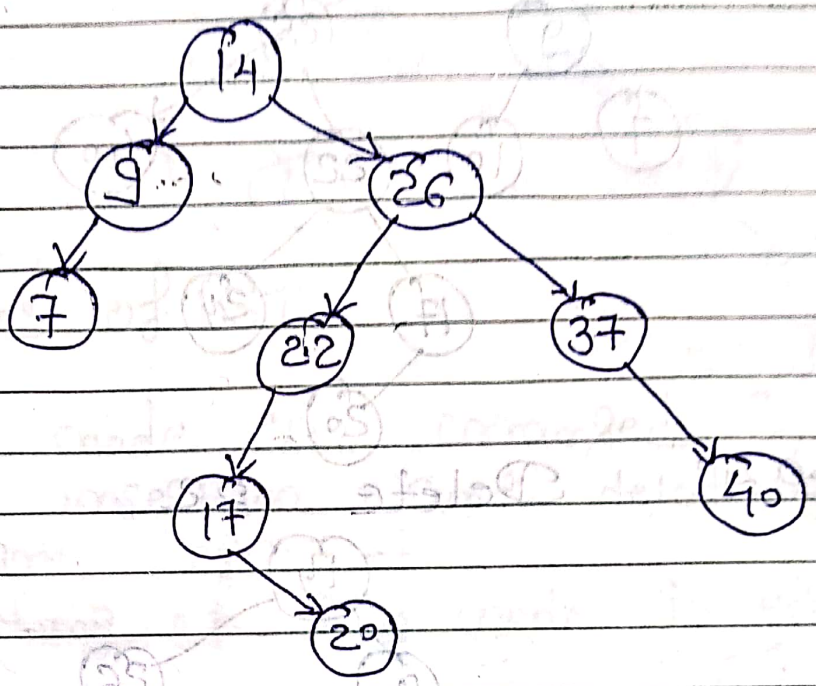
* Disply BST

⇒ This fun displays the tree in inorder fashion.

```
Void inorder (node *temp)
{
  if (temp != NULL)
  {
    inorder (temp→left);
    Printf ("%d", temp→data);
    inorder (temp→right);
  }
}
```

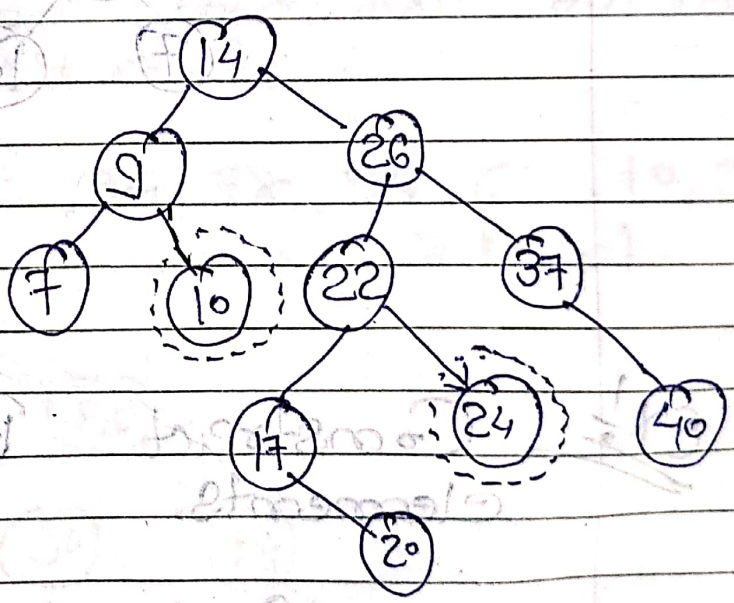
Ex1, First Insert 10 & 24, then Delete 37 & 22 from following BST. Draw each operation

GTU 7 marks



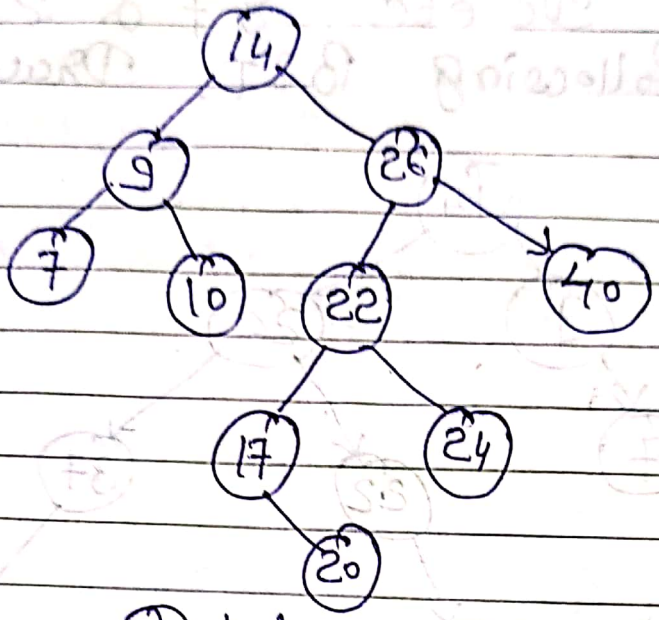
Step 1:

Insert 10

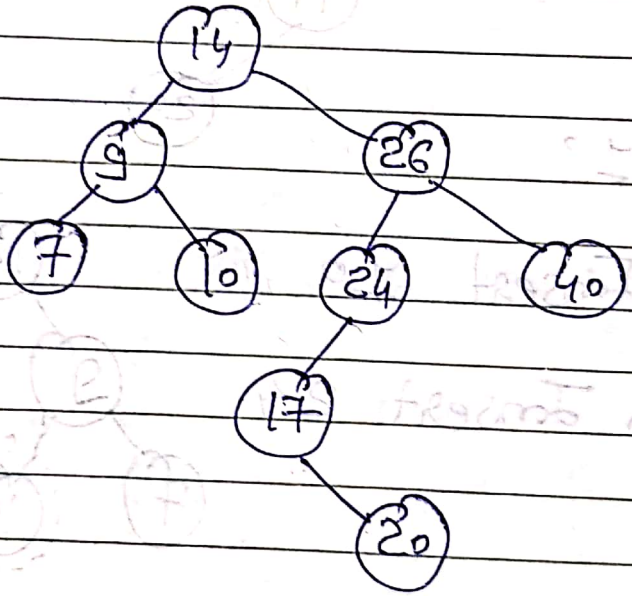


Step 2: then Insert 24

Step 3: Delete 37

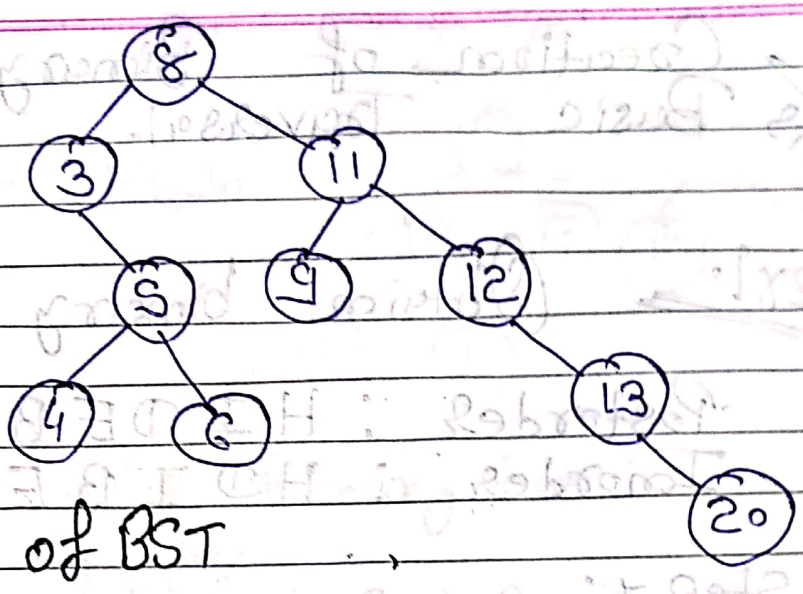


Step 4: Delete 22



Ex:- Construct BST for the following elements.

8, 3, 11, 5, 9, 12, 13, 4, 6, 20

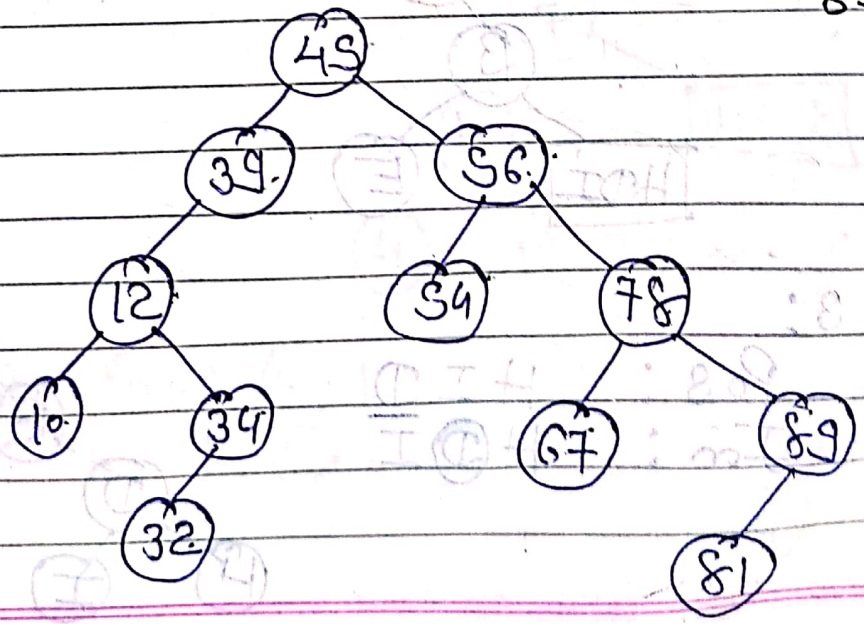


* Advantages of BST :-

(1) The node are arranged systematically.
(2) The insertion and deletion operation are efficient.
(3) Searching of any node is efficient.

Ex! Construct a BST also write Inorder, Preorder, Postorder

45, 56, 39, 12, 34, 78, 54, 67, 10, 32, 89, 81.



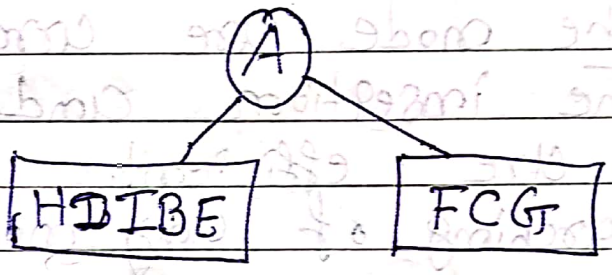
GTU 7 Marks

* Creation of Binary Tree from Basic Traversal.

