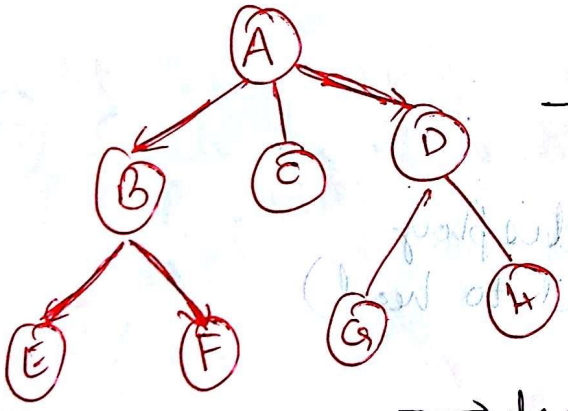
TREES

Defn:
A tree is a set of finite set of one or more nodes that shows parent-child relation such that:

- There is a spl node called Root node.
- The remaining nodes are partitioned into disjoint subsets $T_1, T_2, T_3 \cdots T_n$, $n \geq 0$

where $T_1, T_2 \cdots T_n$ = all children nodes of root node & are themselves trees called subtrees



→ Tree has 8 nodes:
A, B, C, D, E, F, G, H.

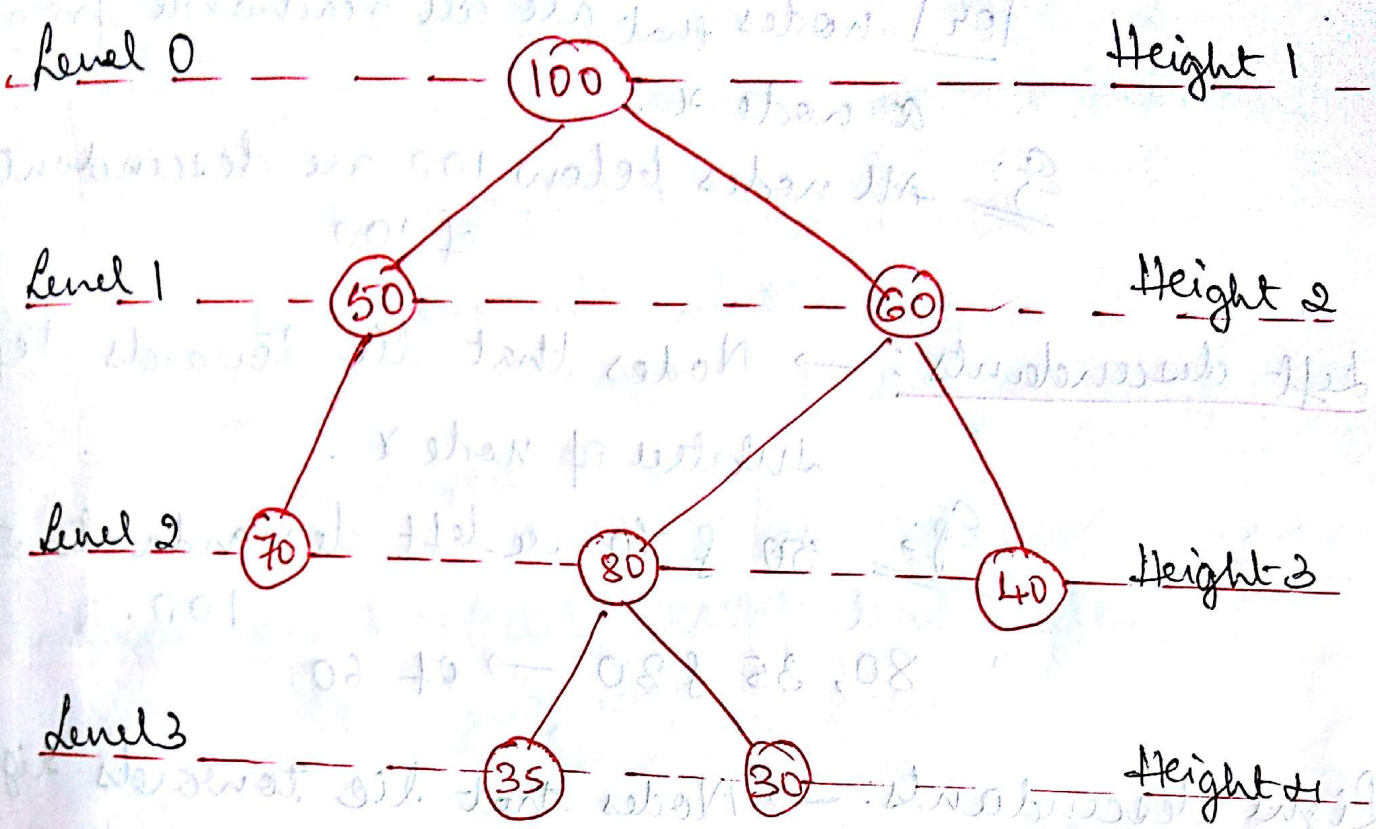
→ Node A = root

→ draw tree ∵ root @ top

→ Nodes B, C & D are children of root & hence are 3 subtrees identified by B, C & D

→ Node A → Parent of B, C & D.
Node D → Parent of G & H
Node B → Parent of E & F

Tree terminologies:



No. of levels = 4

Max. height / depth = 4

Root node: → First node → written @ top.
→ Does not have a parent.

Child node: → Node got from the parent node.
→ Parent can have 0 / more child nodes.

Siblings: → 2 / more nodes → in same parent
Eg: 50 & 60 → same parent 100.

Ancestors: → Nodes obtained in the path from the specified node x while moving upwards towards the root node.
Eg: 100 → ancestor of 50 & 60,

Descendants: → Nodes in the path below the parent
OR nodes that are all reachable from a node x.

Eg: All nodes below 100 are descendants of 100.

Left descendants: → Nodes that lie towards left subtree of node x.

Eg: 50 & 70 are left descendants of 100.
80, 35 & 30 → of 60.

Right descendants: → Nodes that lie towards right subtree of node x.

Eg: 60, 80, 40, 35 & 30 of 100
30 of 80.

Left subtree: → All nodes that are left descendants of node x form left subtree.

Eg: Left subtree of 100 = 50, 70.

Right subtree: → All nodes that are right descendants of a node x form right subtree

Eg: Right subtree of 100 = 60, 80, 40, 35, 30

Parent: → A node having left subtree OR right subtree OR both

Eg: Parent for 50 & 60 = 100

Degree: → No. of subtrees of a node.

Eg: 100 has 2 subtrees, ∴ degree = 2.
50 has 1 " , ∴ " = 1.
70 has NO " , ∴ " = 0.

Leaf: → Node in a tree, with degree = 0
OR Node ∈ empty left & right child.
→ Also called Terminal node.

Internal nodes: → Nodes except leaf nodes.

Eg: 100, 50, 60 & 80

External nodes: → NULL link of any node in a tree is an external node.

Eg: 2 link of 50
2 link & 2 link of 70, 35, 30, 40.

Level: → Distance of a node from the root.
→ Distance from root to itself = 0
∴ Level = 0
→ Distance from 50 to 100 = 1
∴ Level = 1

Height (Depth): → Max. level of any ~~level~~ leaf in the tree.

Eg: Ht = 4

Representation of Trees

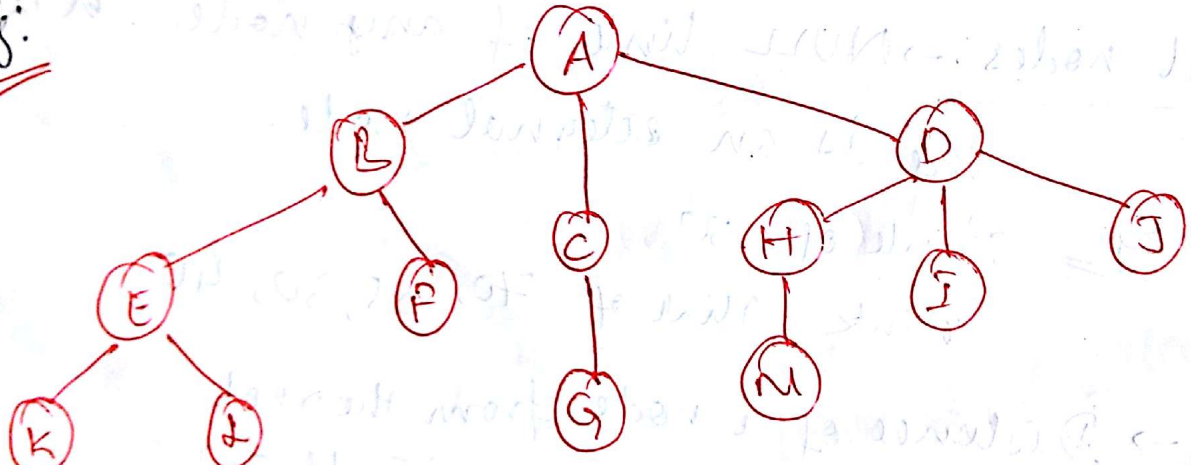
- List representation
- Left child – right sibling representation
- Binary tree representation (Degree → 2 representation)

I List representation:

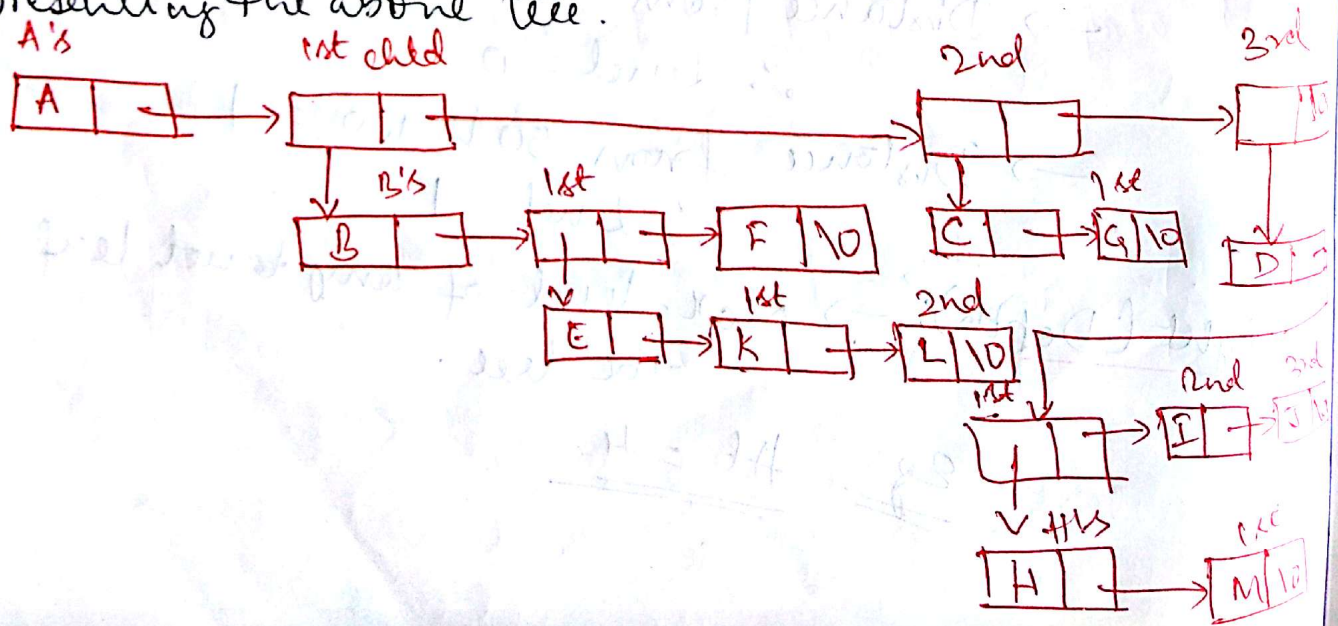
Steps:

1. Root node comes first
2. Immediately followed by a list of subtrees of that node
3. Recursively repeated for each subtree.

Eg:



Representing the above tree:



NOTE:

① Since there are 3 children for node A, there are 3 nodes to the right of A in the list

② A's 1st child → B, 2nd → C, 3rd → D

③ 2 children for node B, 2 nodes to the right of B

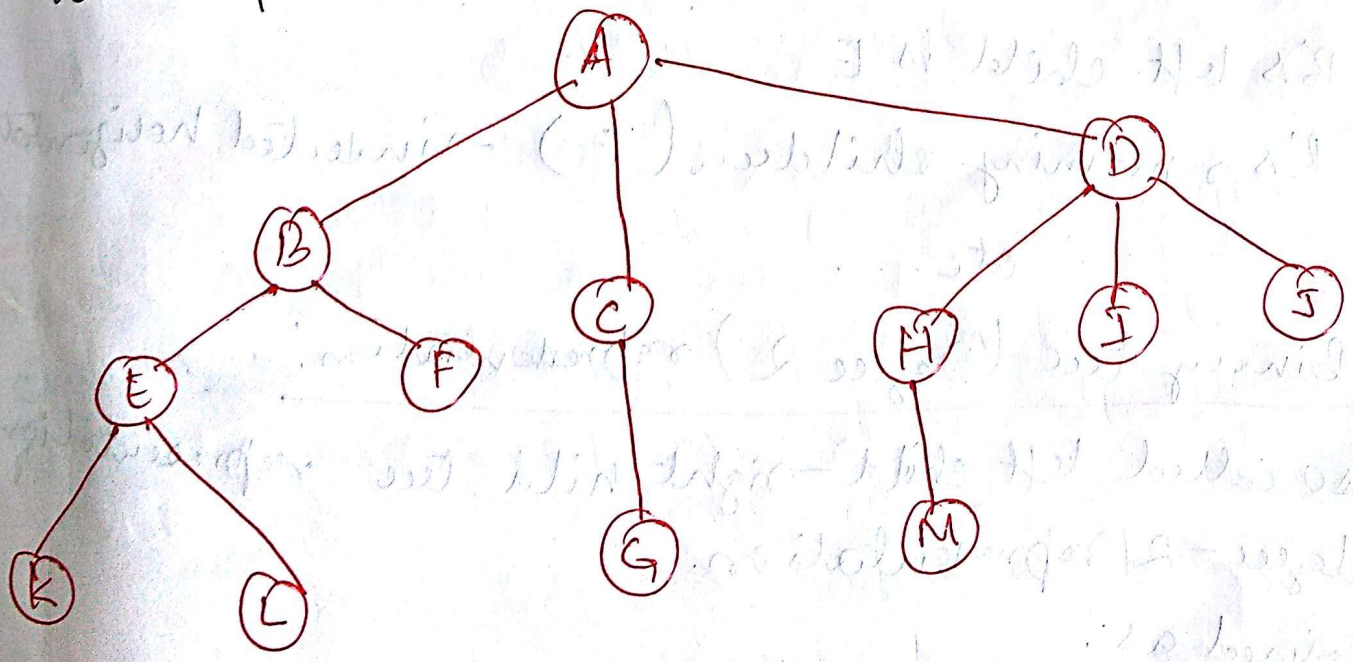
etc. . . .

II Left-child Right-sibling representation:

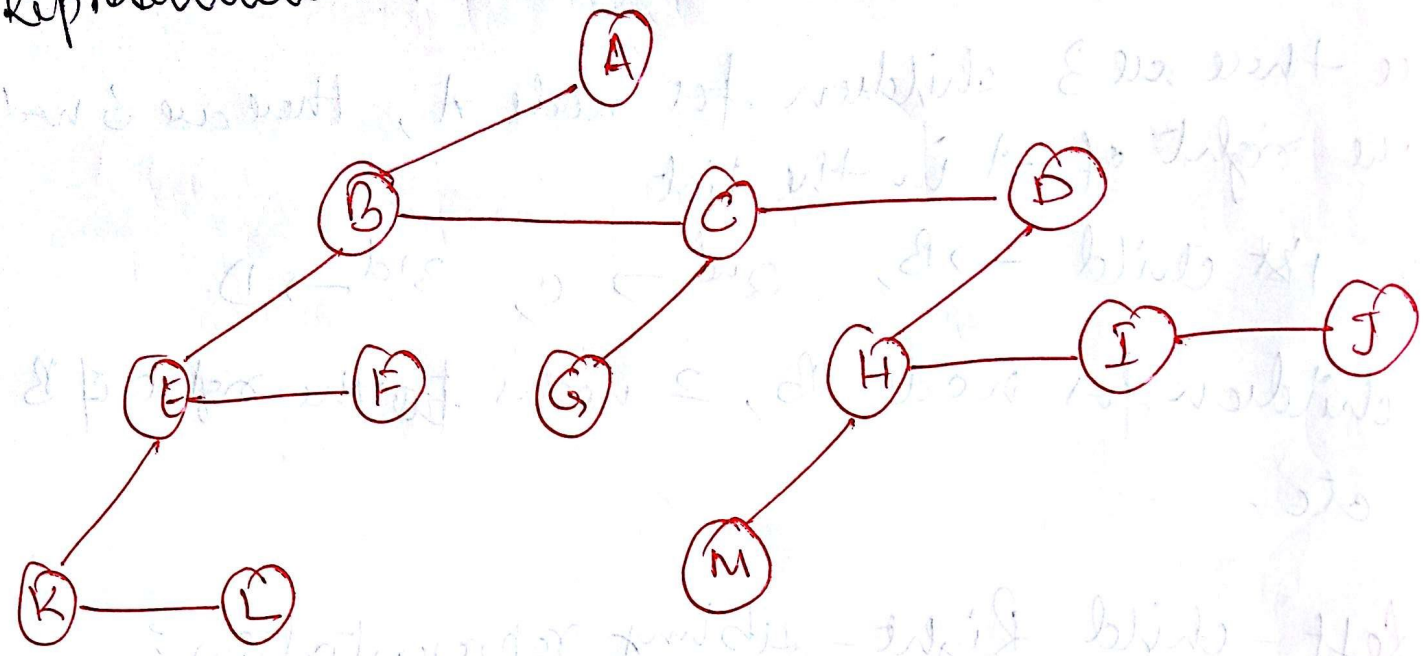
Can be obtained as:

① Left pointer of a node in the tree will be the left child in this representation.

② Remaining children of a node are inserted horizontally to the left child in the representation.



Representation:



NOTE:

(1) A's left child is B, ∴ same in representation

(2) A's remaining children (C & D) are inserted horizontally to B

(3) B's left child is E

B's remaining children (F) → inserted horizontally

etc...

III Binary tree (Degree 2) representation:

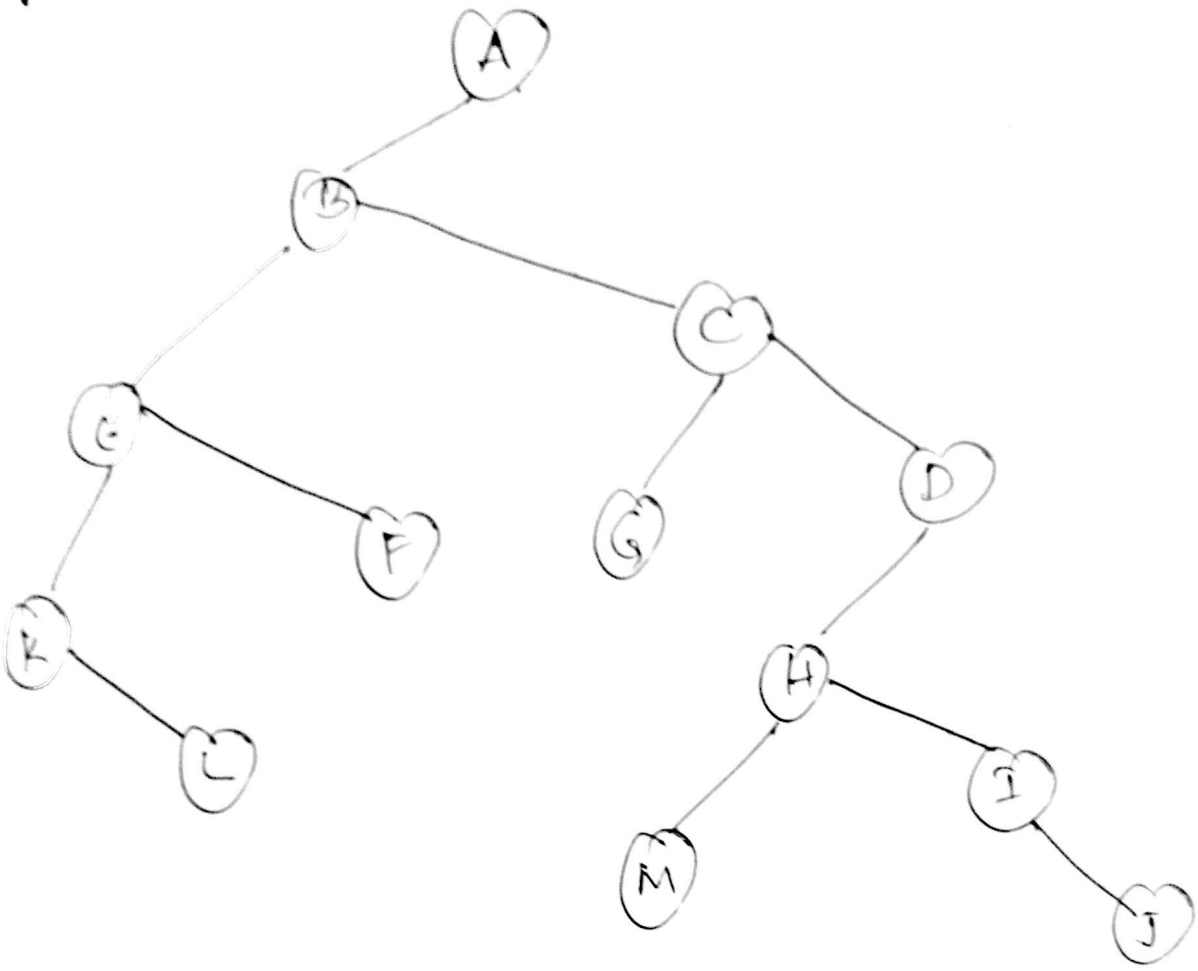
Also called left child–right child tree representation of degree-2 representation.

Obtained as:

(1) Obtain left child–right sibling representation

(2) Rotate horizontal lines 45° clockwise by 45 degrees.

Representation:

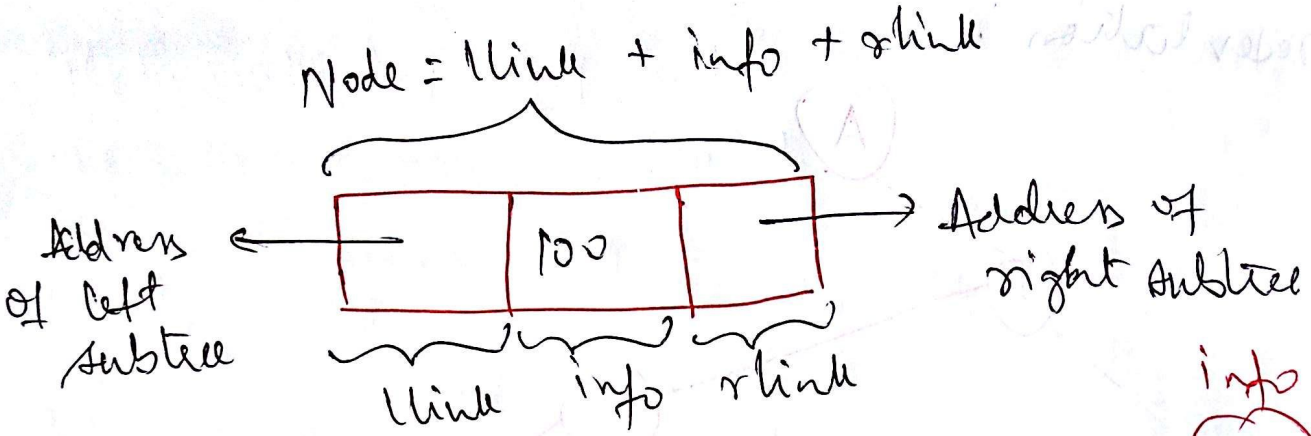


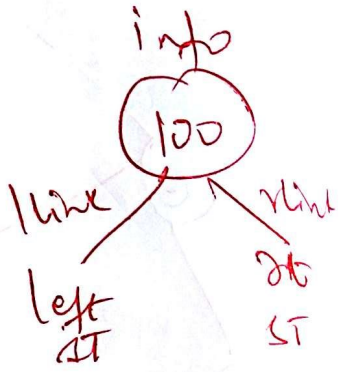
Binary trees:

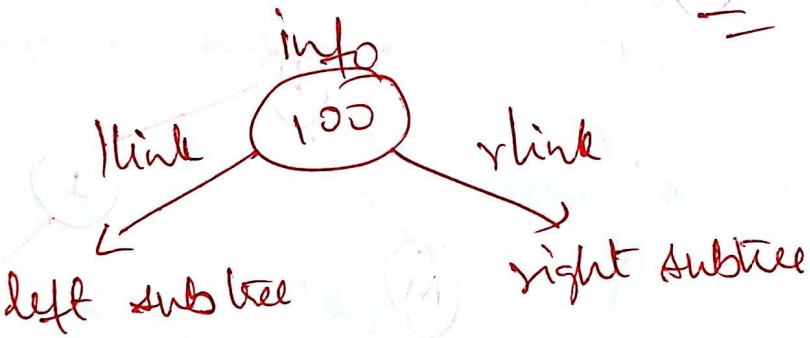
Defn: A binary tree is a tree which has finite set of nodes that is either empty or consist of a root & 2 subtrees called left subtree & right subtree.

It can be partitioned into 3 subgroups:

1. Root: If tree is not empty, the first node is called root node.
2. Left subtree: tree which is connected to the left of the root. It comes towards left
3. Right subtree: tree which is connected to the right of the root. Comes towards right



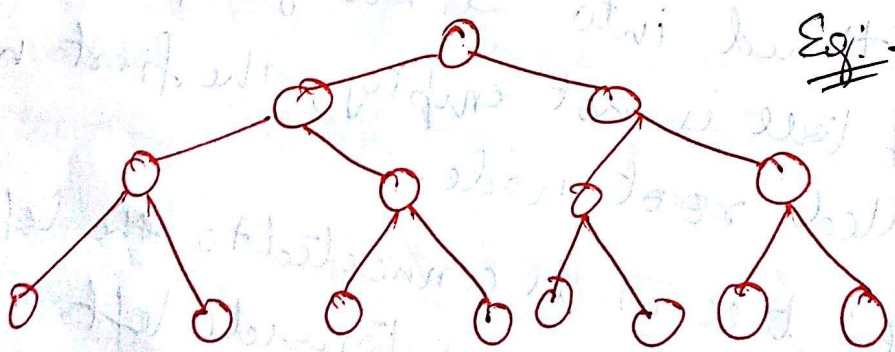




Properties of Binary trees:

I a) Max. no of nodes on level i of a binary tree $= 2^i$ for $i \geq 0$

b) Max. no of nodes in a binary tree of depth $k = 2^k - 1$.



Eg: A complete binary tree

Proof :

No of nodes @ level 0 = $2^0 = 1$

@ level 1 = 2 = 2^1

@ level 2 = 4 = 2^2

@ level 3 = 8 = 2^3

@ level i = 2^i

Total no of nodes in a full binary tree of level i

$= 2^0 + 2^1 + 2^2 \cdots + 2^i$

This is a Geometric progression, sum

$$S = \frac{a(r^n - 1)}{(r-1)}$$

where $a = 1$, $n = i+1$ & $r = 2$.

Total no of nodes $n_t = a(r^n - 1) / (r-1)$

$= 1(2^{i+1} - 1) / 2 - 1)$

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

∴ total no of nodes =

$$\boxed{n_t = 2^{i+1} - 1}$$

Substituting $i = 3$, $n_t = 15$

Depth of a tree k = max level + 1

$= i + 1$

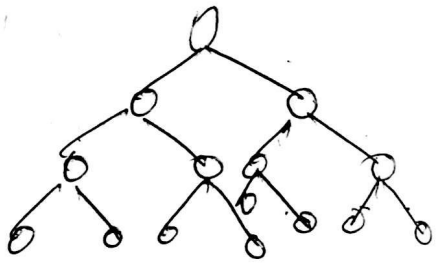
∴ $\boxed{n_t = 2^k - 1}$

II) The number of leaf nodes is equal to the no of nodes of degree 2 plus 1

Consider a binary tree of degree-2 nodes:

∴ each node has max 2 children

A node cannot have more than 2 children.