Ex1: Obtain binary tree from -

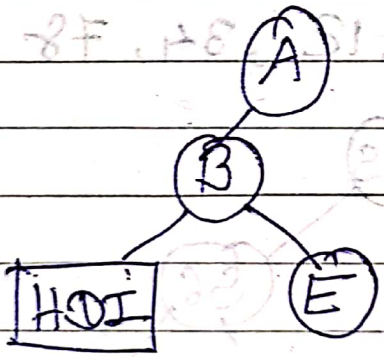
Postorder : H I D E B F G C A
Inorder : H D I B E A F C G

Step 1:



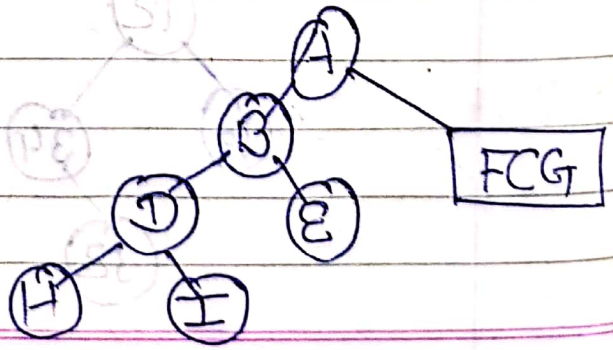
Step 2:

Pos : H I D E B
In : H D I B E



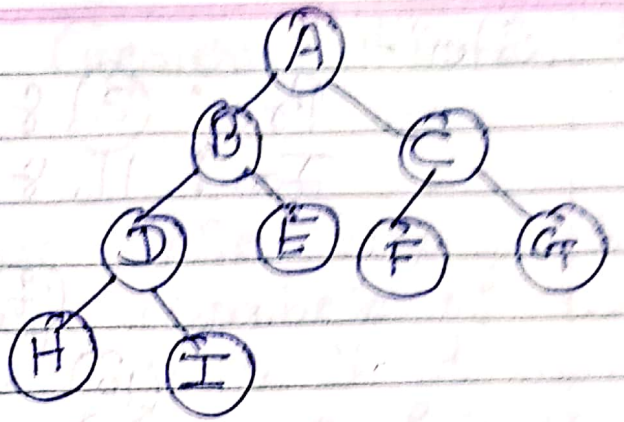
Step 3:

Pos : H I D
In : H D I



Step 4:

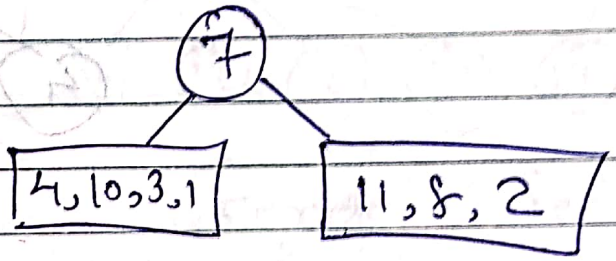
Pos: F G C
In: F C G



Ex: Create Binary tree:

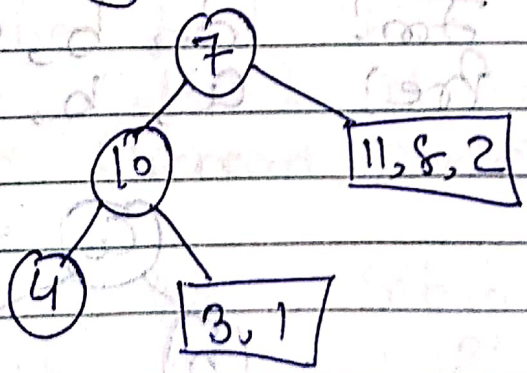
Preorder = { 7, 10, 4, 3, 1, 2, 8, 11 } (7 = root)
Inorder = { 4, 10, 3, 1, 7, 11, 8, 2 }

Step 1

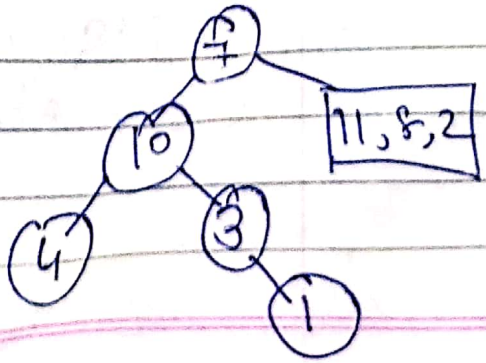


Step 2

Pre: 10, 4, 3, 1
In: 4, 10, 3, 1



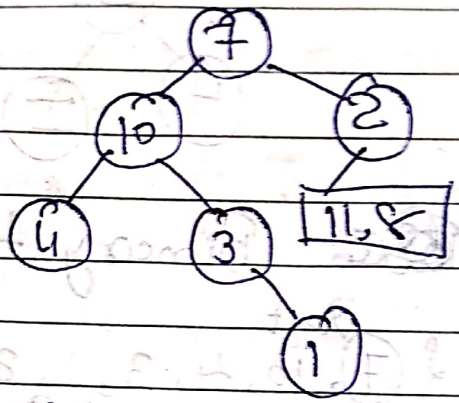
Step 3 Pre: 3, 1
In: 3, 1



Step 4 :

Pre : 2, 8, 11

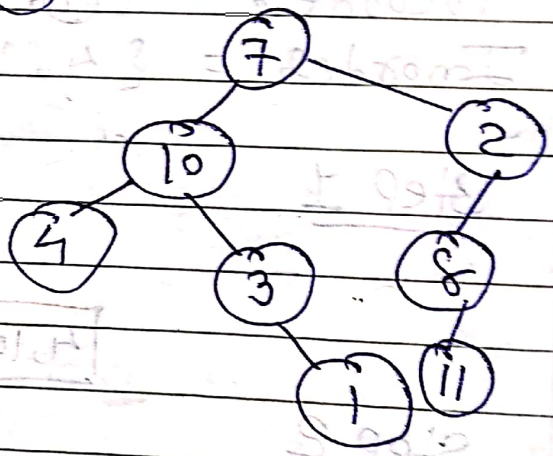
In : 11, 8, 2



Step 5 :

Pre : 8, 11

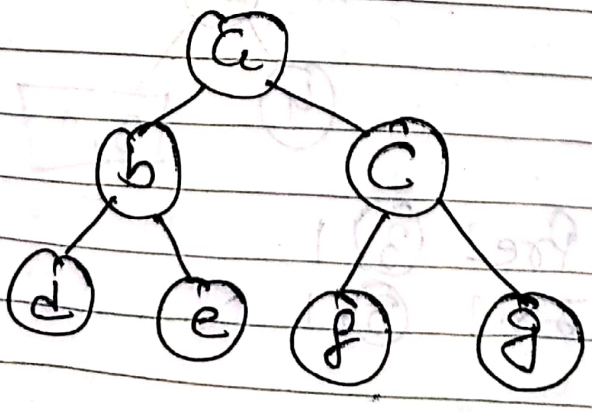
In : 11, 8



Ex: Create Binary tree!

In : d, b, e, a, f, c, g

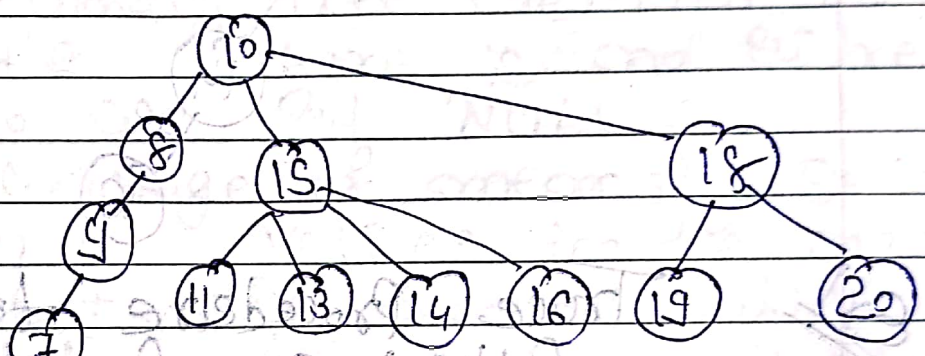
Pre : a, b, d, e, c, f, g



* Conversion of General Trees to Binary Trees:

→ General trees are those tree in which there could be any member of nodes that can be attached as a child nodes.

→ The member of subtrees for each node may be varying.

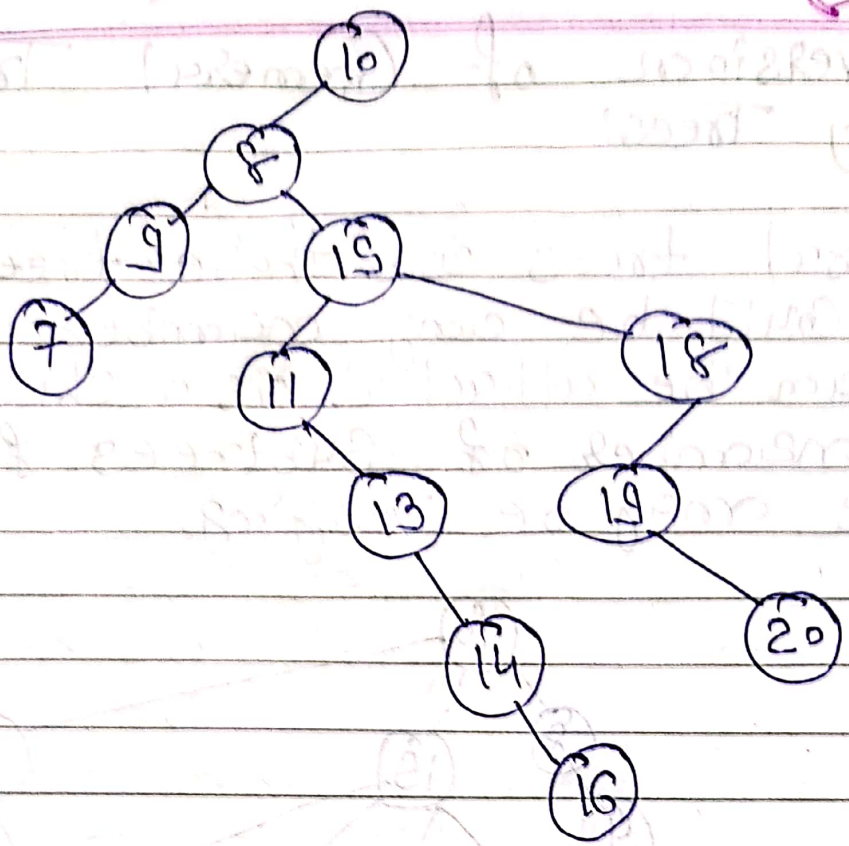


General Tree

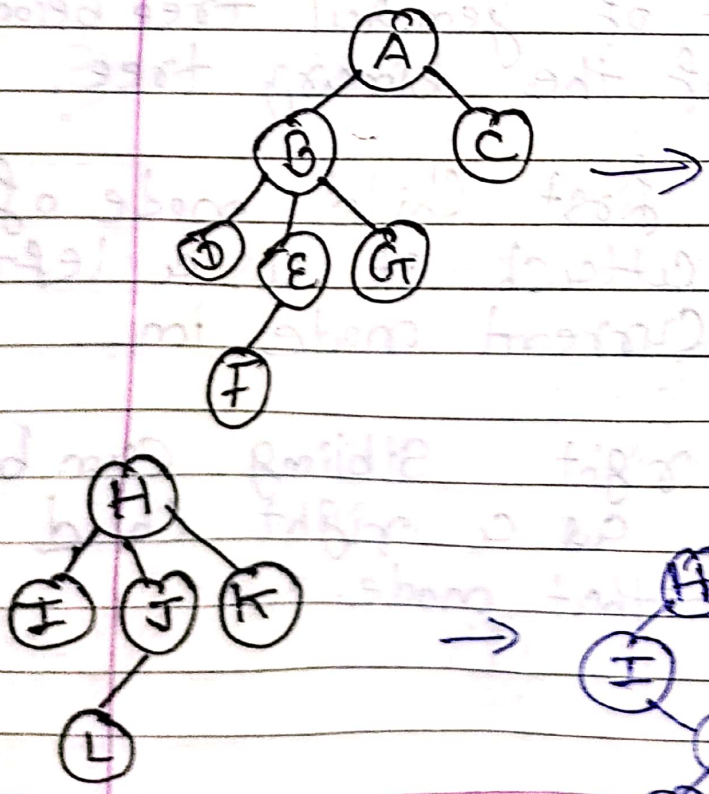
Rule 1: The root of general tree become the root of the binary tree.

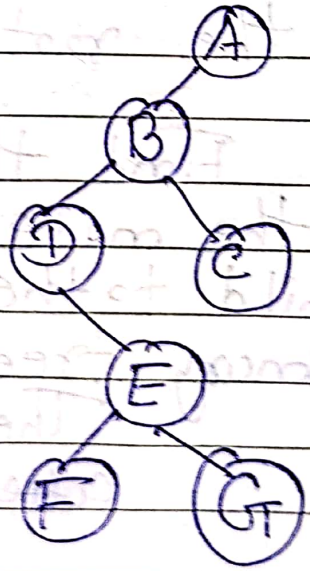
Rule 2: Find the first child node of the node attach it as a left child to the current node in binary tree.

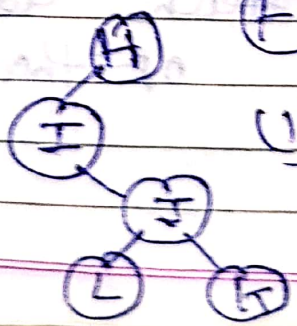
Rule 3: The right sibling can be attach as a right child of that node.



Ex) Trace Procedure to convert following forest into binary tree.







(1) left child of node become left child of that node

(2) The Sibling of a node becomes its right Child.

* Conversion of General Trees to Bi

Threaded Binary Trees :-

→ In Binary tree. Creation for the leaf node there is no subtree further so we Put NULL.