Let no of nodes of degree-0 = n_0

" " " 1 = n_1

" " " 2 = n_2

Let total no of nodes in the tree = n

$= \boxed{n_0 + n_1 + n_2}$ ——— (1)

From tree, total no of nodes =

total no of branches + 1

∴ $\boxed{n = B + 1}$ ——— (2)

If node has degree = 1,

no of branches = 1.

∴ for n_1 no of nodes of degree 1,

no of branches = $n_1 \times 1$

$= \boxed{n_1}$ ——— (3)

∴ for node of degree 2,

no of branches = 2

So, for n_2 no. of nodes of degree 2,

no. of branches = $\boxed{2n_2}$ ⟶ (4)

Adding (3) & (4),

total no. of branches $\boxed{B = n_1 + 2n_2}$ ⟶ (5)

Substituting (5) in (2)

$$\boxed{n = n_1 + 2n_2 + 1} \longrightarrow (6)$$

Relation between total no. of nodes of degree-2 & total no. of leaf nodes.

Subtract (6) from (1)

$$n = n_0 + n_1 + n_2$$
$$- \quad n = n_1 + 2n_2 + 1$$
$$0 = n_0 - n_2 - 1$$

$$n = n_0 + n_1 + n_2$$
$$- \quad n = n_1 + 2n_2 + 1$$
$$\boxed{0 = n_0 - n_2 - 1}$$

$$\therefore \boxed{n_0 = n_2 + 1}$$

n_0 = total no. of nodes of degree 0 (leaf nodes)

n_2 = ——— " ——— 2 (binary tree)

Types of Binary trees :

- Strictly binary tree (Full BT)
- Skewed tree
- Complete binary tree
- An expression tree
- Binary Search tree (BST)

I Strictly (Full) Binary Tree:

Defn: A BT having 2^i nodes in any given level i is called a Strictly BT.

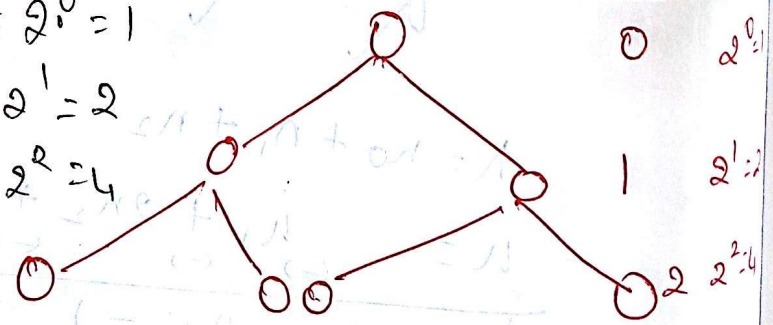
Here, every node, other than the leaf node has 2 children.

Also called full binary tree / proper binary tree.

No. of nodes @ level = 0 = $2^0 = 1$

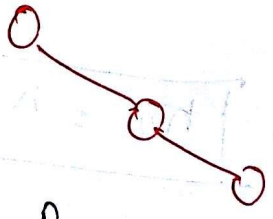
——, —— $1 = 2^1 = 2$

——, —— $2 = 2^2 = 4$

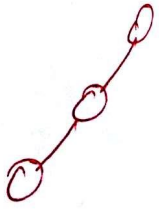


II Skewed tree:

Defn: A tree consisting of only left subtree or only right subtree.



Right skewed tree

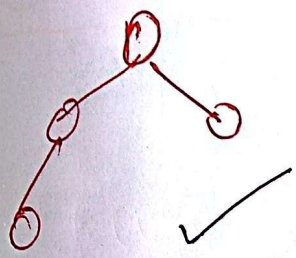


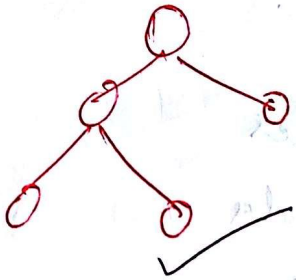
Left skewed tree

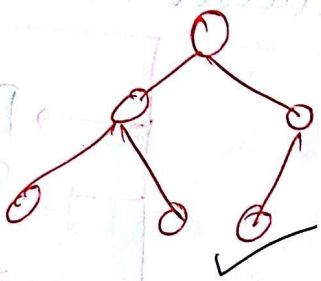
III. Complete binary tree:

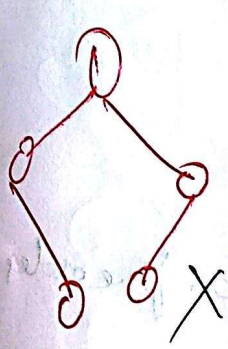
Def: A BT in which every level, except possibly the last level is completely filled.

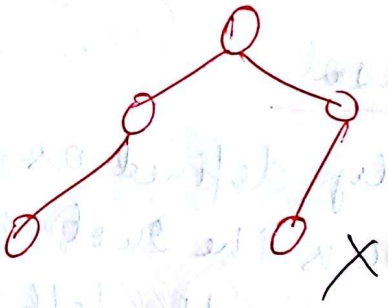
If nodes in the last level are not completely filled, then all the nodes in that level should be filled only from left to right.





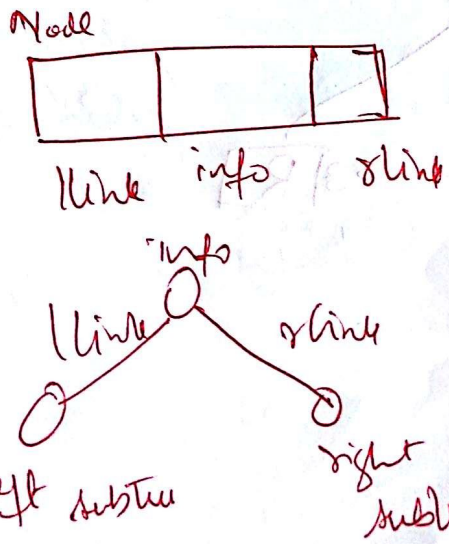






Binary tree representation

DLL



Binary tree traversal

Defn: Traversing is a method of visiting each node of a tree exactly once in a systematic order.

During traversal, we may print the info field of each node visited.

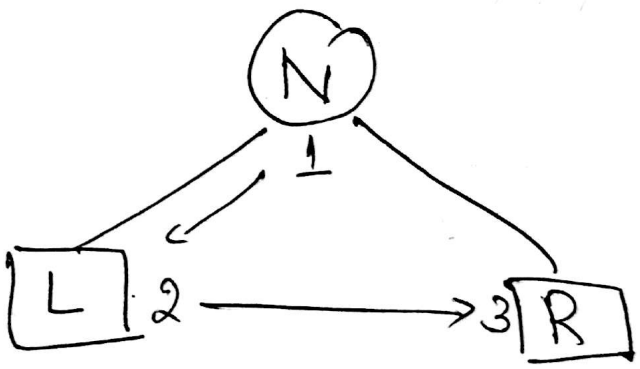
Traversal techniques:

- Preorder
- Post order
- Inorder

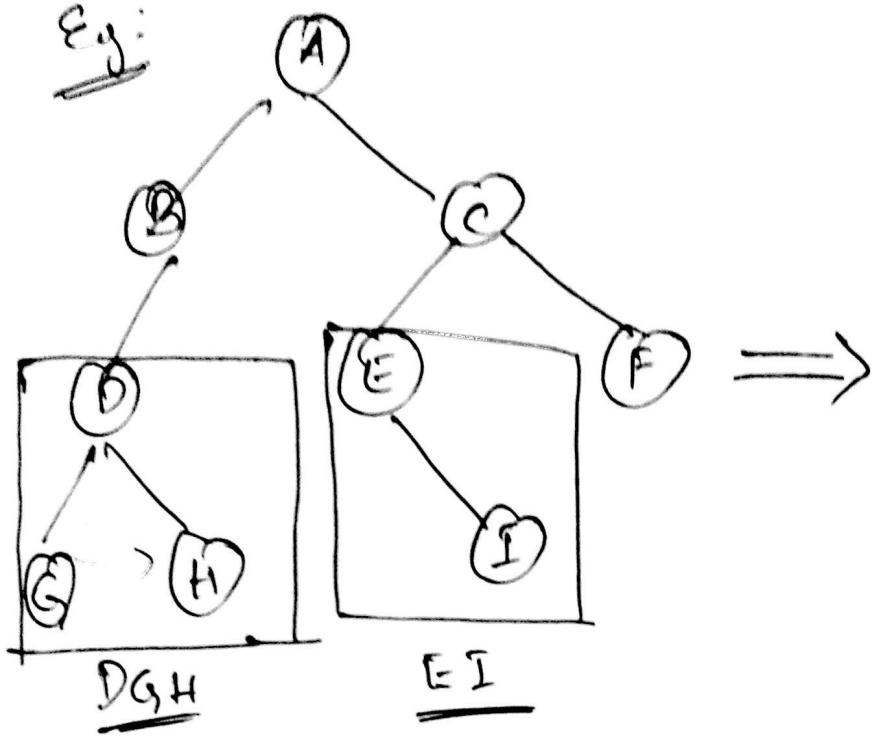
Preorder traversal:

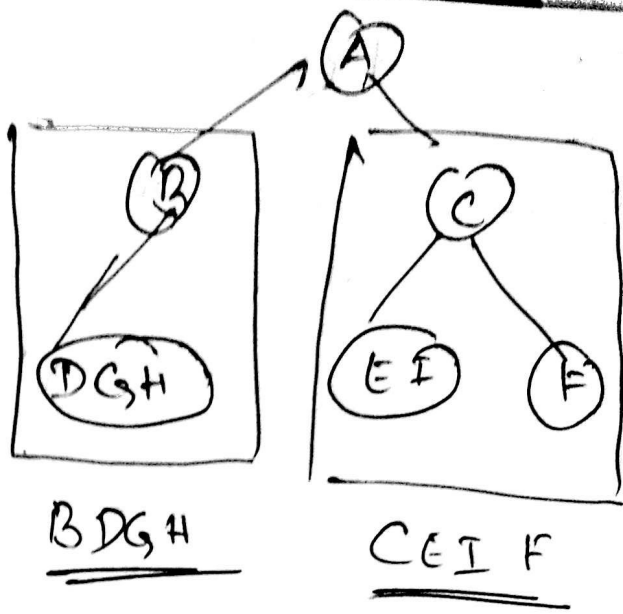
Defn Recursively defined as:

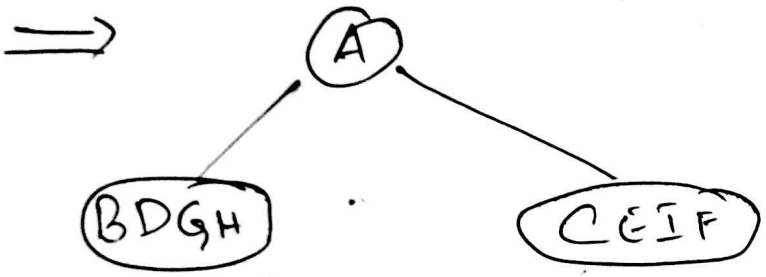
- process the root node [N]
- traverse the left subtree in preorder [L]
- —"— Rt —"— [R]



Eg:







⟹ A B D G H C E I F

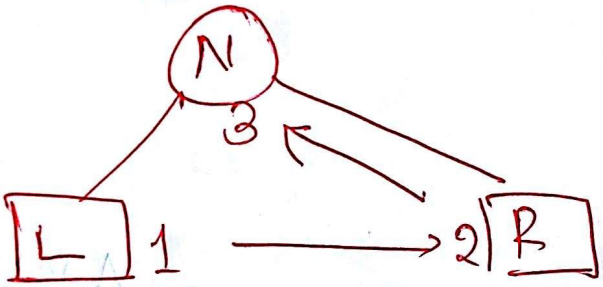
Fn:

```
void preorder (NODE root)
{
  if (root == NULL)
      return;
  printf ("%d", root -> info);   /* visit root */
  preorder (root -> llink);      /* visit left subtree in preorder */
  preorder (root -> rlink);      /* visit right subtree in preorder */
}
```