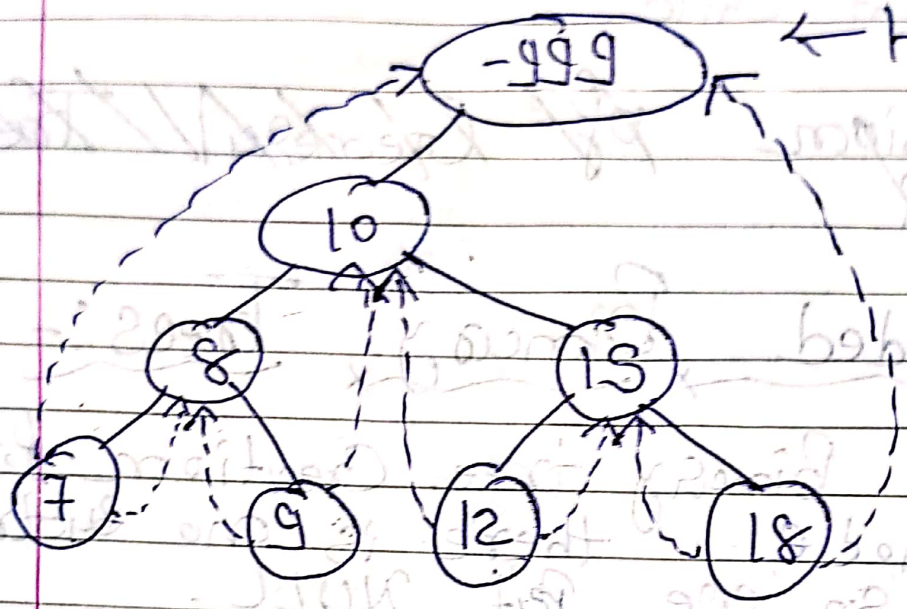
→ It is wastage of memory so, to avoid NULL values in the node we first set the threads which are actually the links to the Predecessor and Successor nodes.

→ Three types of threading is Possible Inorder, Preorder and Postorder.

→ We will discuss inorder threading.

→ Here left thread Should Point to the Predecessor and right thread Points to the Successor.

→ Assume Head node is starting node. and root node of tree is attach to left of head node.


-999
Head node
10
8
15
7
9
12
18

Page
Date

* Application of Trees!

→ The Purpose of binary tree is to have efficient retrieval of data.

→ It is a useful data structure when we require to take two-way decisions.

→ The use of binary tree is exploited by following data structure.

(1) BST (2) Expression trees
(3) Threaded binary trees.
(4) Game tree.

* Concept of Balanced Trees!

→ Balanced trees are useful data structures that are useful to store the data at some specific location.

Def → A balanced tree is a rooted tree where each subtree of the root has equal number of nodes.

→ The height of such binary tree is O(logN)

Many types of balanced tree is

(1) Height balanced tree or AVL-tree

(2) Splay tree

(3) Red-black trees.

(4) B-trees.

* AVL Trees:-

Def An empty tree is height balanced if T is a non-empty binary tree with T_L and T_R as its left and Right Subtrees. The T is height balanced if and only if

i) T_L and T_R are height balanced

ii) $h_L - h_R <= 1$ where h_L and h_R are height of T_L and T_R.

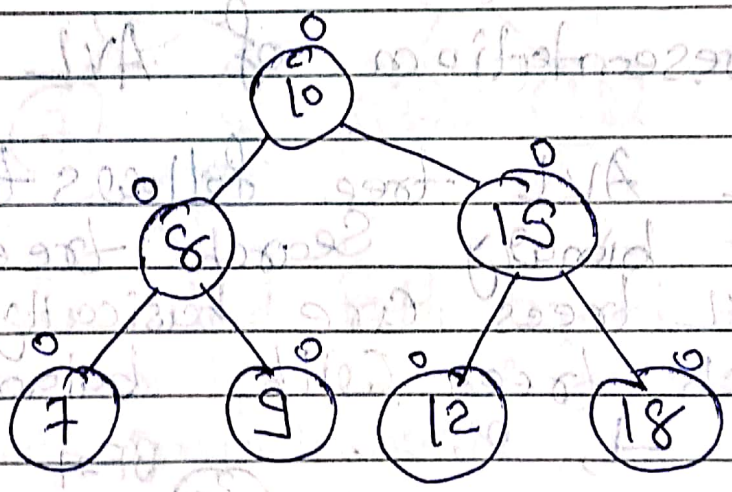
The idea of balancing a tree is obtained by calculating the balance factor of a tree.

Defination of Balance factor

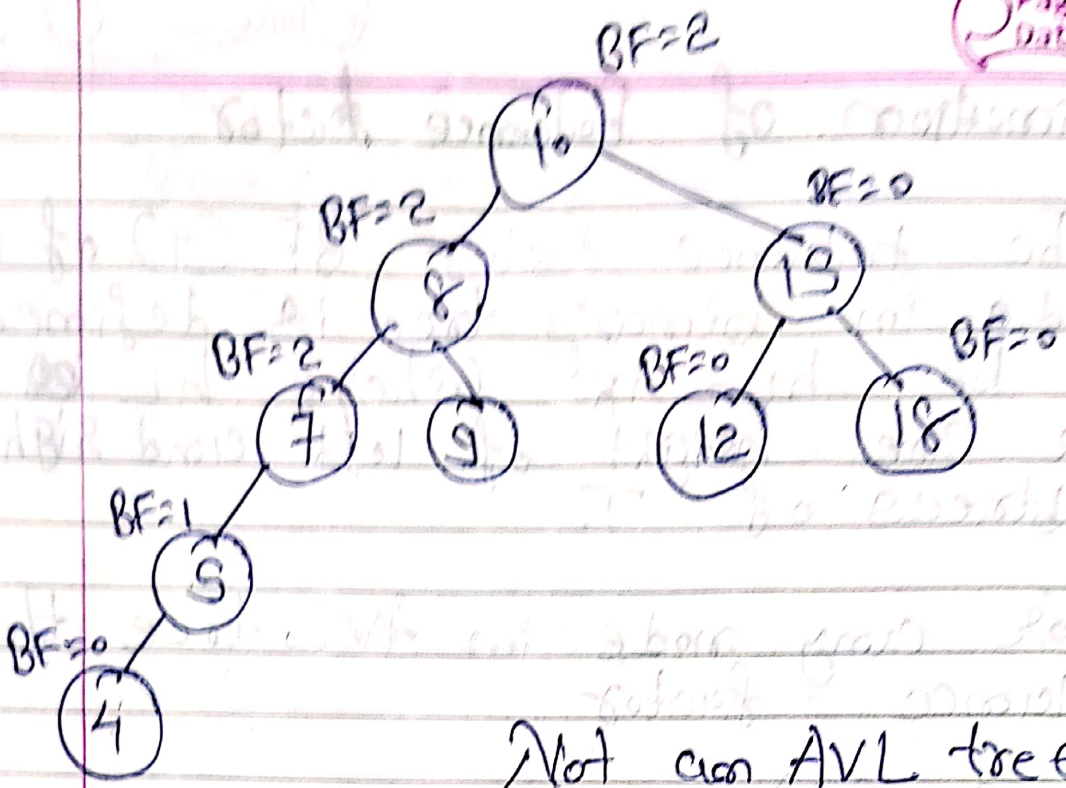
→ The balance factor BF (T) of a node in binary tree is defined to be $h_L - h_R$ where h_L & h_R are height of left and right subtrees of T.

* For any node in AVL tree the balance factor

BF(T) is -1, 0 or +1.



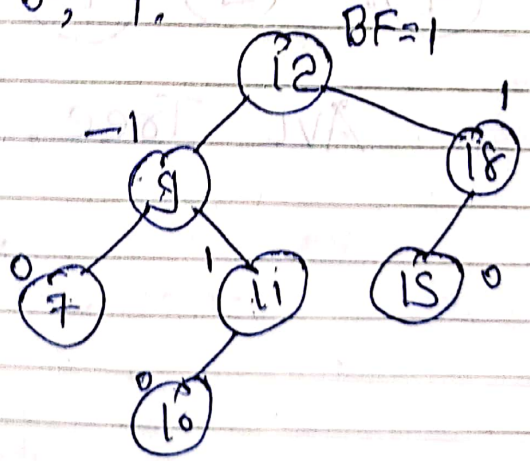
AVL Tree



Not an AVL tree.

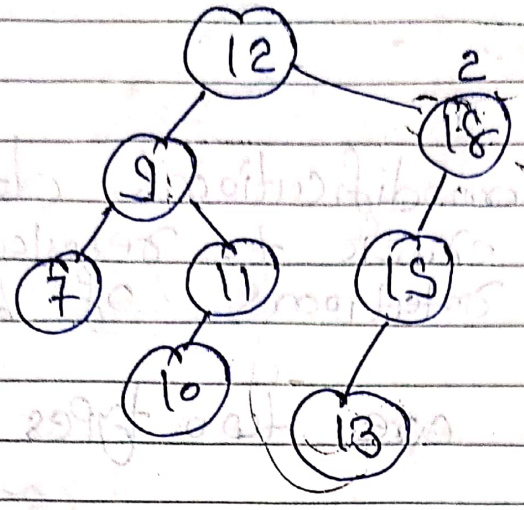
* Representation of AVL Tree.

⇒ The AVL tree follows the property of binary Search tree. Infact AVL trees are basically binary Search tree with balance factor as −1, 0, 1.



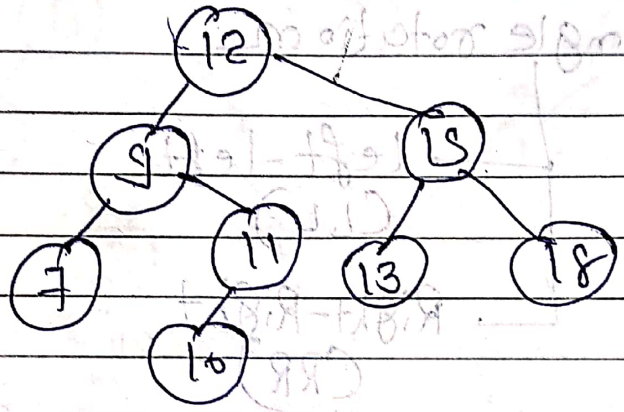
AVL tree.

Insert 13.



Property violated

So,



- Node on that Path need to be readjust
- means only the affected Subtree is to be rebalanced.

* Insertion

There are four different cases when rebalancing is required after insertion of new node.

1. An insertion of new node into left Subtree of left child (LL)

2. (LR)
3. (RL)
4. (RR).

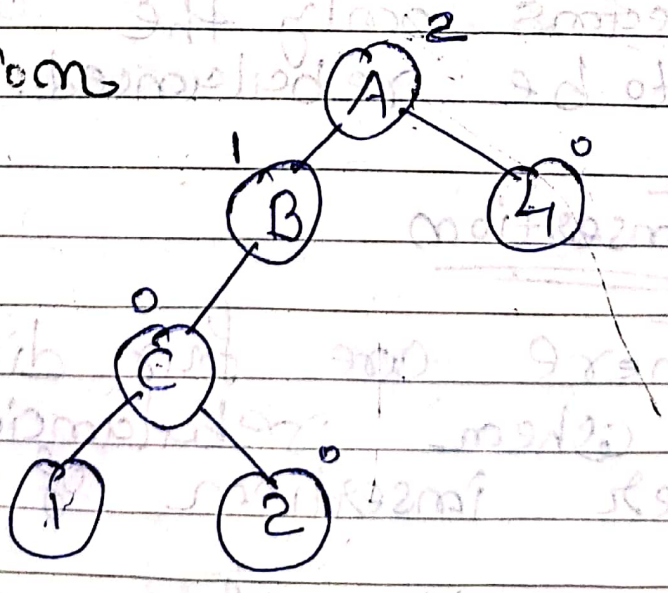
* Some modifications done on AVL tree in order to rebalance it is called rotations of AVL tree.

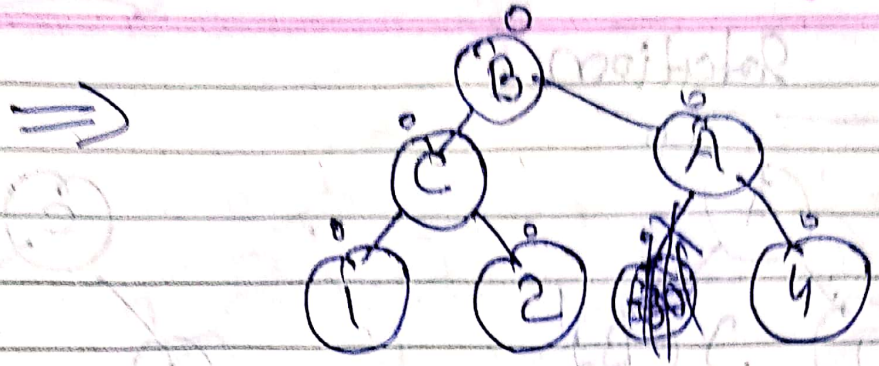
* There are two types of rotations.