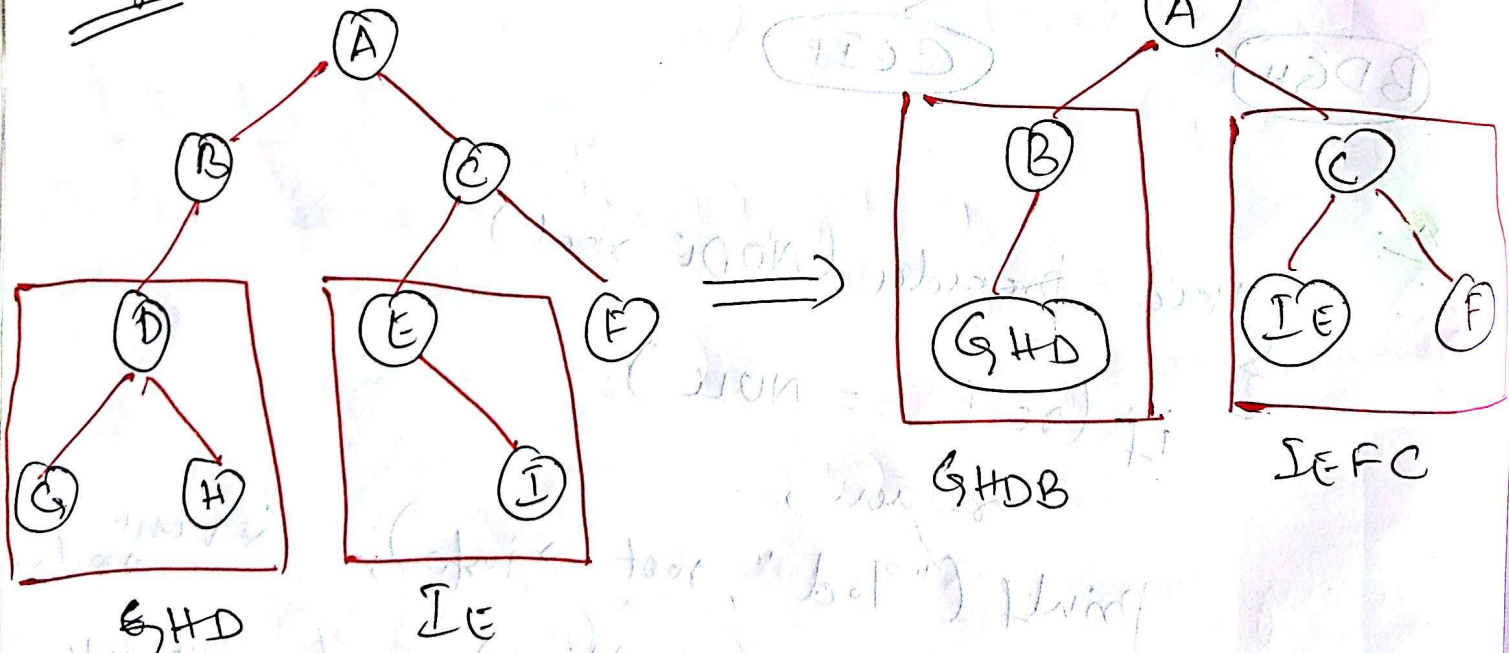
Post order traversal

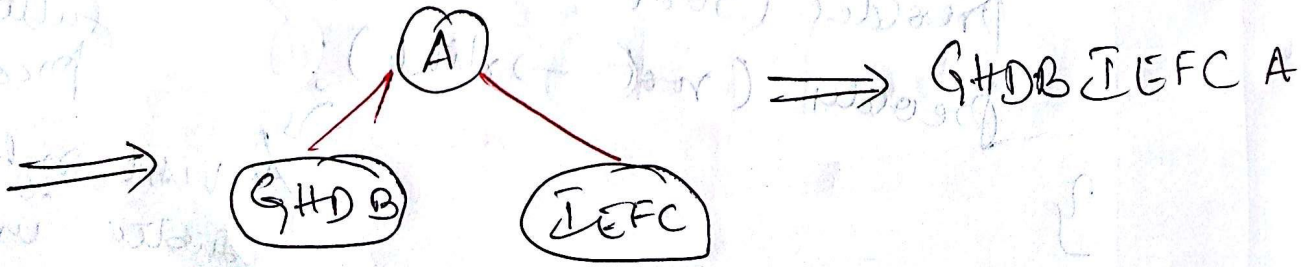
Defn: Recursively defined as:

- Traverse left subtree in post order [L]
- " right subtree " " [R]
- Process root node [N]



Eg:





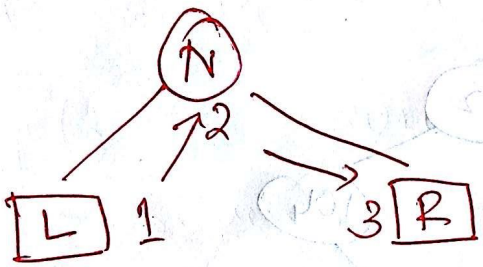
Fn:

```
void postorder(NODE root)
{ if (root == NULL)
      return;
  postorder(root->llink);      -> LST in postorder
  postorder(root->rlink);
  printf("%d", root->info);
}
```

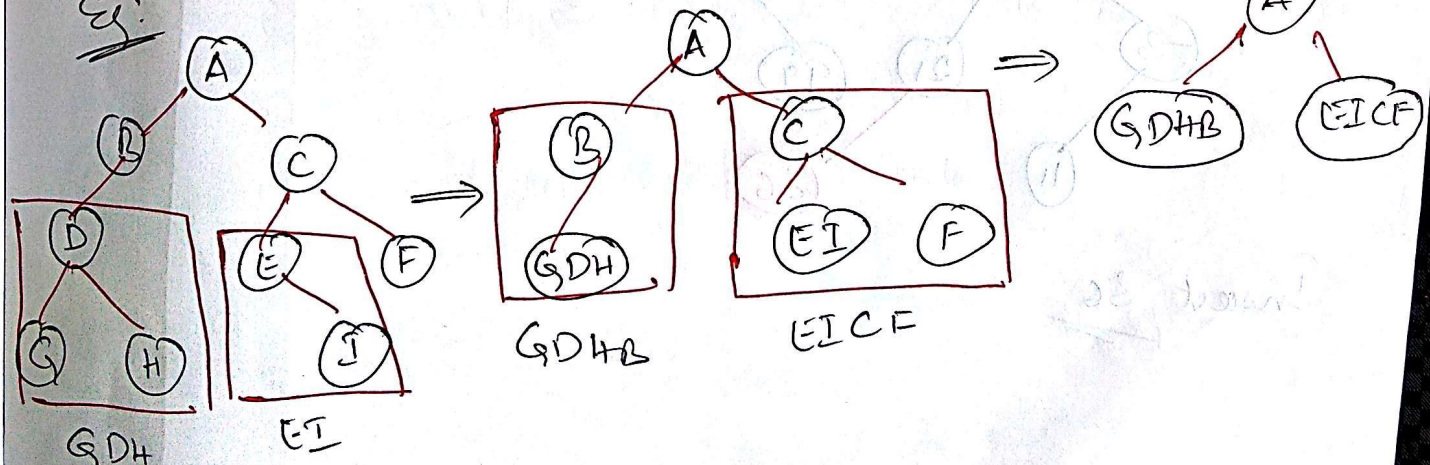
Inorder traversal:

Defn Recursively defined as:

- Traverse the left subtree in inorder [L]
- Process the node [N]
- Traverse the right subtree in inorder [R]



Eg:



⟹ GDHBA EICF

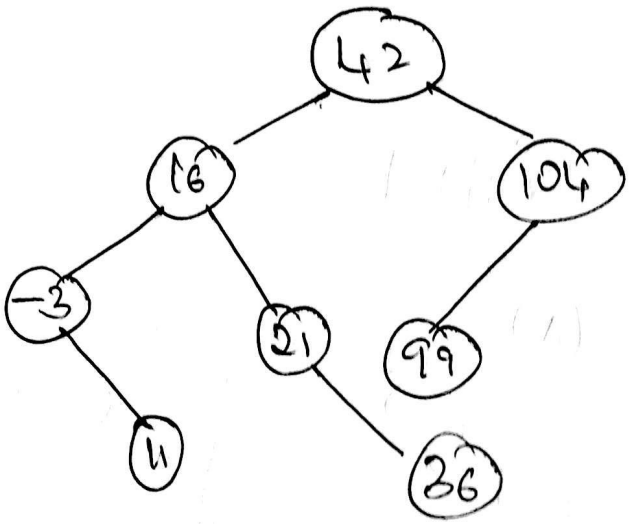
Fn.
```
void inorder(NODE root)
{ if (root == NULL)
        return;
  inorder(root -> llink);
  printf("%d", root -> info);
  inorder(root -> rlink);
}
```

Binary Search Tree :

A binary tree in which

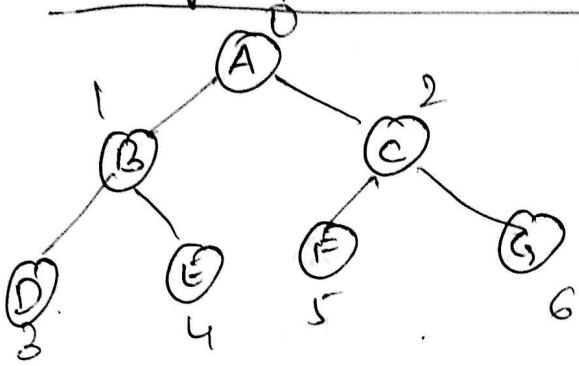
info (left subtree) ≤ info (root) ≤ info (right subtree)

42, 16, -3, 104, 99, 21, 11

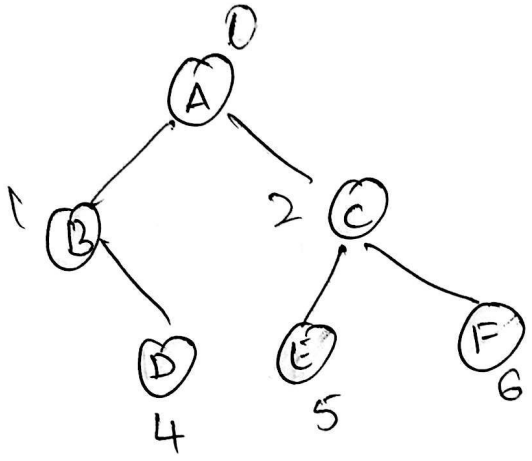


Insert 36

Array representation of trees :



0	1	2	3	4	5	6
A	B	C	D	E	F	G



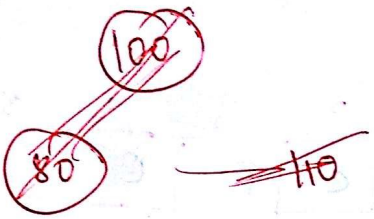
0	1	2	3	4	5	6
A	B	C		D	E	F

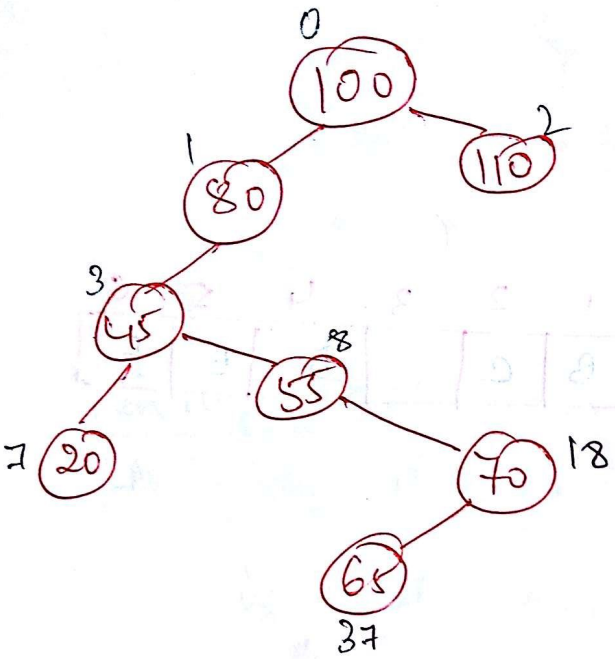
Notes:

1. Nodes are numbered sequentially from 0
2. Node ~~c~~ position = 0 → Root node
3. If index, i = 0 → position of the root node
4. If node is @ position = i,
   position of left child = 2i + 1
   —"— rt " = 2i + 2
5. If i = position of left child,
   i + 1 = —"— rt "
6. If i = position of rt. child,
   i − 1 = —"— left "
7. If node @ position i, parent @ position $\frac{(i-1)}{2}$
8. i = odd ⟹ left child
   else → right child.