Single rotation
- left-left (LL)
- Right-Right (RR)

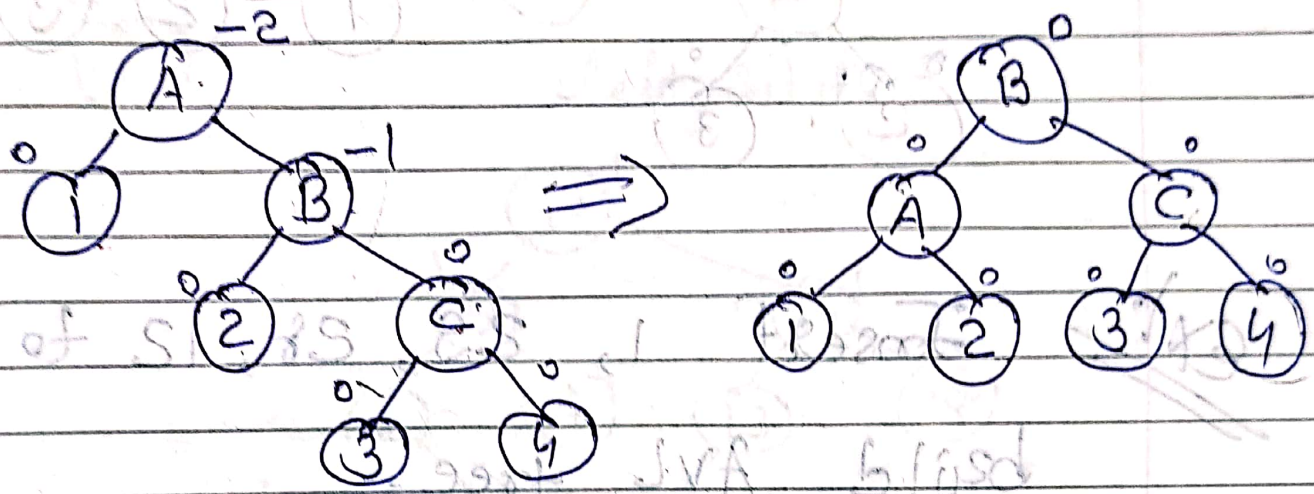
Double rotation
- left-Right (LR)
- Right-left (RL)

(1) LL Rotation

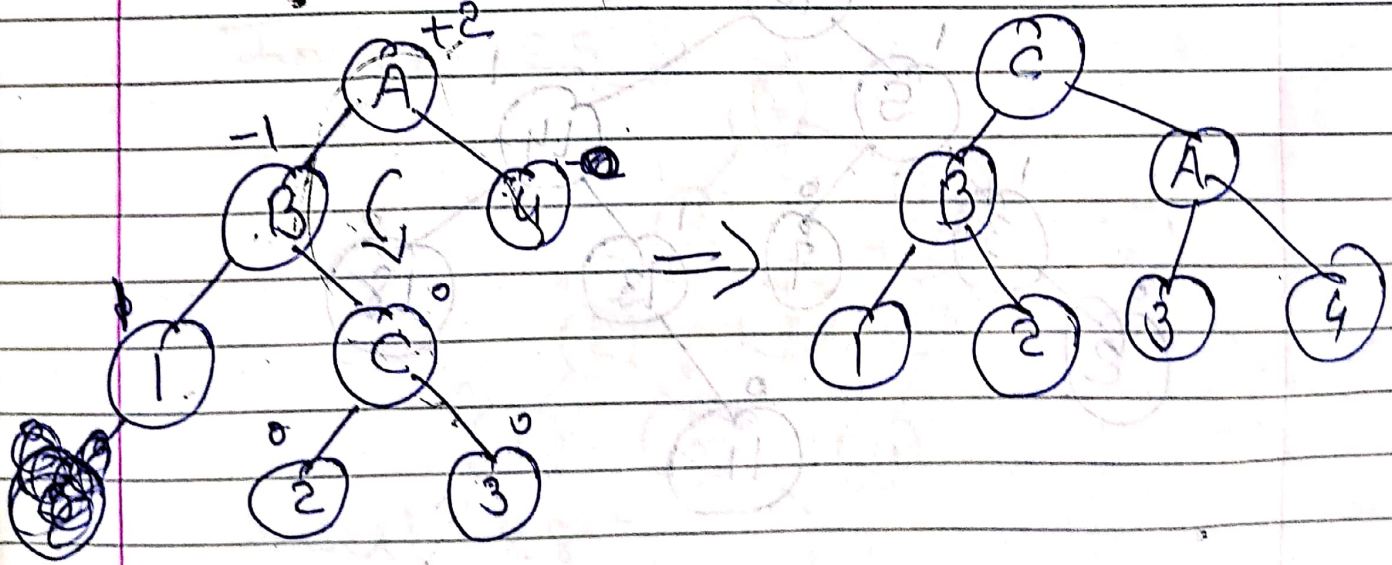




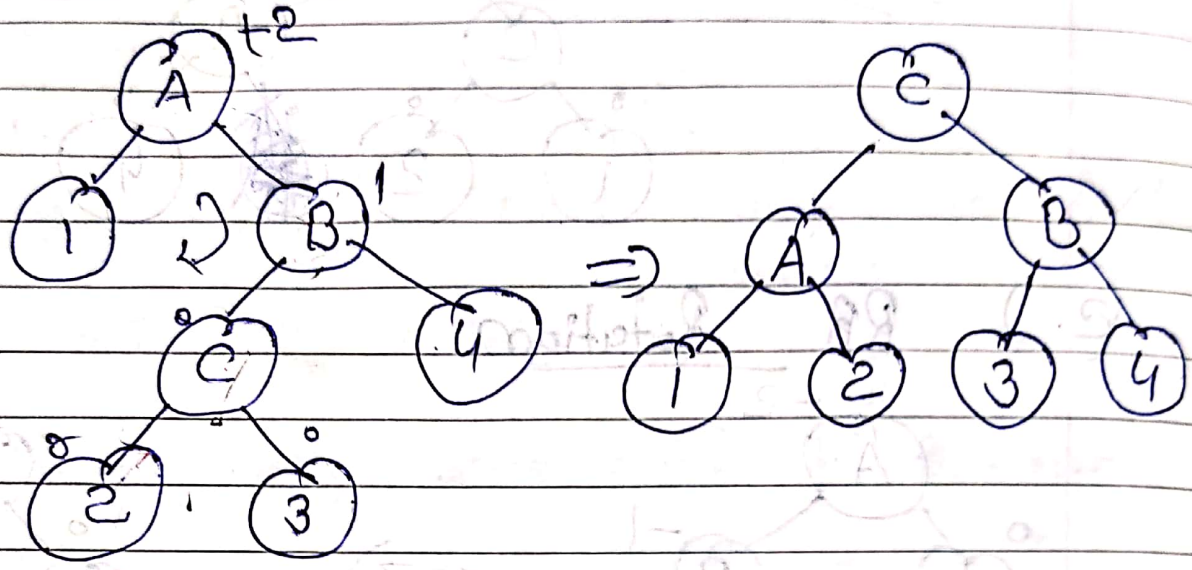
(2) RR Rotation



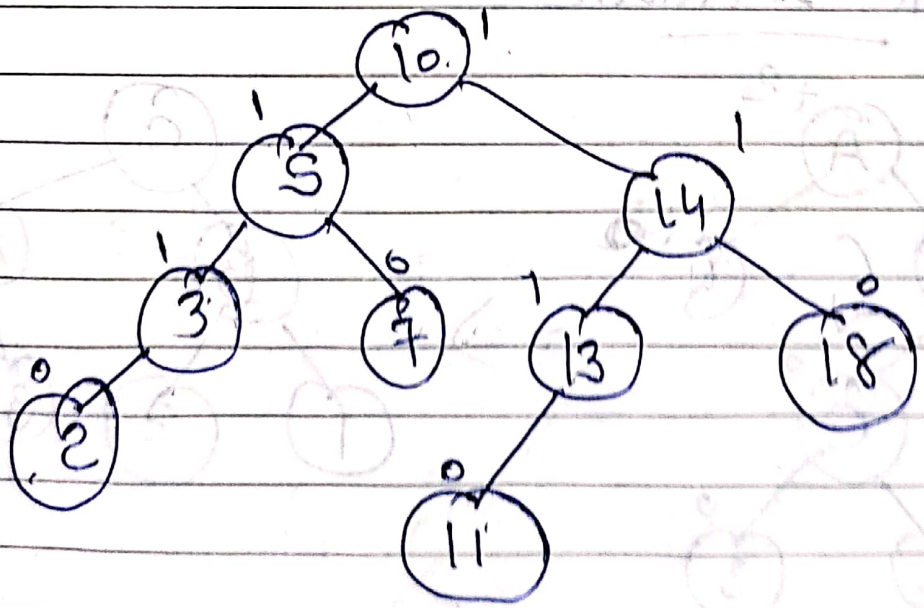
(3) LR Rotation



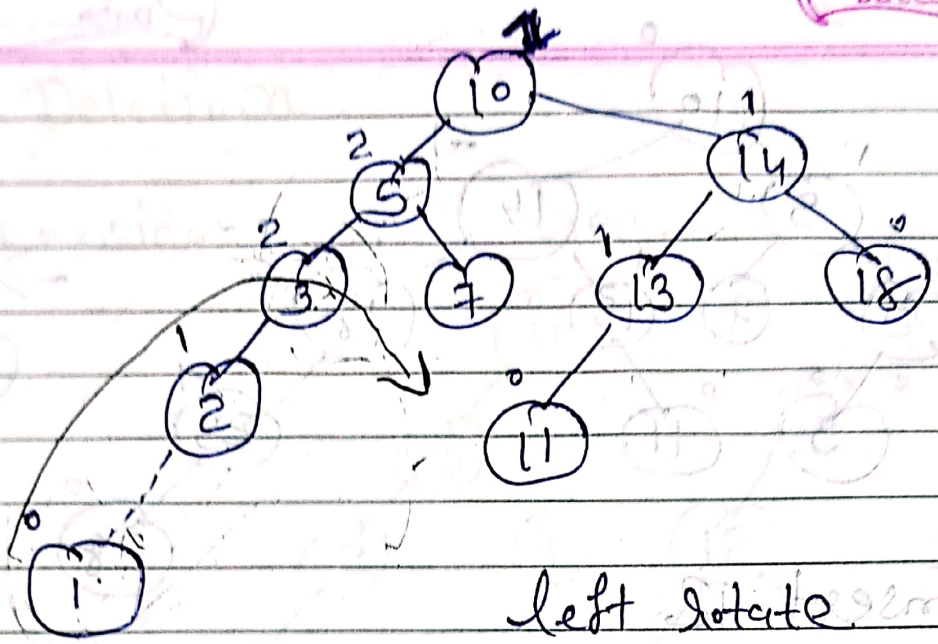
(4) RL Rotation



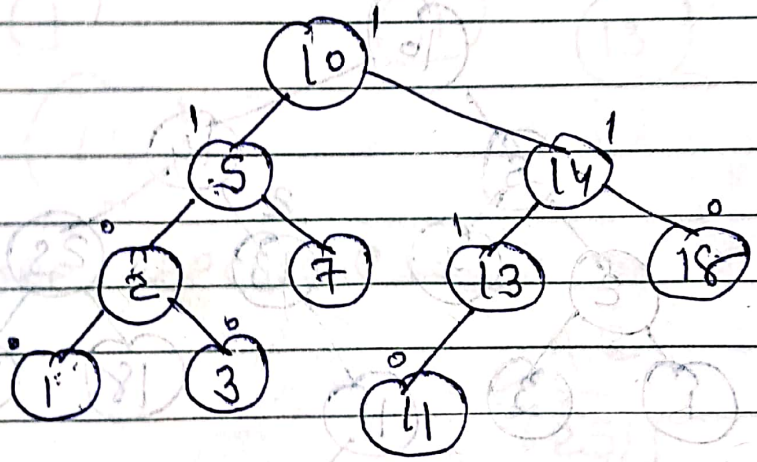
Ex:- Insert 1, 25, 28, 12 to build AVL tree.