Eg: Build BST & represent using arrays

100, 80, 45, 55, 110, 20, 70, 65



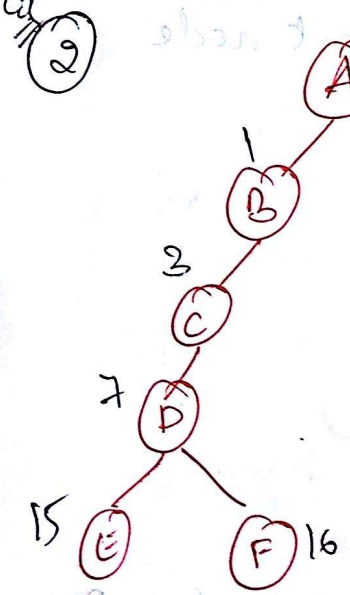


0 — 100
1 — 80
2 — 110
3 — 45
⋮
7 — 20
8 — 55
⋮
18 — 70
⋮
37 — 65

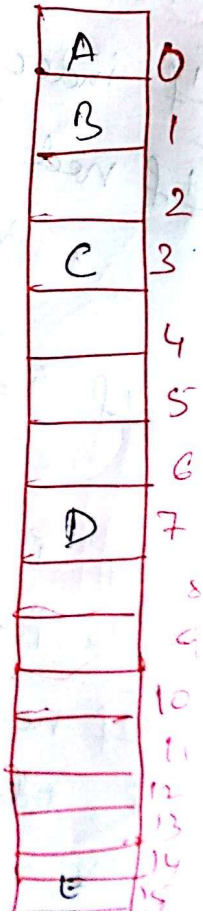
left = 2i + 1

right = 2i + 2

Eg: 2

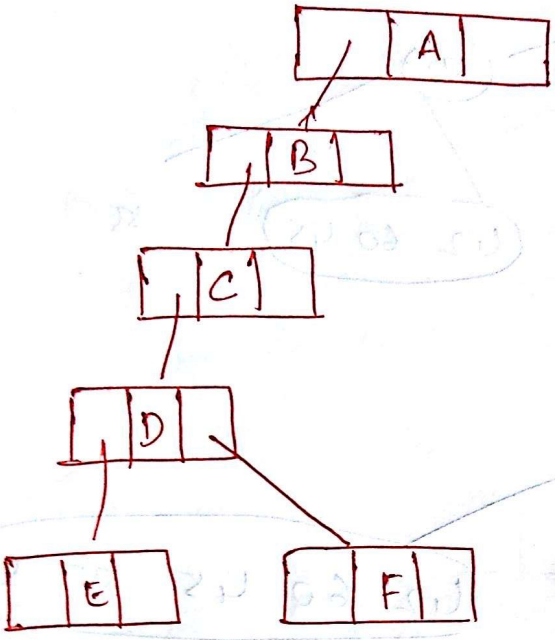


0 — A
1 — B
3 — C
7 — D
15 — E
16 — F

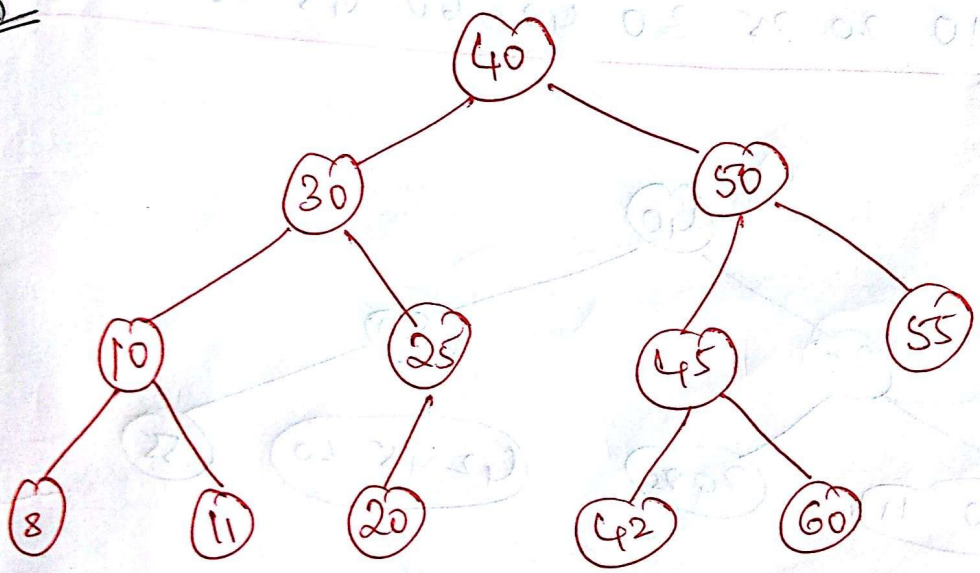


Pre

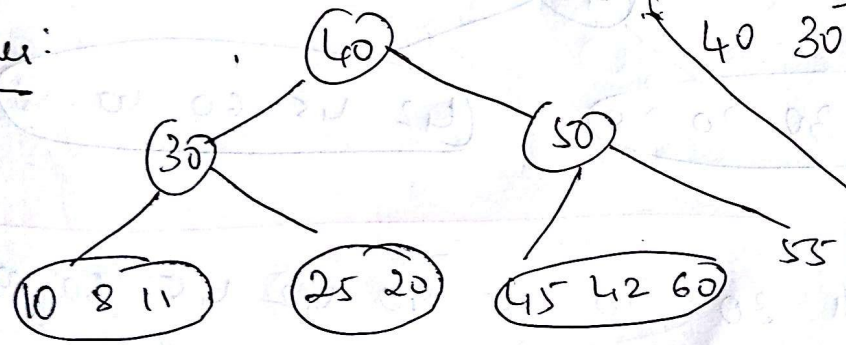
Linked list representation



Eg ②

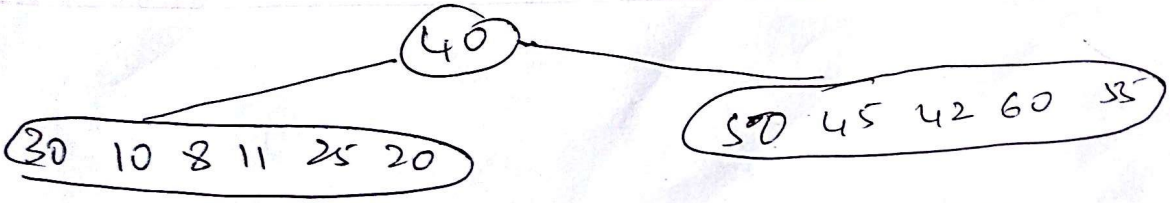


Preorder:

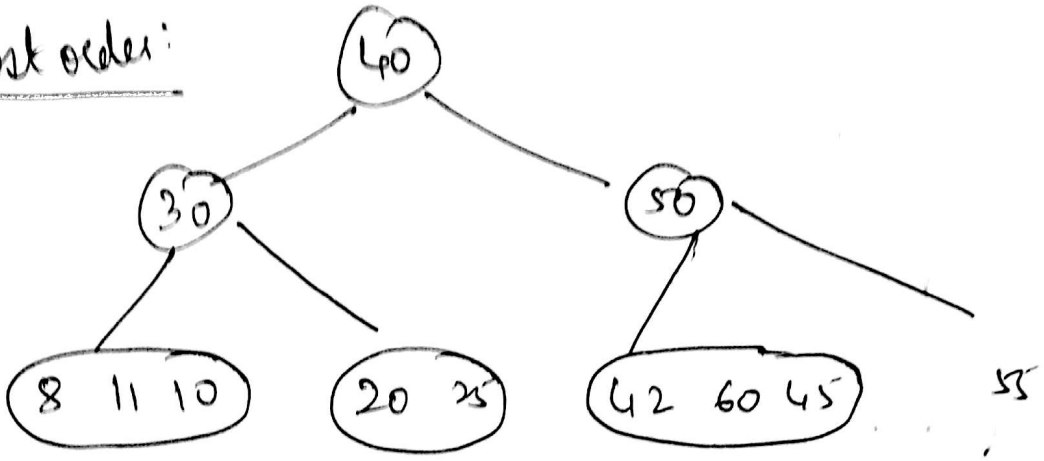


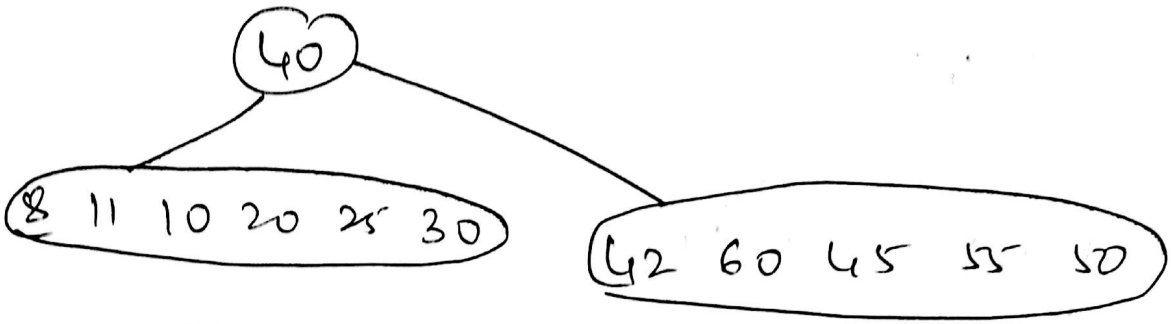
Ans

40 30 10 8 11
25 20 50 45
42 60 55



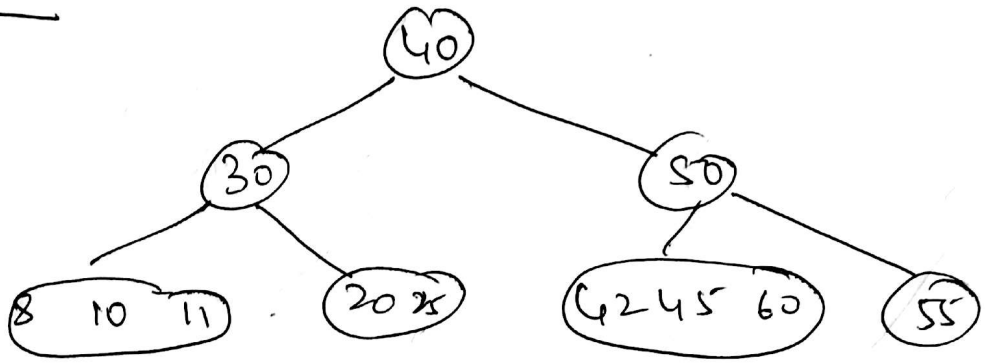
Post order:

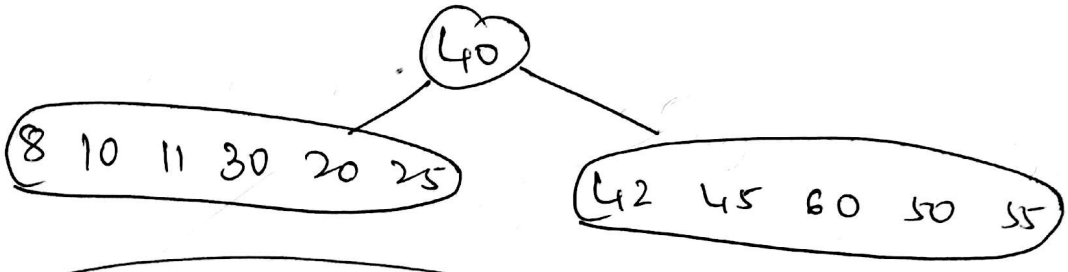




8 11 10 20 25 30 42 60 45 55 50 40

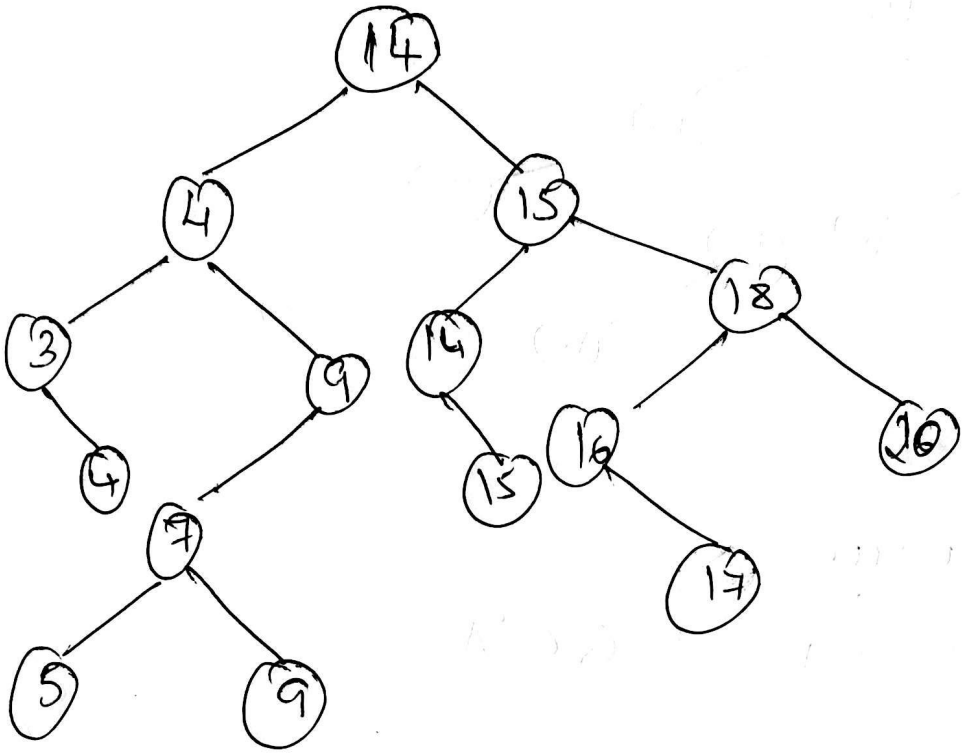
Inorder:



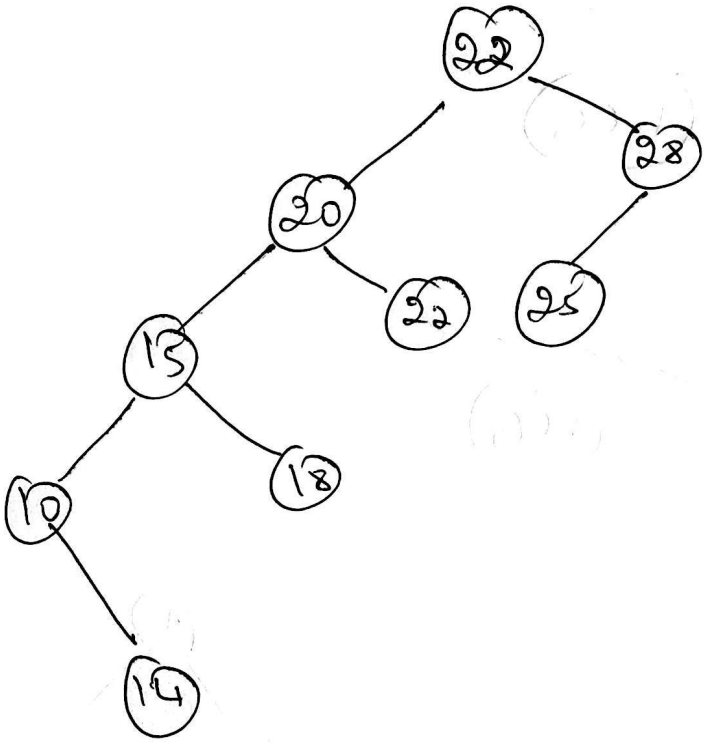


8 10 11 30 20 25 40 42 45 60 50 55

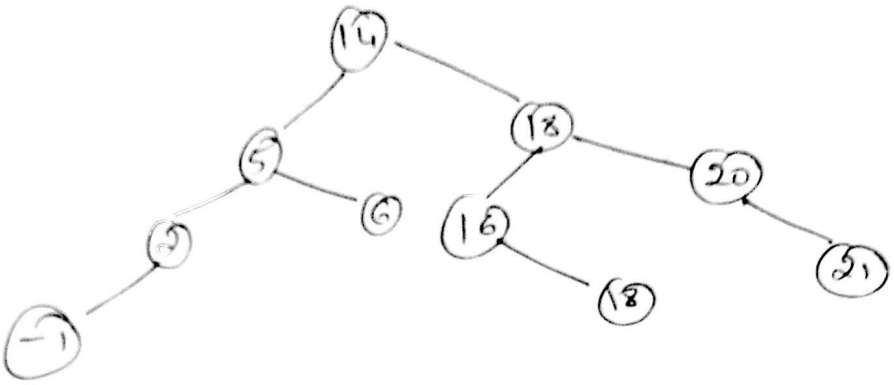
(5) BST 14, 15, 4, 9, 7, 18, 3, 5, 16, 4, 20, 17, 9, 14, 15



(6) BST → 22, 28, 20, 25, 22, 15, 18, 10, 14



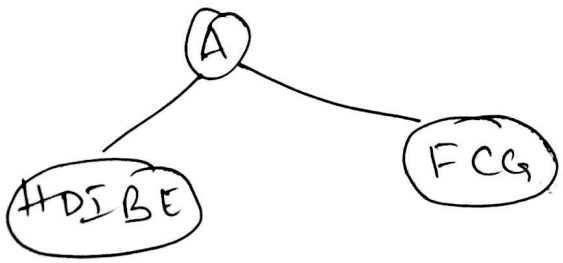
(7) BST: 14, 5, 6, 2, 18, 20, 16, 18, −1, 21