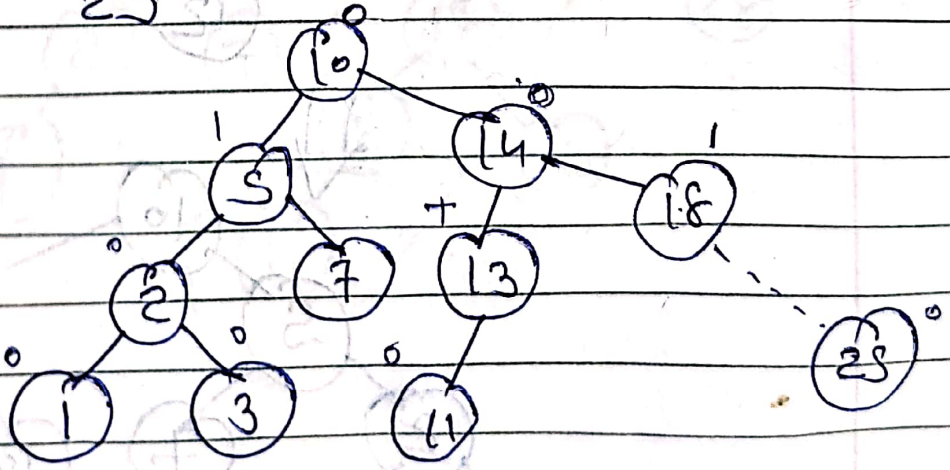
Insert 1



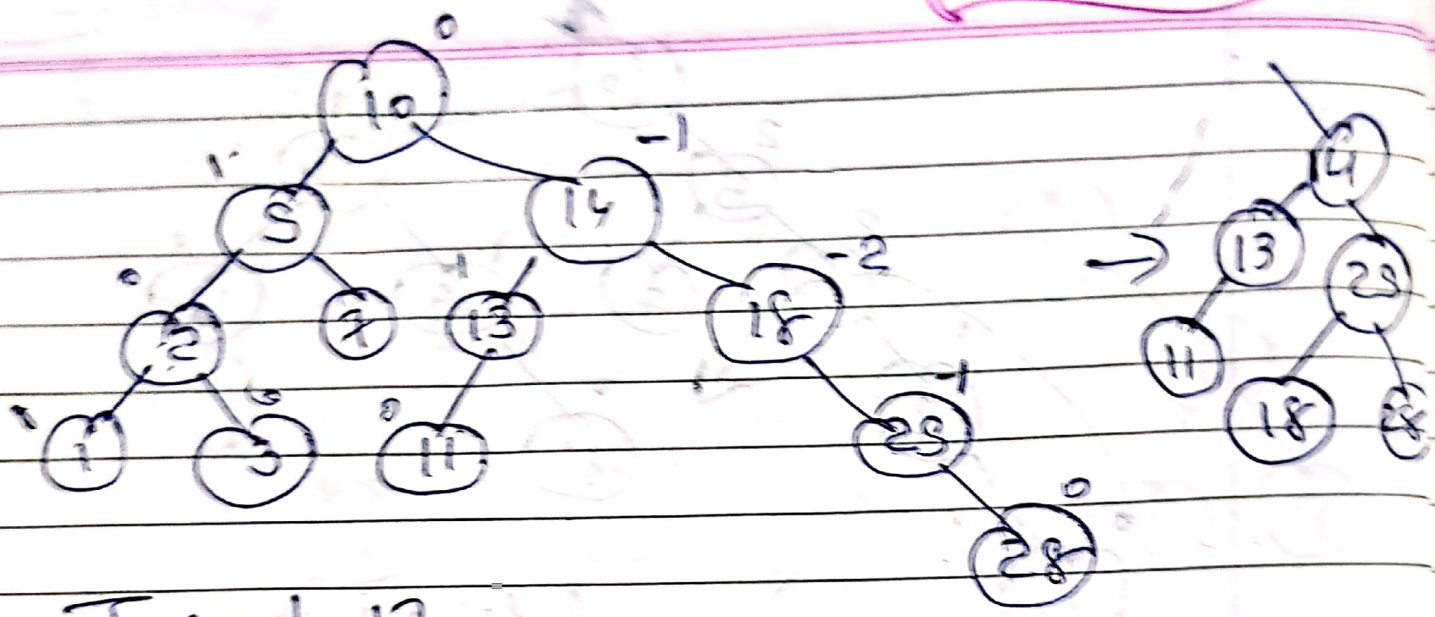
left rotate



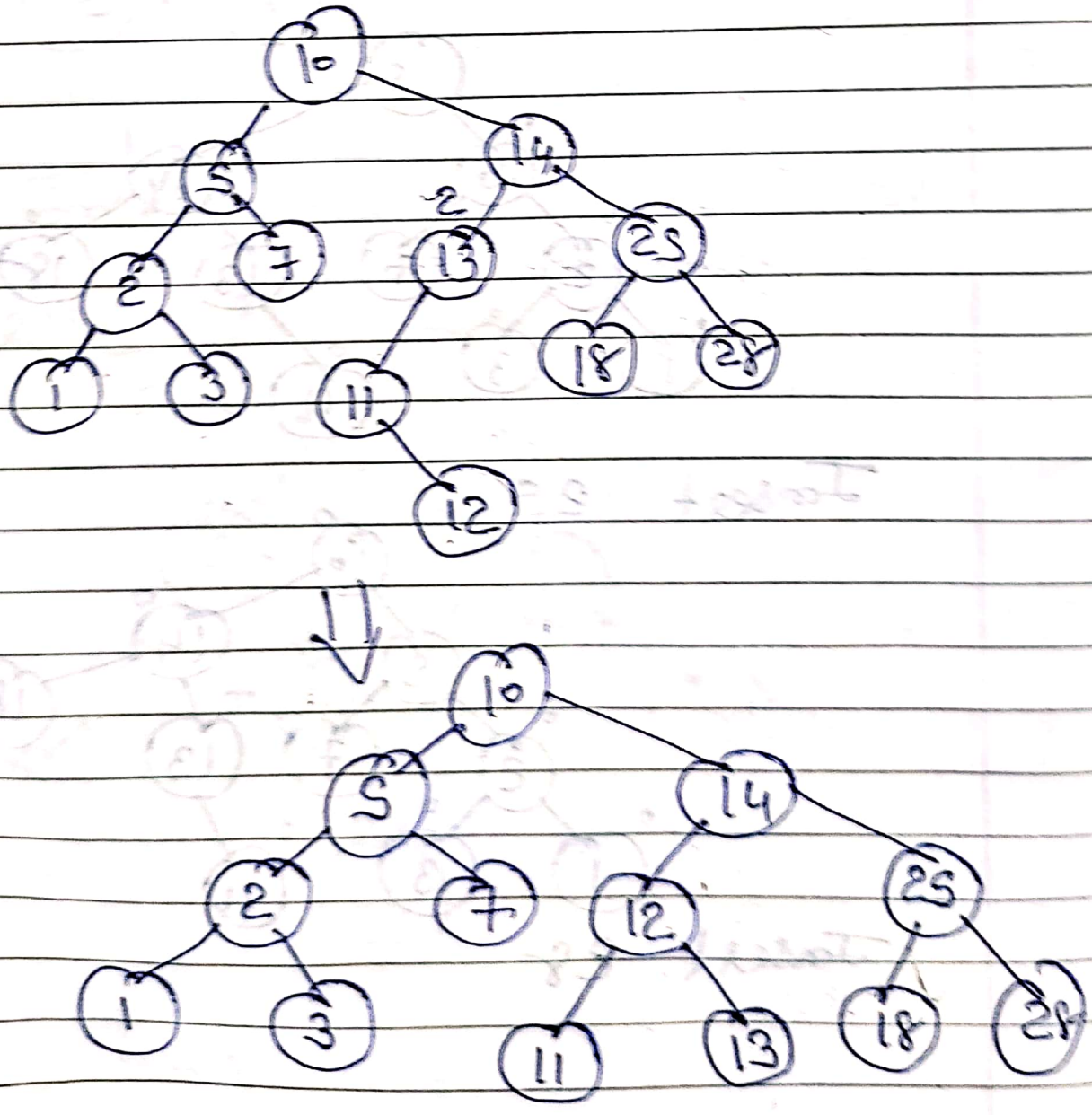
Insert 25



Insert 28

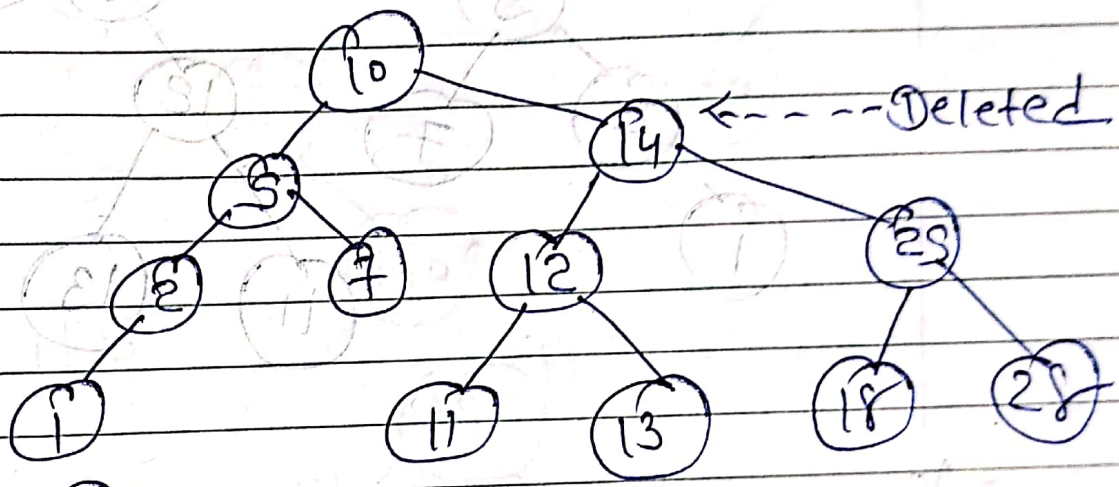


Insert 12



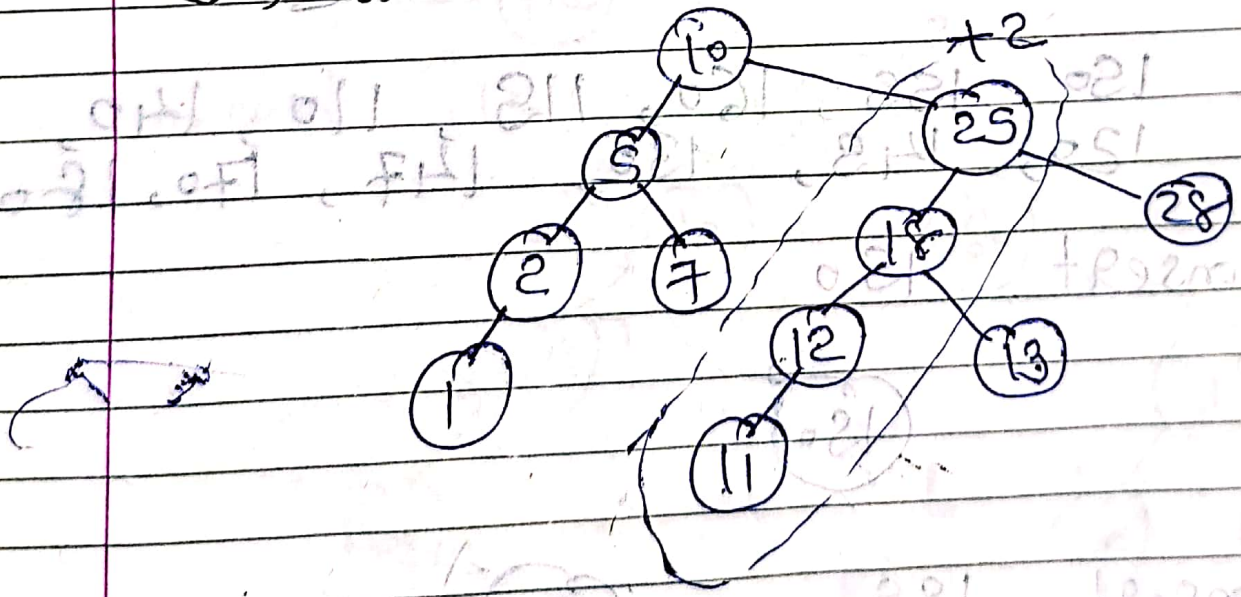
* Deletion

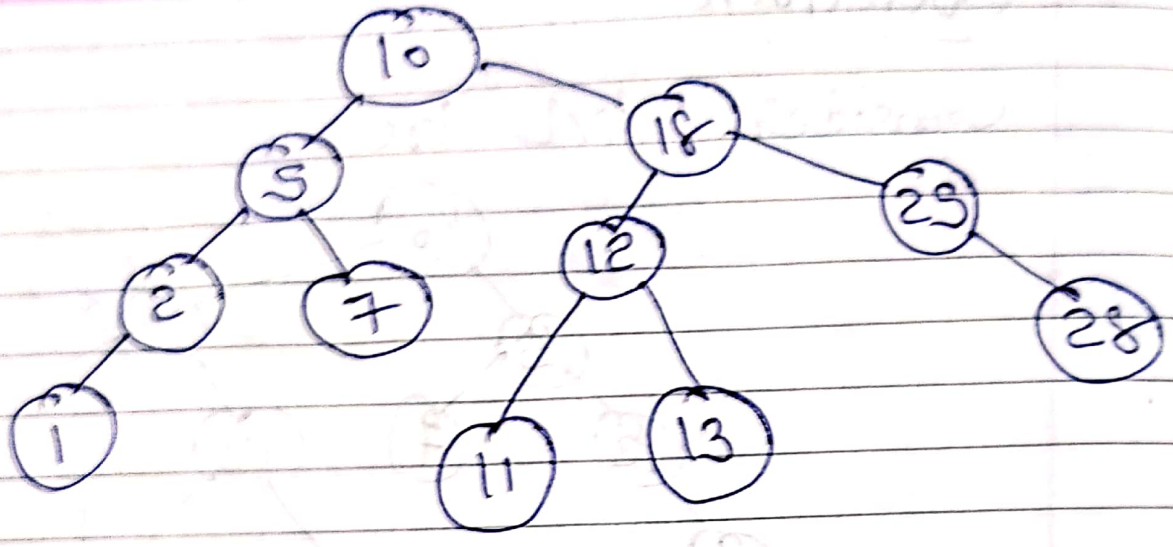
Consider AVL tree.