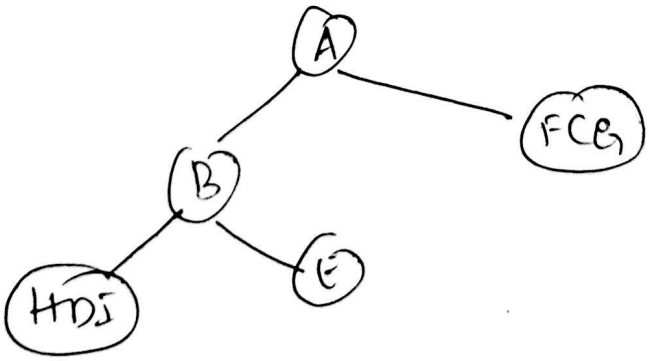


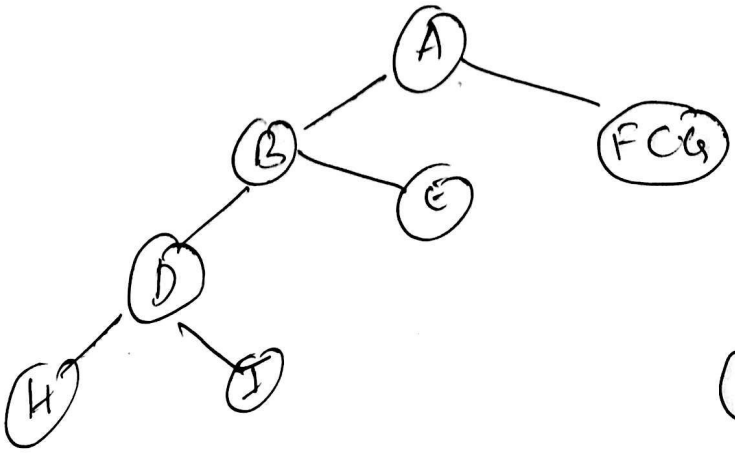
(8) Construct the tree:

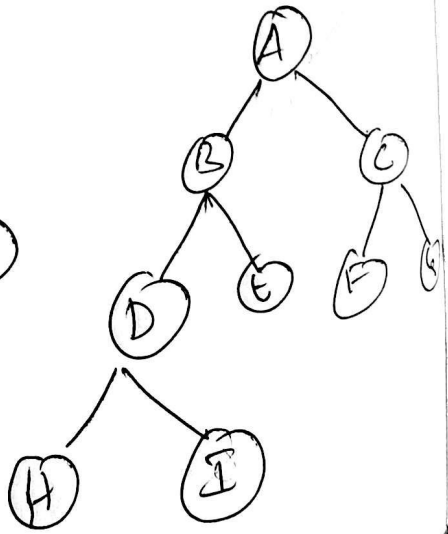
Postorder: H I D E B F G C [A]

Inorder: H D I B E [A] F C G



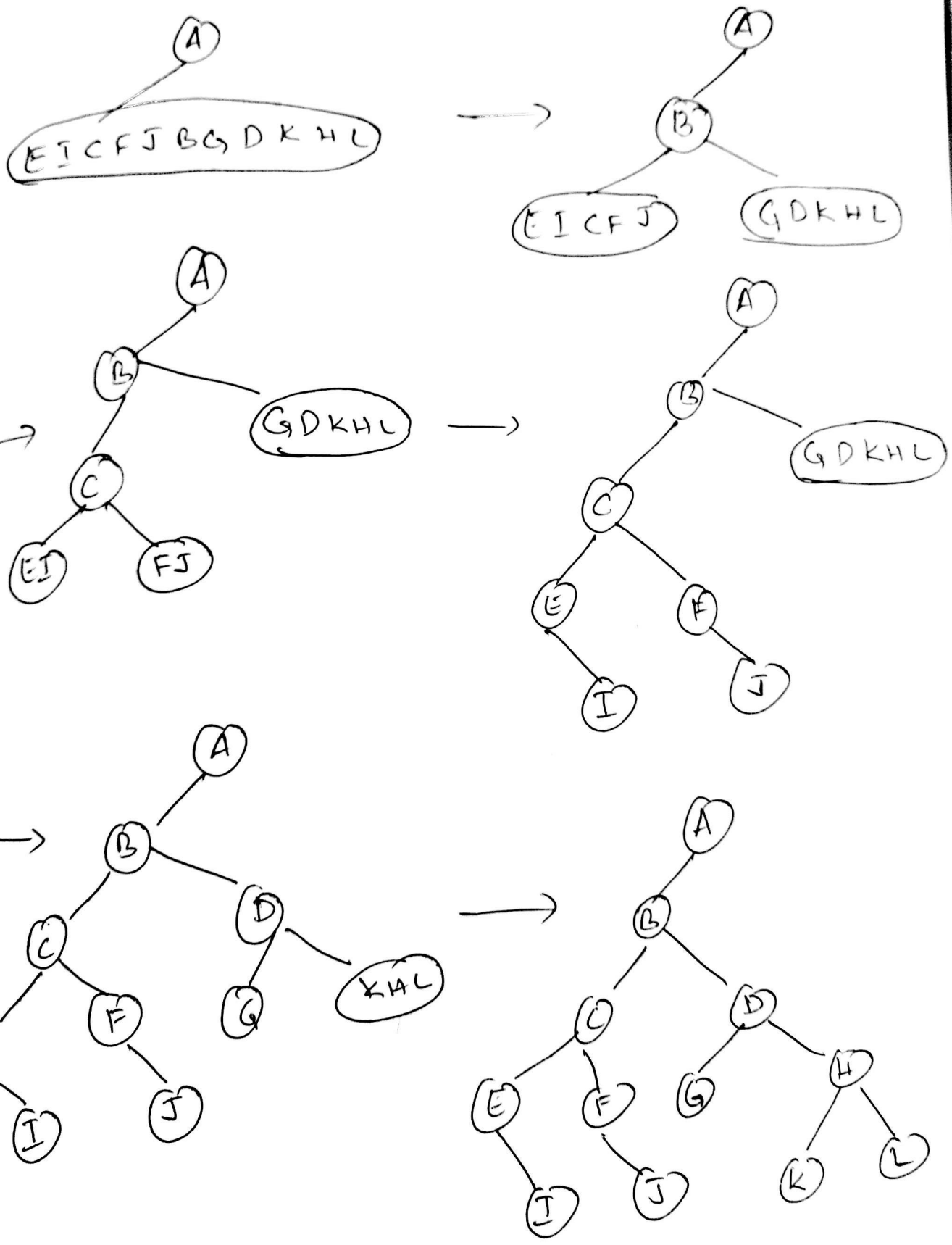






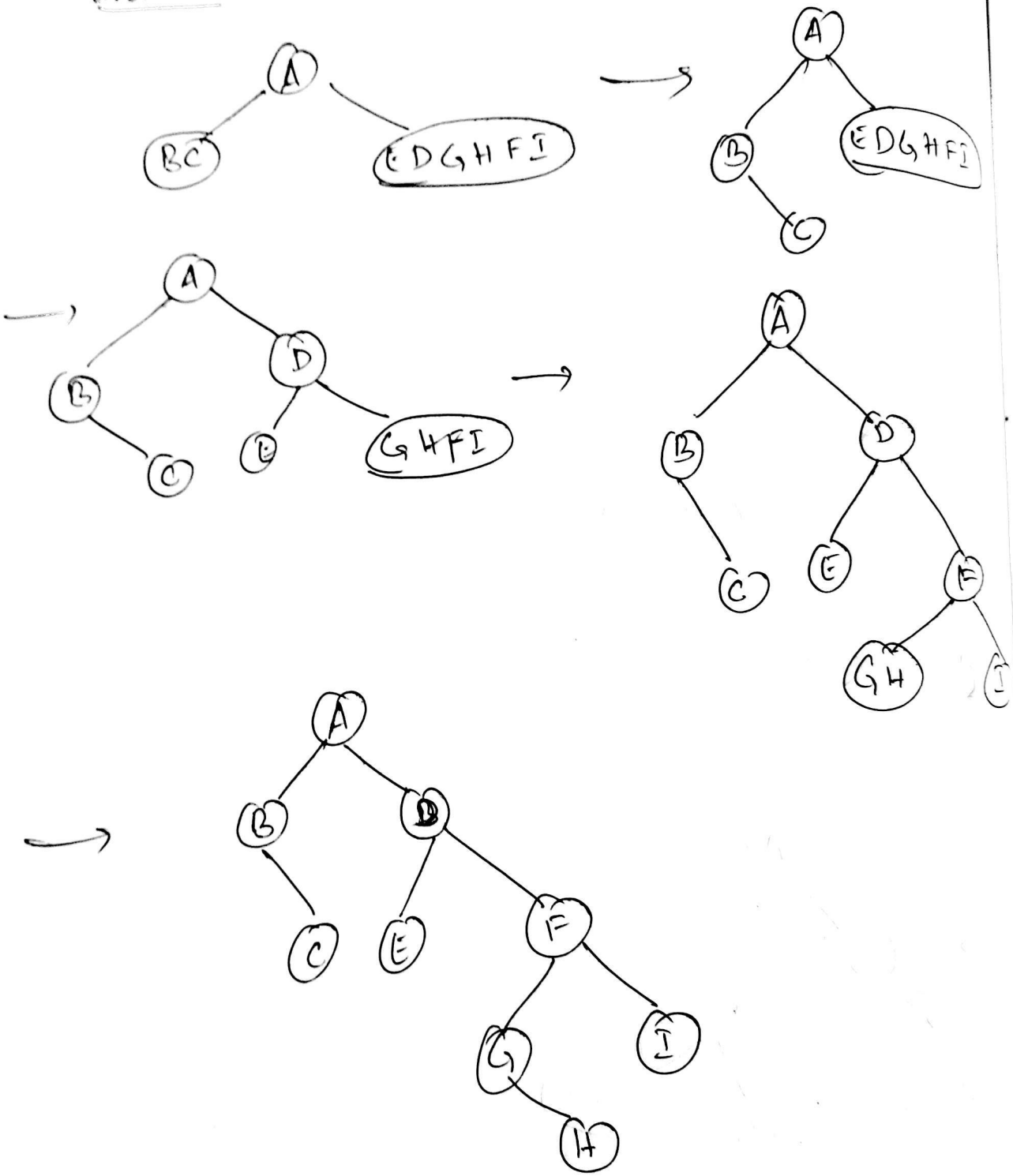
(9) Inorder: E I C F J B G D K H L [A]

Postorder: I E J F C G K L H D B [A]



(10) Preorder: A B C D E F G H I

Inorder: B C A E D G H F I



(11)

(12)

(13)

the

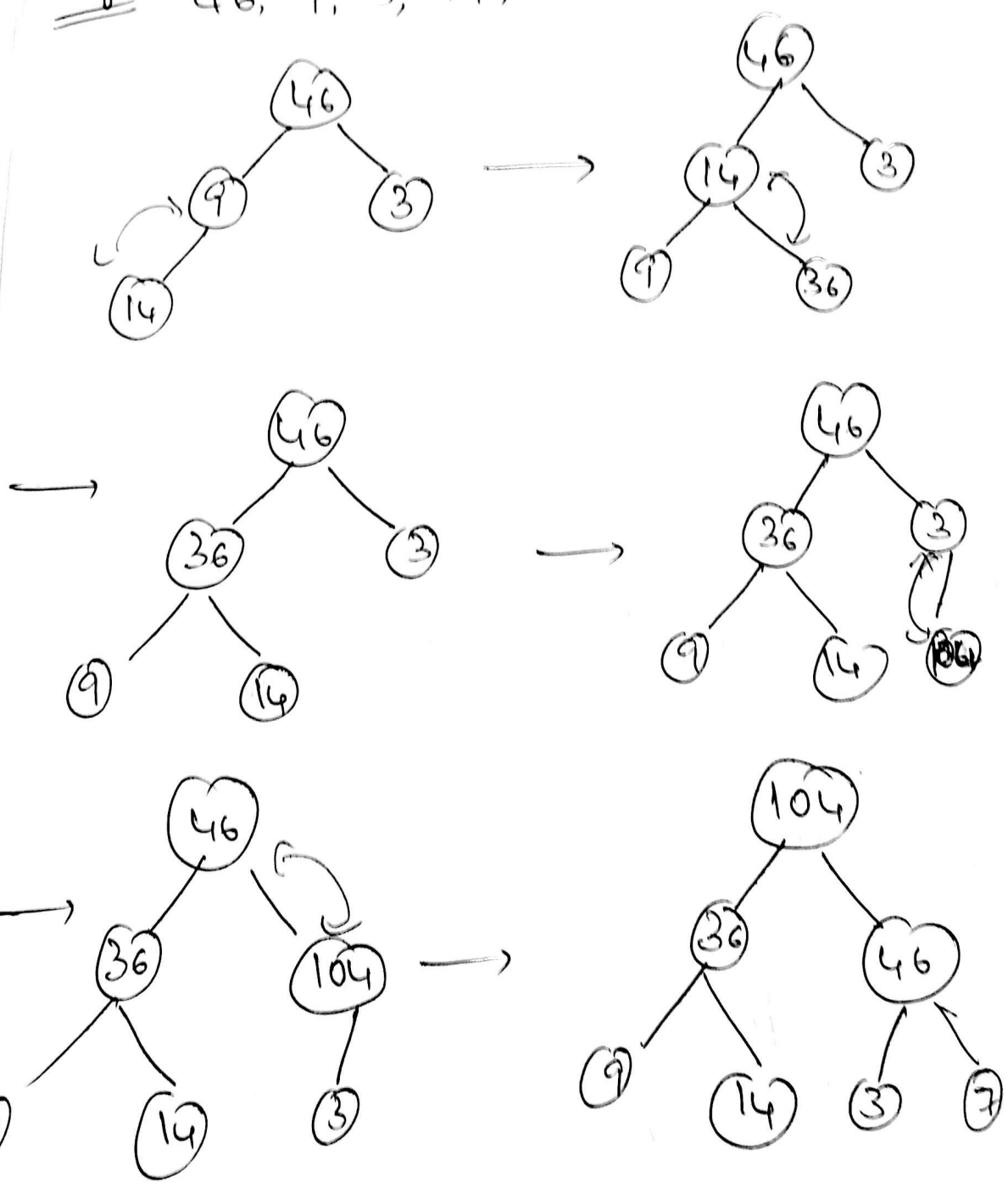
the

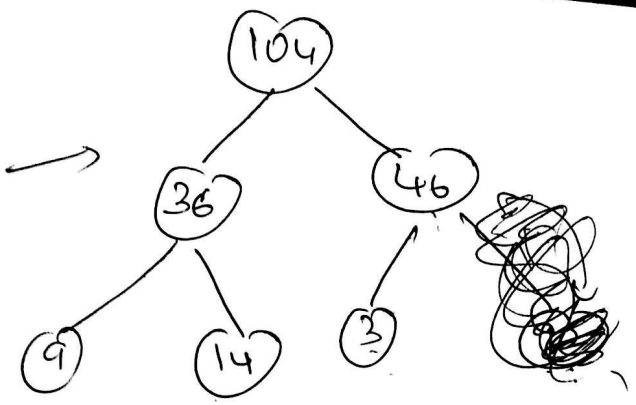
Max heap:

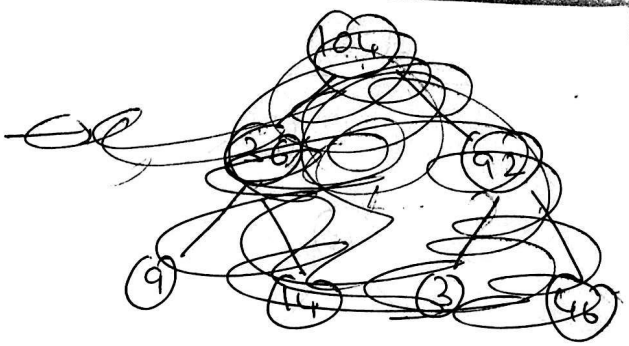
Or Descending order

Defn: A complete binary tree such that an item @ any given node is greater than or equal to the left & right child.

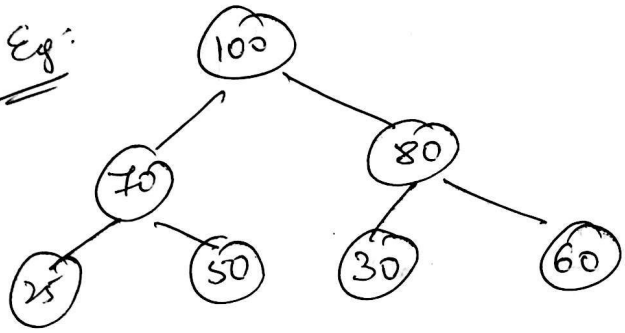
Eg: 46, 9, 3, 14, 36, 104, 7



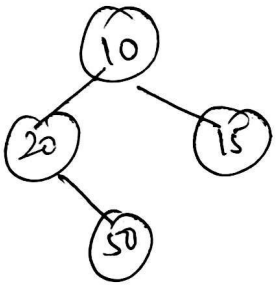


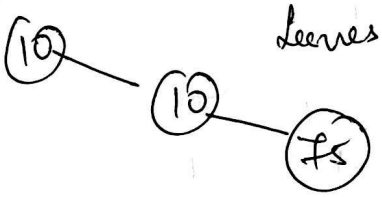


Eg:

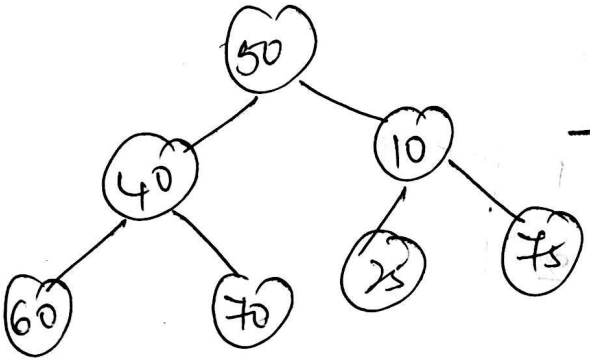


Not a heap examples:



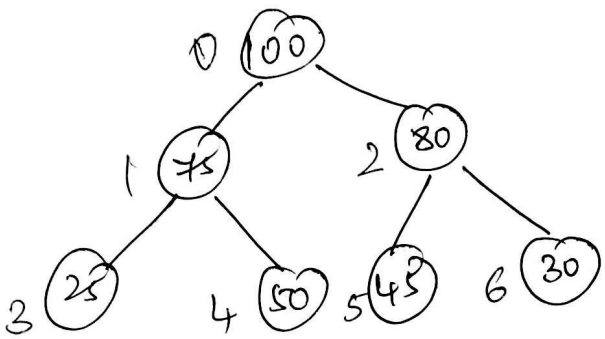


Leaves Not filled from left to right



→ neither ascending nor descending.

Representing nodes in a heap:



Nodes can be numbered from the root, one level @ a time.

In each level, nodes are numbered sequentially from left to right.

To obtain parent position:

If child node position = "c"

Position of parent, "p" = $\boxed{\frac{(c-1)}{2}}$

Eg: child 25, $c = 3$

$p = \frac{3-1}{2} = 1 \rightarrow 75$

To obtain child position:

If parent node = "p"

Left child = $\boxed{2p+1}$

Right child = $\boxed{2p+2}$

Selection Trees

Binary Search Trees (BST)

A BST is a binary tree in which for each node say X in the tree, elements in the left subtree are less than info(X) & elements in the right subtree are greater than info(X).

Every node should satisfy this condition, if left subtree or right subtree exists.

Insertion:

Step1:
```
temp = malloc(sizeof(struct node))
temp -> info = item;
temp -> llink = NULL;
temp -> rlink = NULL;
```

Step2:
```
if (root == NULL)          /* If tree does not exist */
    return temp;
```

Step3: prev & cur → 2 pointers
(prev → parent, cur → child)

Initialize
```
prev = NULL
cur = root
```

if item less than cur → put in left
else → right

```
while (cur != NULL)
{   prev = cur;
    if (item < cur -> info)
        cur = cur -> llink;
    else
        cur = cur -> rlink;
}
```

```
NODE insert (int item, NODE root)
{
    NODE temp, cur, prev;
    temp = malloc (sizeof (struct node));
    temp -> info = item;
    temp -> llink = NULL;
    temp -> rlink = NULL;
    if (root == NULL)
        return temp;
    prev = NULL;
    cur = root;
    while (cur != NULL)
    {
        prev = cur;
        if(item < cur -> info)
            cur = cur -> llink;
        else
            cur = cur -> rlink;
    }
    if (item < prev -> info)
        prev -> llink = temp;
    else
        prev -> rlink = temp;
    return root;
}
```

To check for duplicate items:

```
if (item == cur -> info)
{
    pf(" Duplicate item");
    free(temp);
    return root;
}
```

Search item:

```
NODE search (int item, NODE root)
{
    NODE cur;
    if (root == NULL)
        return NULL;

    cur = root;
    while ( cur != NULL )
    {
        if ( item == cur -> info)
            return cur;
        if (item < cur -> info)
            cur = cur -> llink;
        else
            cur = cur -> rlink;
    }
    return NULL;
}
```

Selection trees / Tournament trees:

Defn: A selection tree is a tree data structure which is used to select a winner in a knockout tournament.

The leaves of the tree represent 'n' players entering the tournament & each internal node represents a winner of a match.

Types: → Winner tree → min. winner tree, max. winner tree
→ Loser tree

Winner tree: A winner tree of 'n' players is a complete binary tree c 'n' leaf nodes & 'n-1' internal nodes.

Each internal node records the winner of the match.

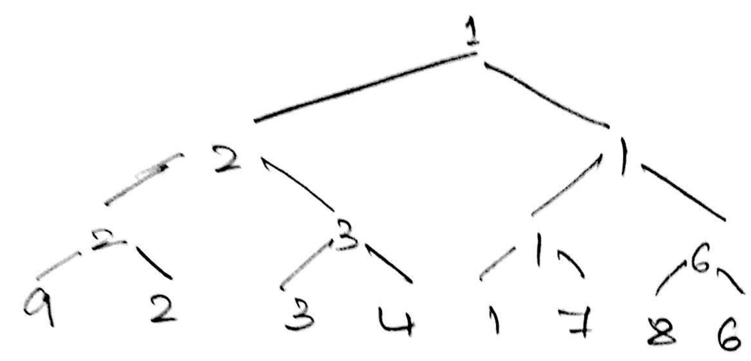
To determine the winner, we assume each player has a value.

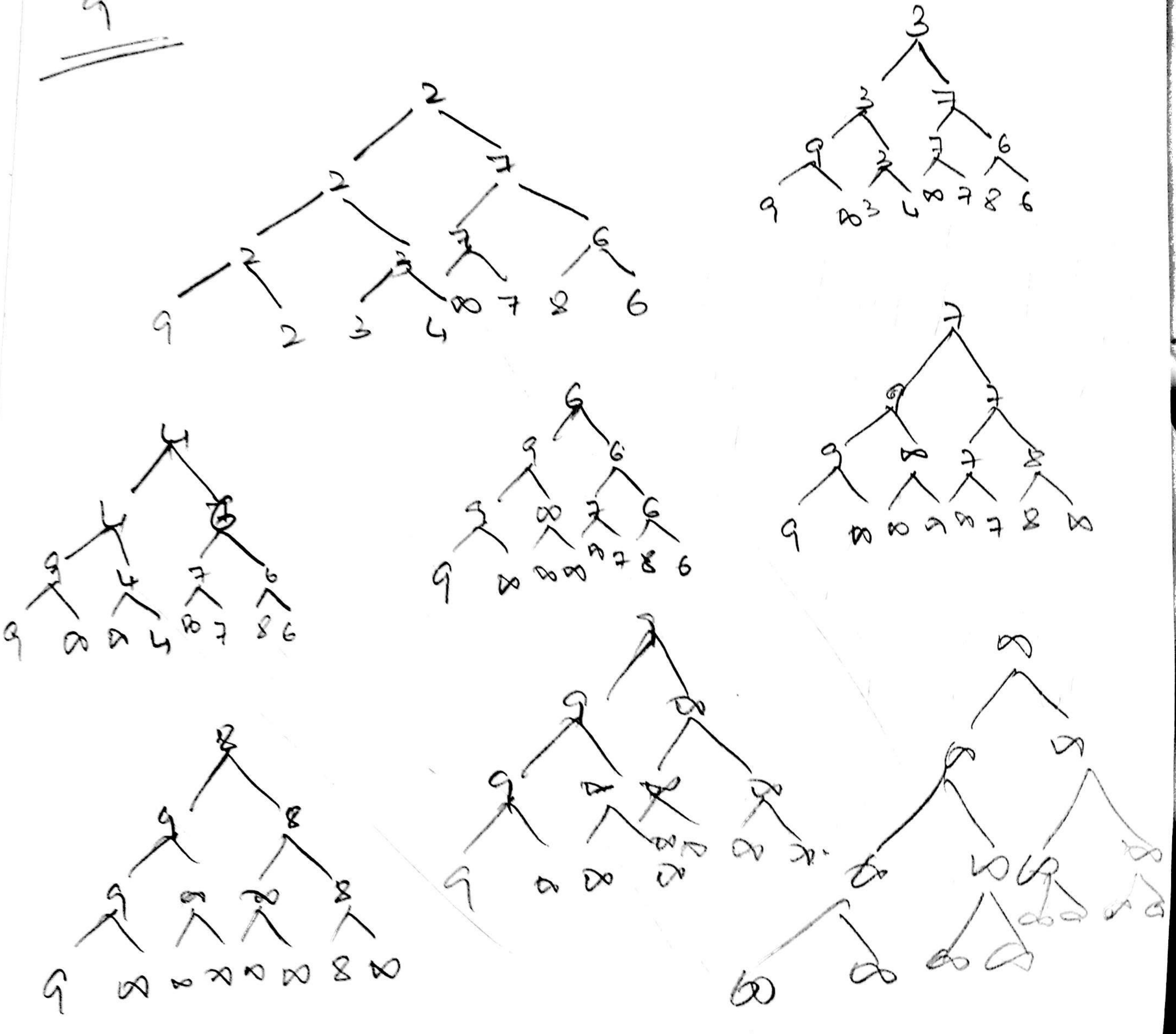
→ In a min-winner tree, each node represents the smaller of its 2 children i.e. the player c the smaller value wins.

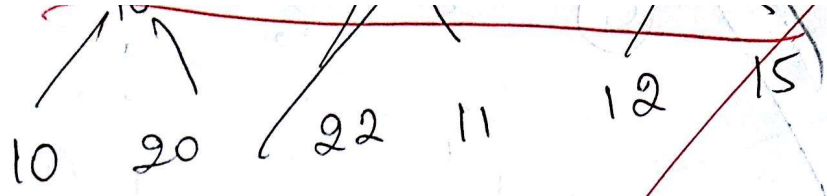
→ In a max-winner tree, each node represents the larger of its 2 children i.e. the player c the larger value wins.