Delete 14.

So, tree become



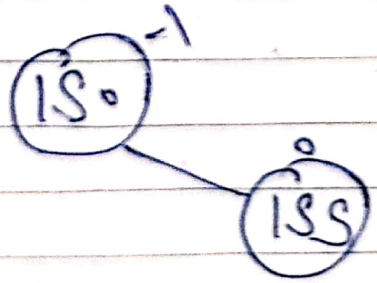


Ex: Define an AVL tree. Obtain AVL using following node insertion in sequence

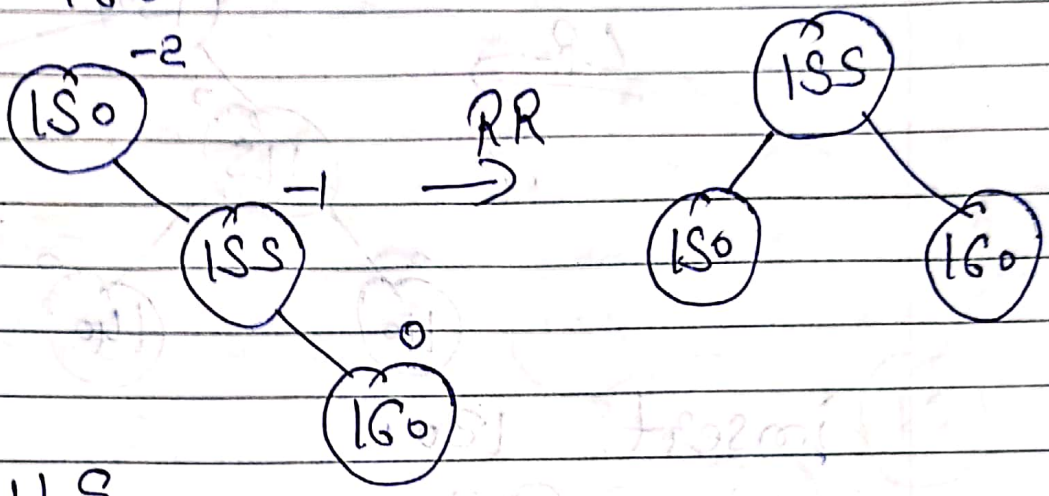
150, 155, 160, 115, 110, 140, 120, 148, 130, 147, 170, 180.

insert 150

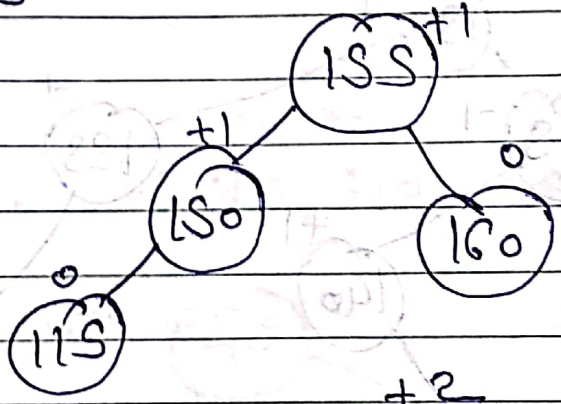
insert 155



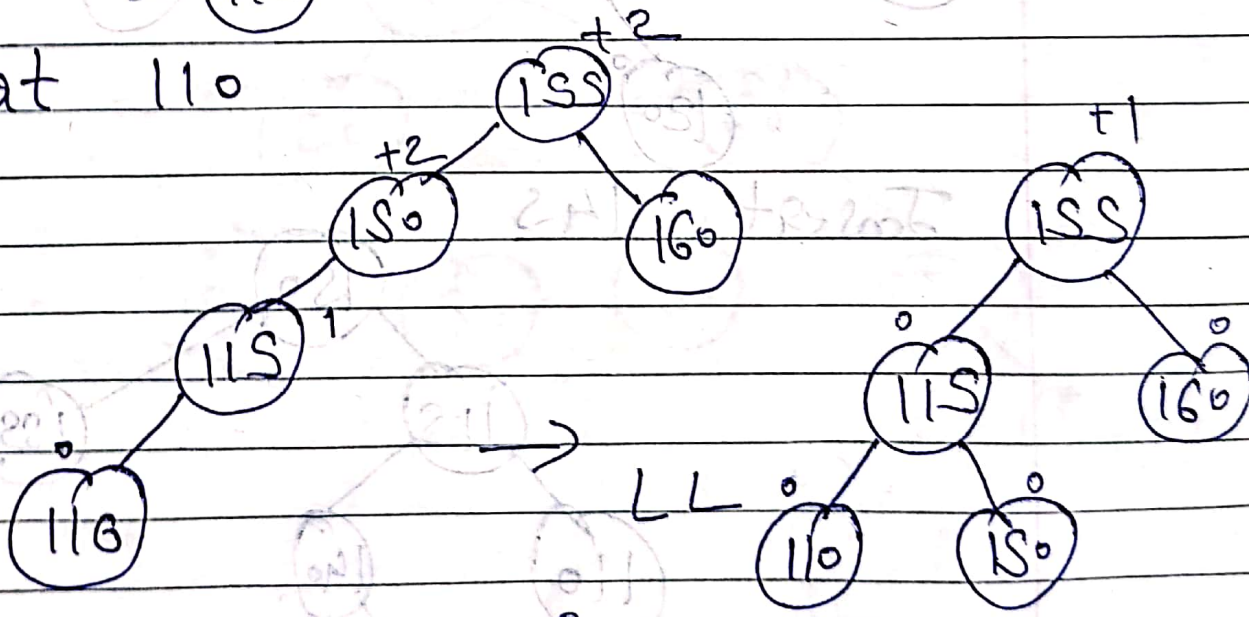
insert 160



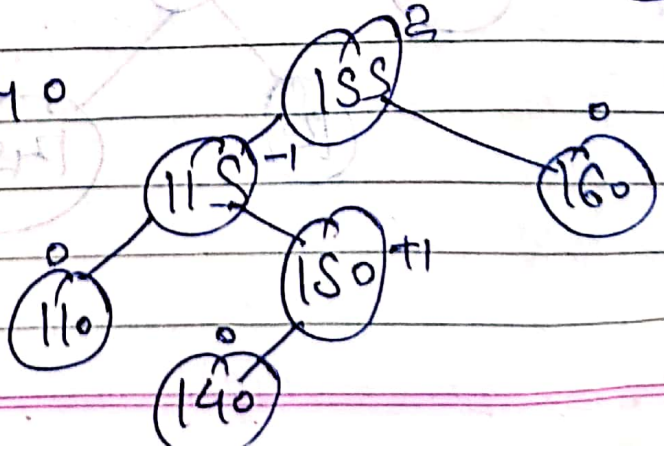
insert 115

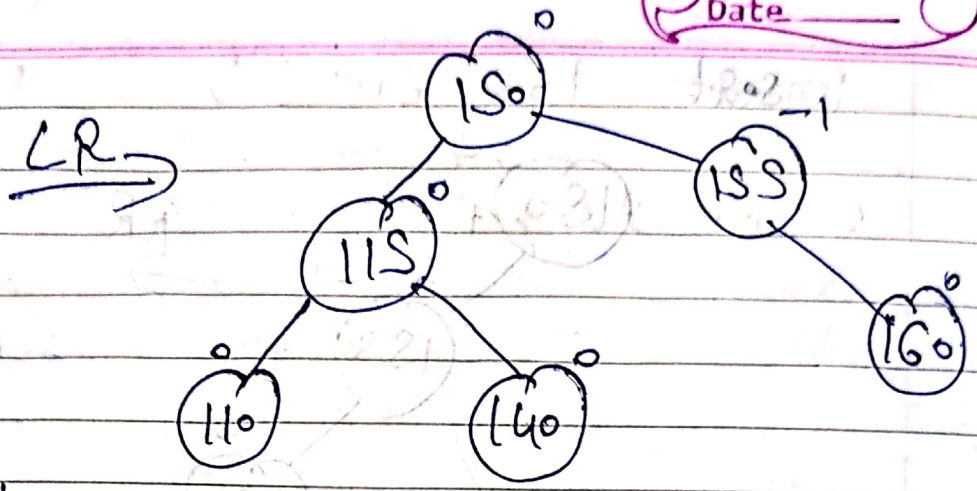


insert 110

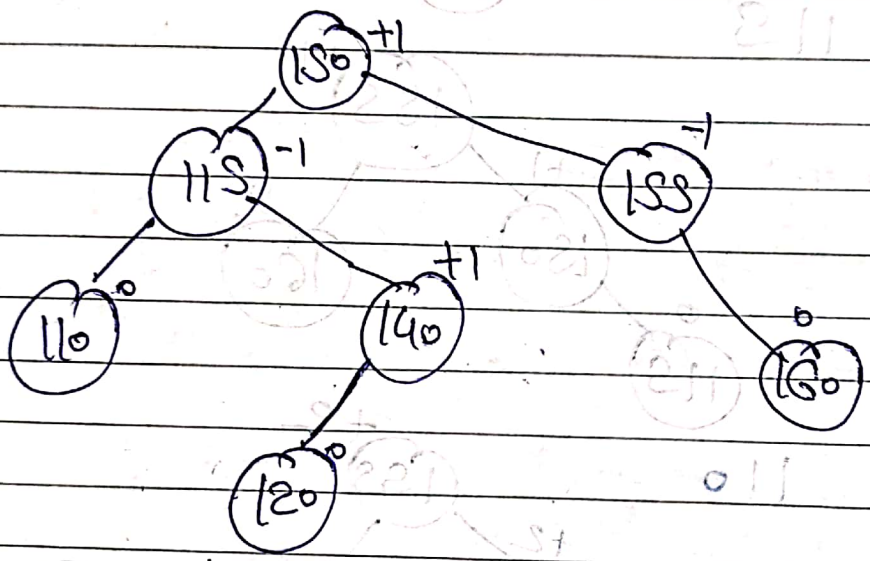


insert 140

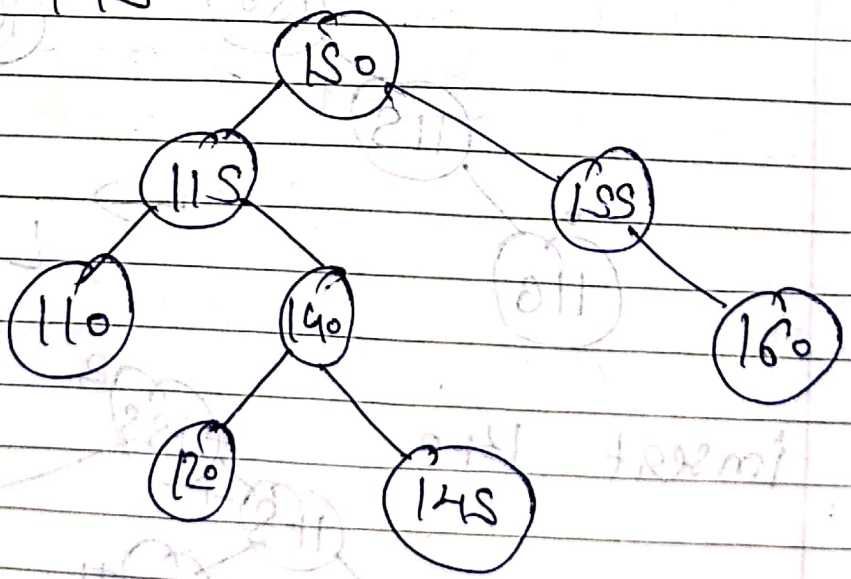


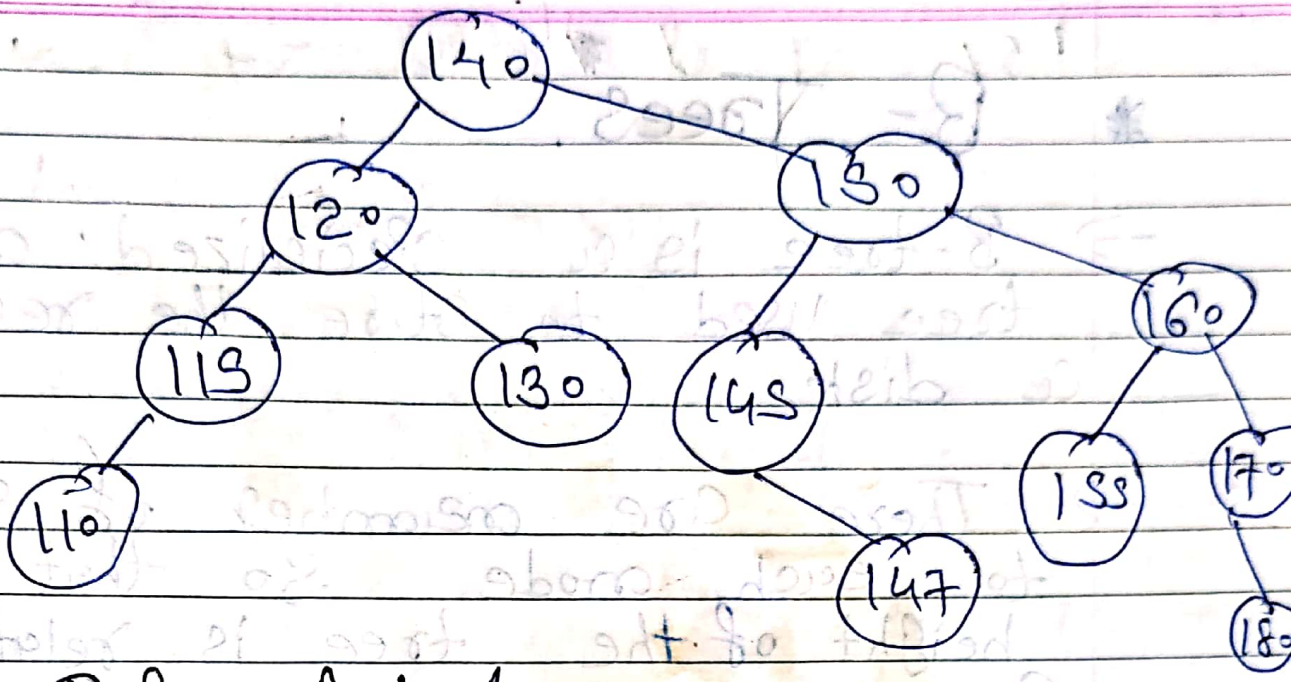


insert 120



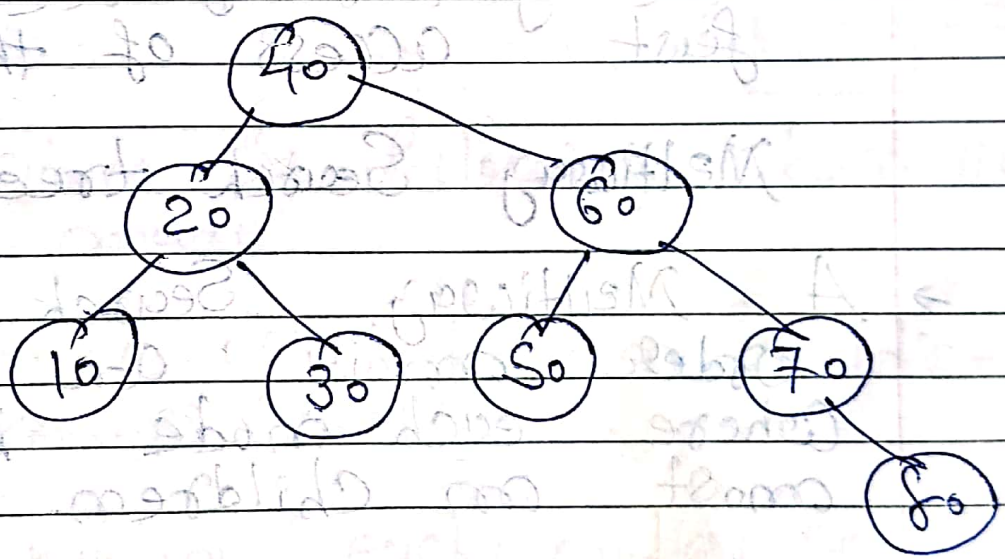
Insert 145





Ex1' Define AVL tree.

10, 20, 30, 40, 50, 60, 70, 80



* B- Trees.

→ B-tree is a specialized multiway tree used to store the record in a disk.

→ There are number of subtree to each node. So that the height of the tree is relatively small.

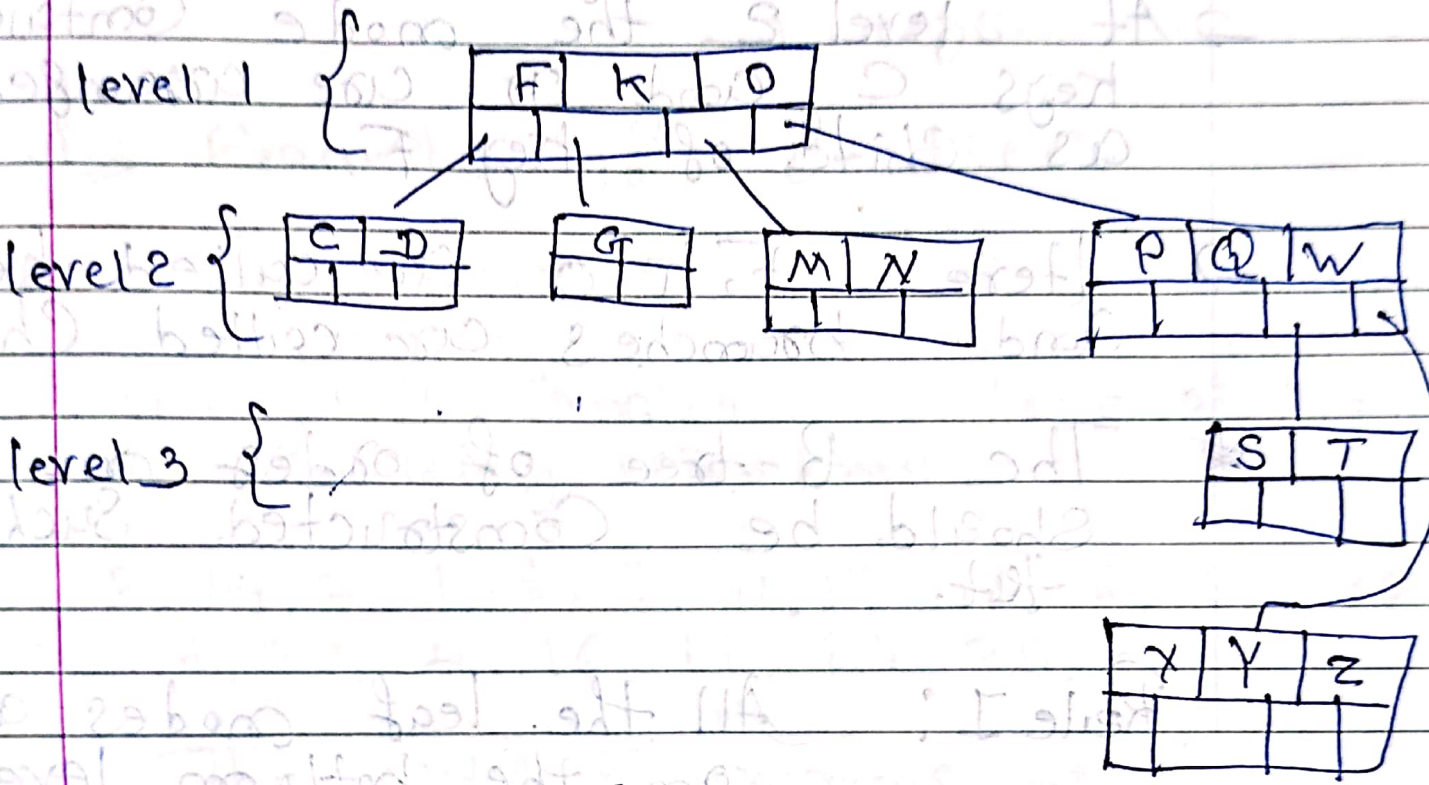
→ The goal of B-tree is to get fast access of the data.

Multiway Search tree:-

→ A Multiway Search tree of order m is an ordered tree where each node has at the most m children.

→ if m number of children in a node then (m-1) is the number of keys in the node.

Ex:- Following is a tree of order 4.



From above fig. following observation Can be made.

(1) The node which has m children Process (m-1) keys.

(2) The key in each node are in ascending order.

(3) For every node Node. Child[0] has only keys which are less than Node. key[0], Child[i] has only keys which are greater than Node. key[0].

→ The node at level 1 has E, K and O as keys.

→ At level 2 the node containing keys C and D are arranged as child of key F.

→ Here F, K, O are called keys and branches are called Childre

* The B-tree of order m should be constructed such that:

Rule 1: All the leaf nodes are on the bottom level.

Rule 2: The root node should have atleast two children.

Rule 3: All the internal node except root node have atleast ceil (m/2) nonempty children. The ceil is a fun such that ceil (3.4) = 4,

ceil (9.3) = 10

ceil (2.93) = 3

ceil (7) = 7

Rule 4: Each leaf node must Contain atleast ceil (m/2) -1 keys.

Ex: Example of B-tree

* Insertion

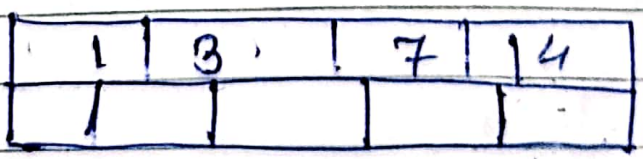
We will Construct B-tree of order 5 following numbers.

3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12 20, 26, 4, 16, 18, 24, 25, 19

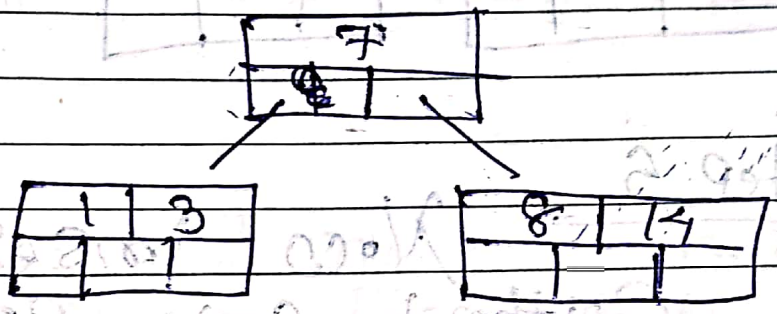
- The order 5 means at the most 4 keys are allowed.

- The internal node should have atleast 3 noneempty children and each leaf node must Contain atleast 2 keys.

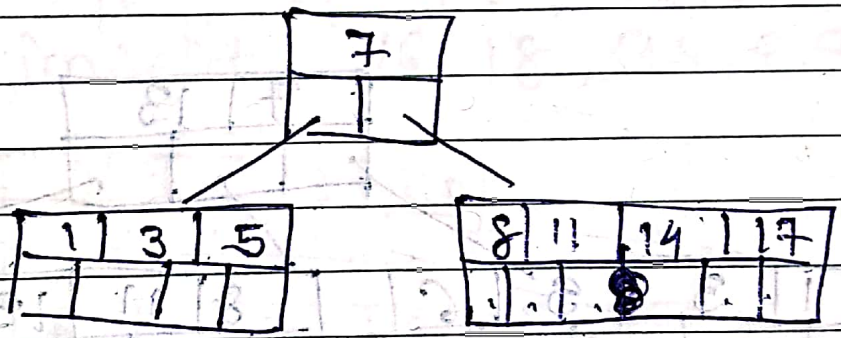
Step 1 Insert 3, 14, 7, 1 as follow



Step 2 — if we insert 8 then we need to split the node 1, 3, 7, 8, 14 at medium.

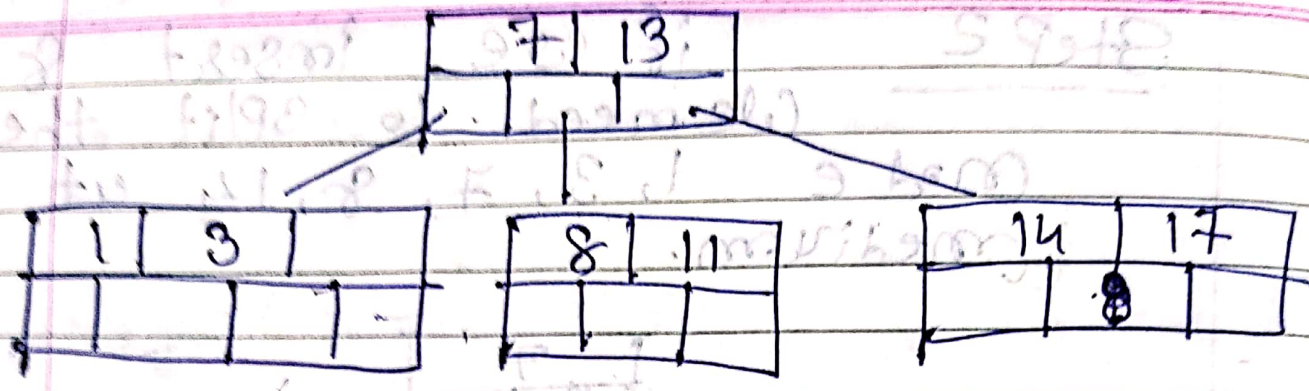


Step 3

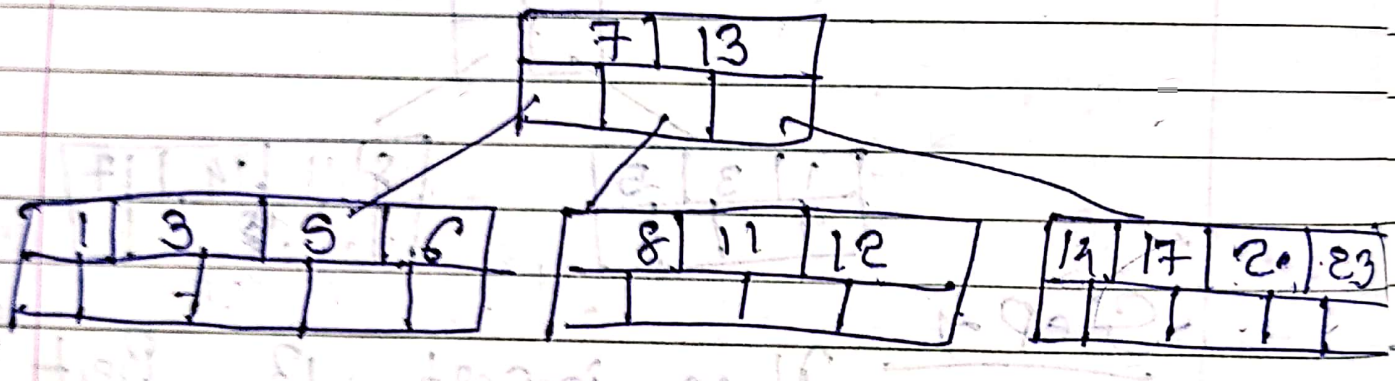


Step 4 — Now insert 13. But if we insert 13 then the leaf node will have 5 keys which is not allowed.

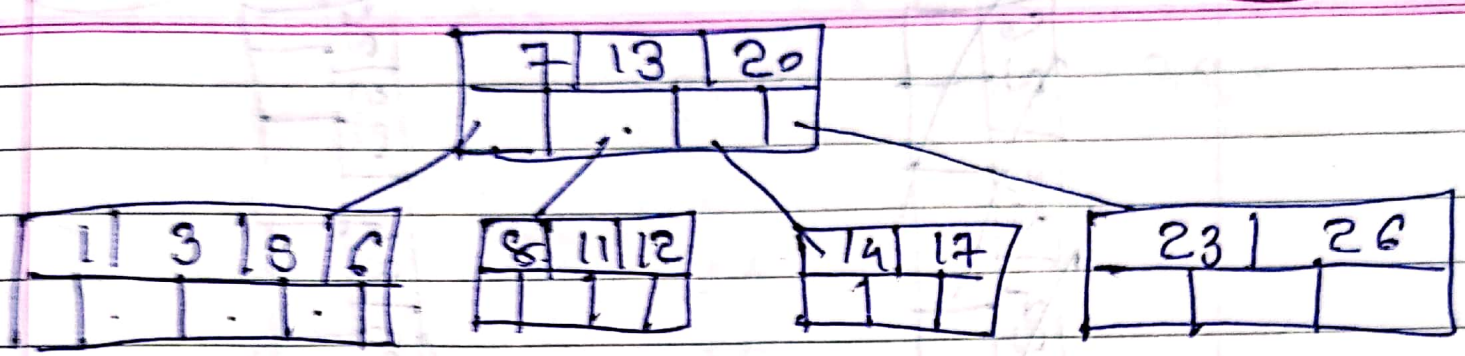
8, 11, (13), 14, 17 is split and medium node 13 is moved UP.



Step 5 :- Now insert 6, 23, 12, 20 without any split.

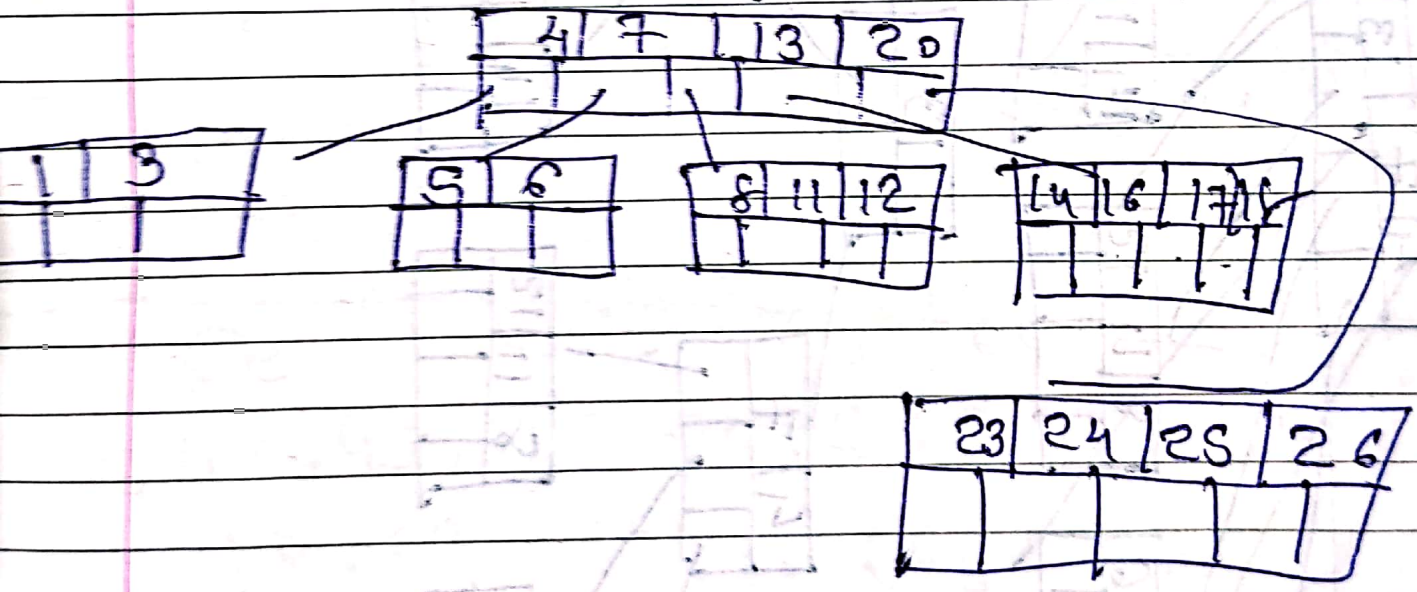


Step 6 :- The 26 is inserted to the rightmost leaf node. Hence 14, 17, 20, 23, 26 the node is split and 20 will moved up



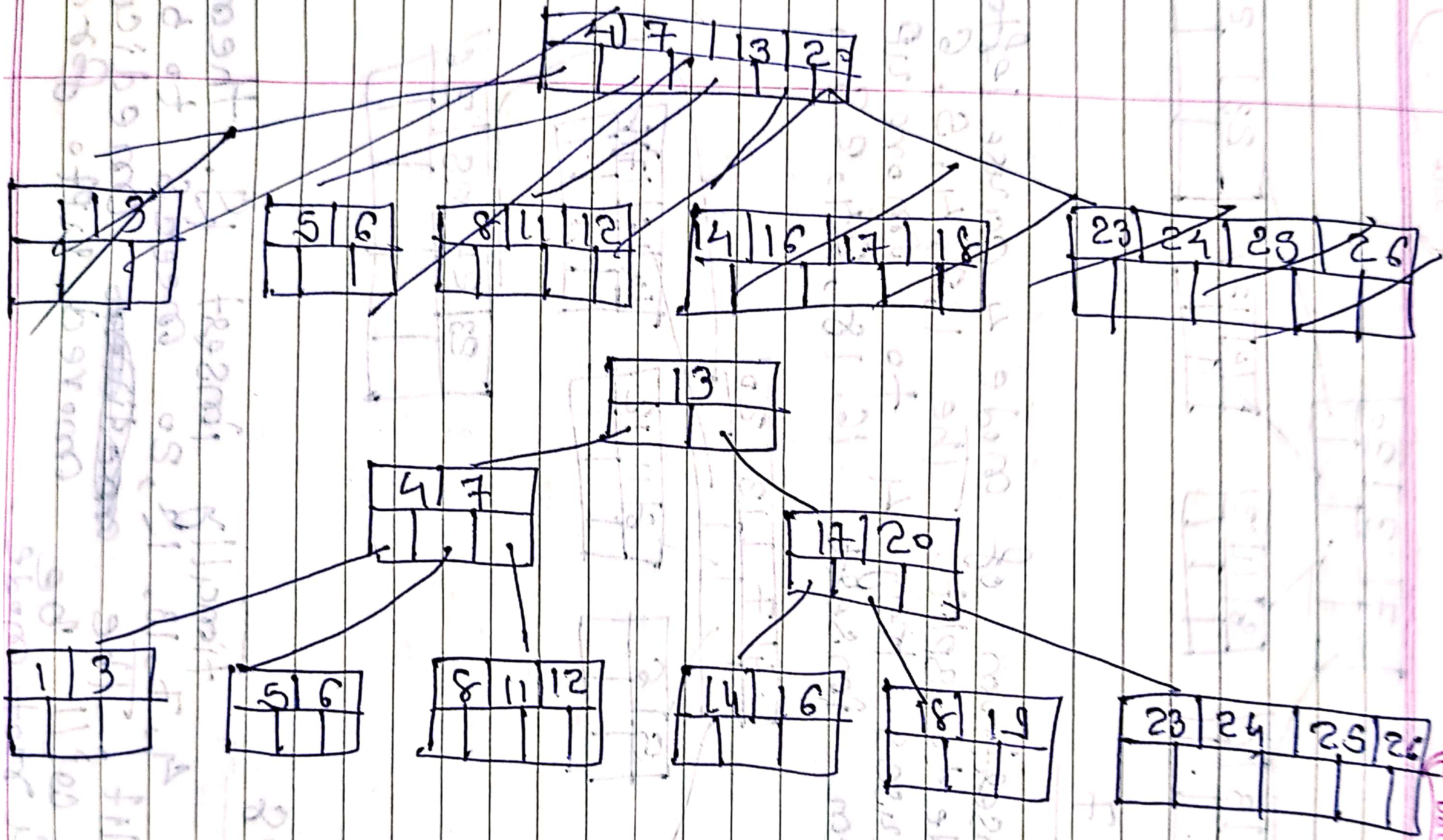
Step 7

Insertion of node 4 cause leftmost node to split. The 1, 3, 4, 5, 6 causes key 4 to move up. Then insert 16, 18, 24, 25.



Step 8

Finally insert 19. Then 4, 7, 13, 18, 20 needs to be split. The ~~median~~ median 13 will be moved upto form a root node.



Page
Date