(1)

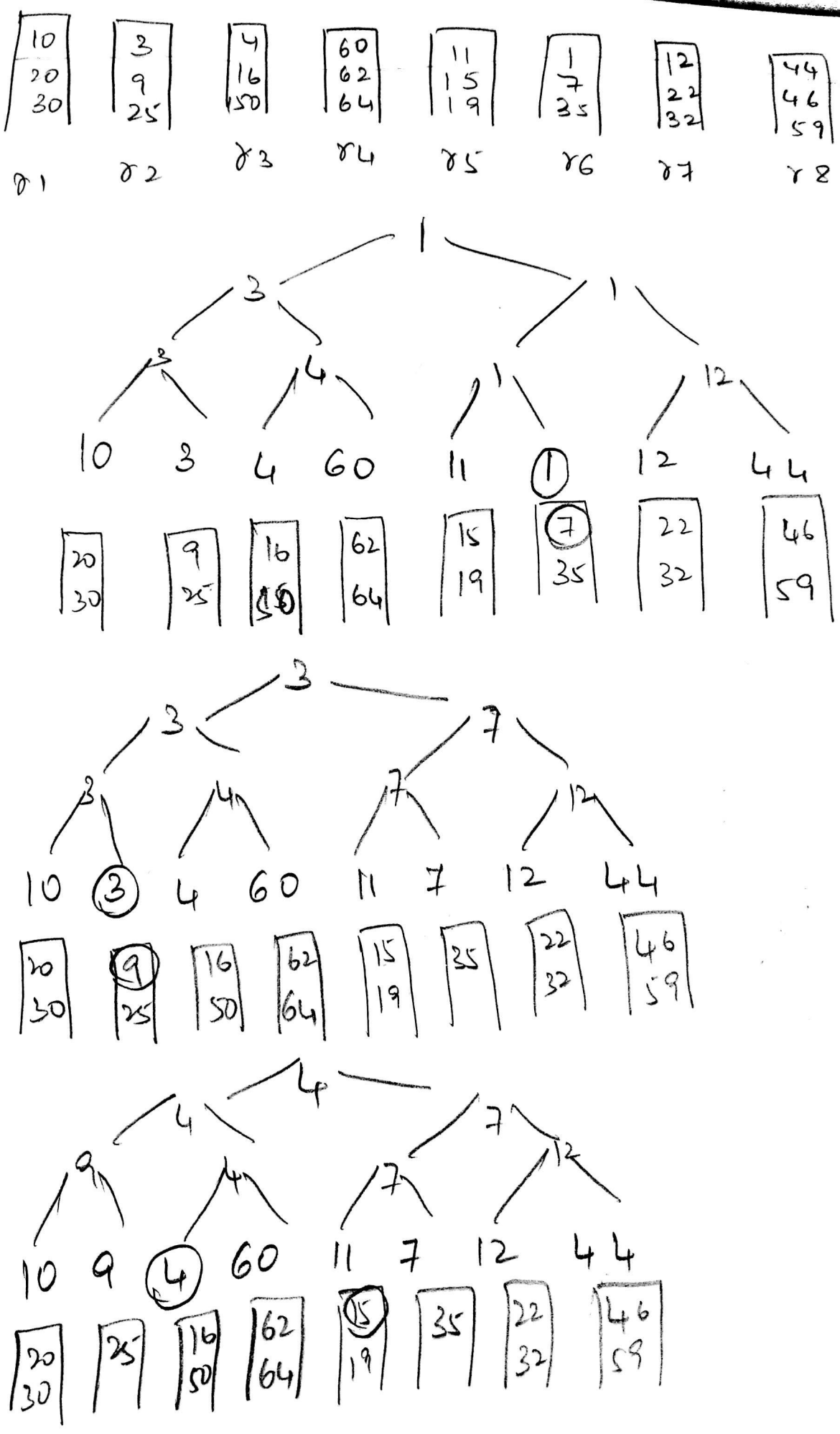
1
2
3
4
6
7
8
9

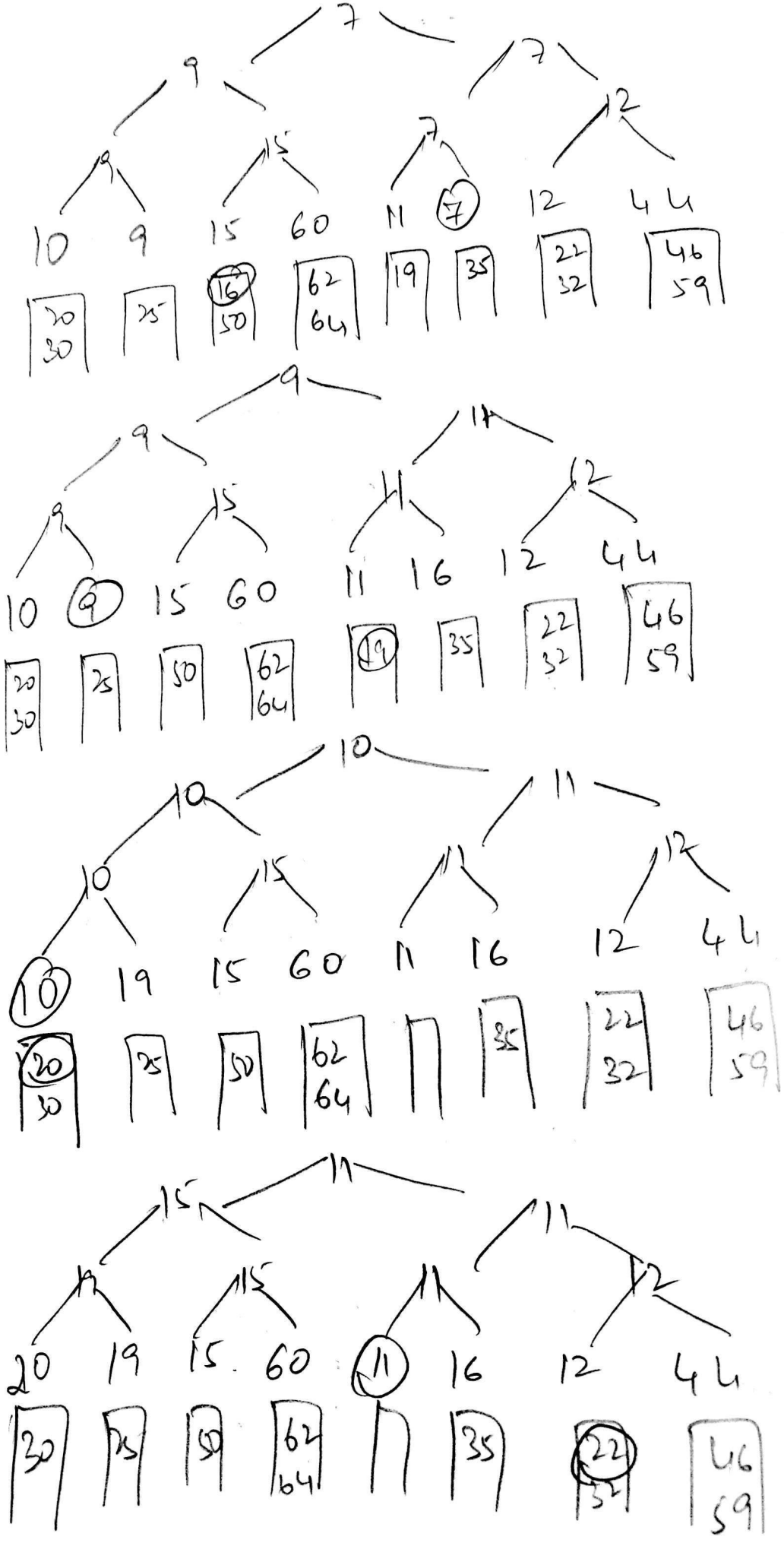


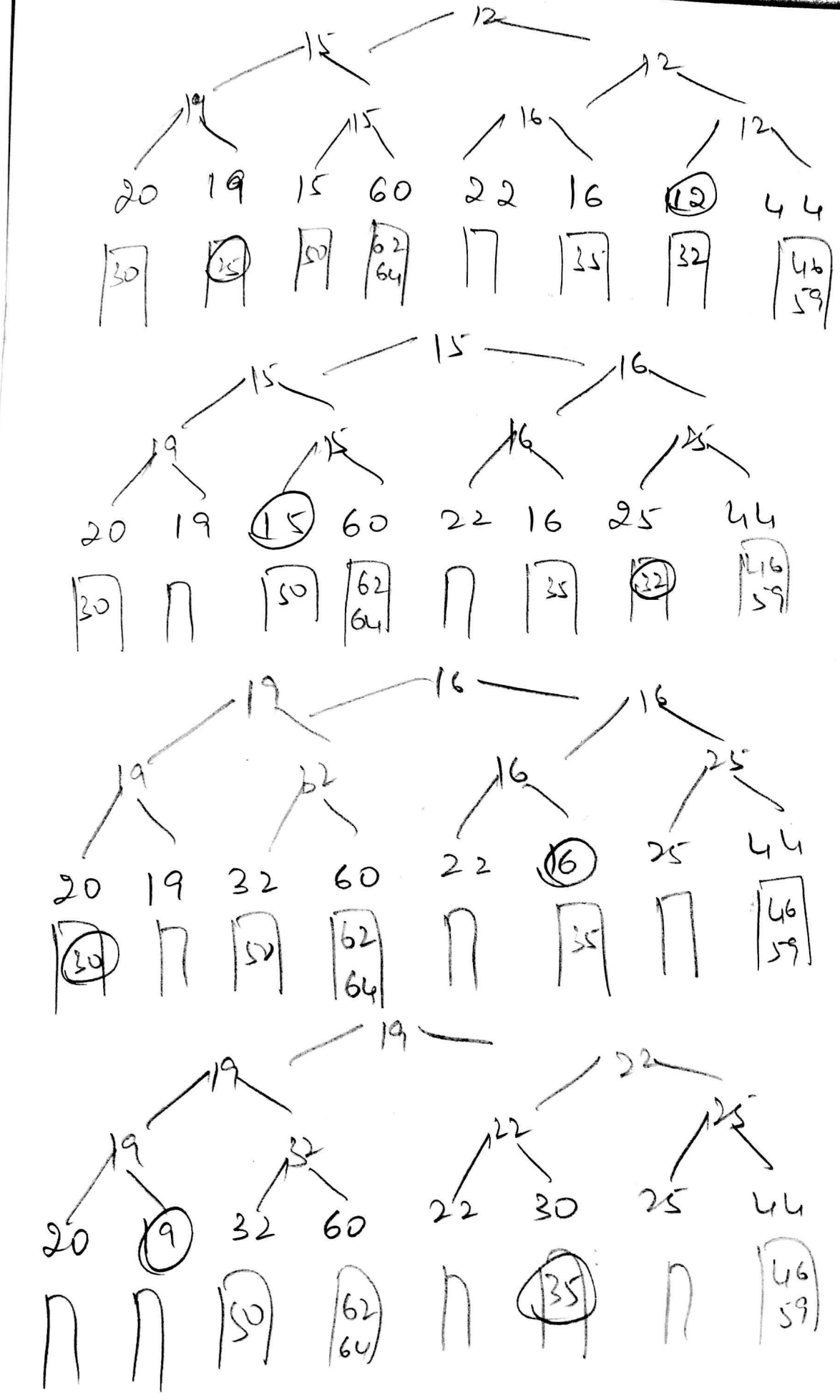


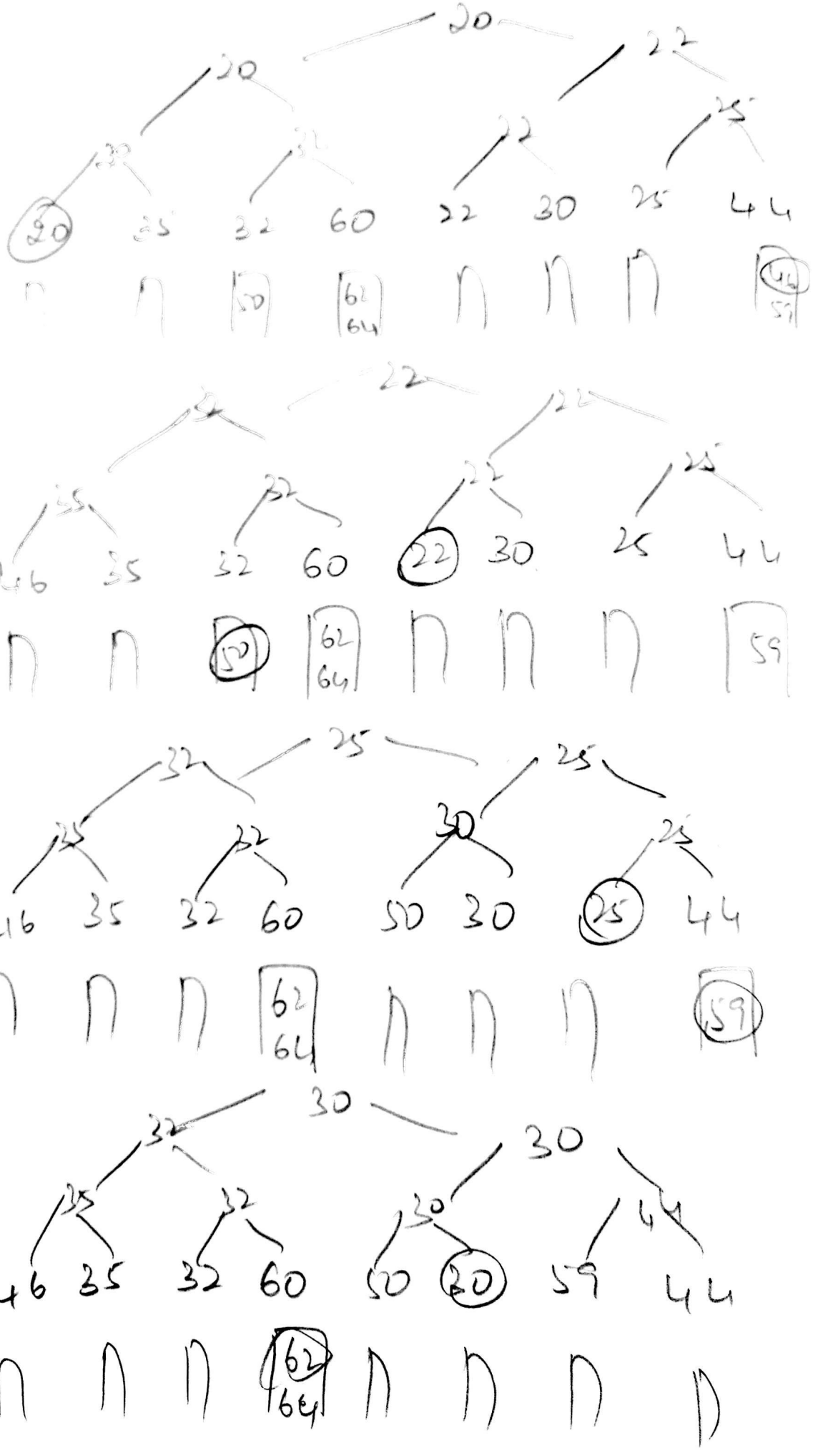


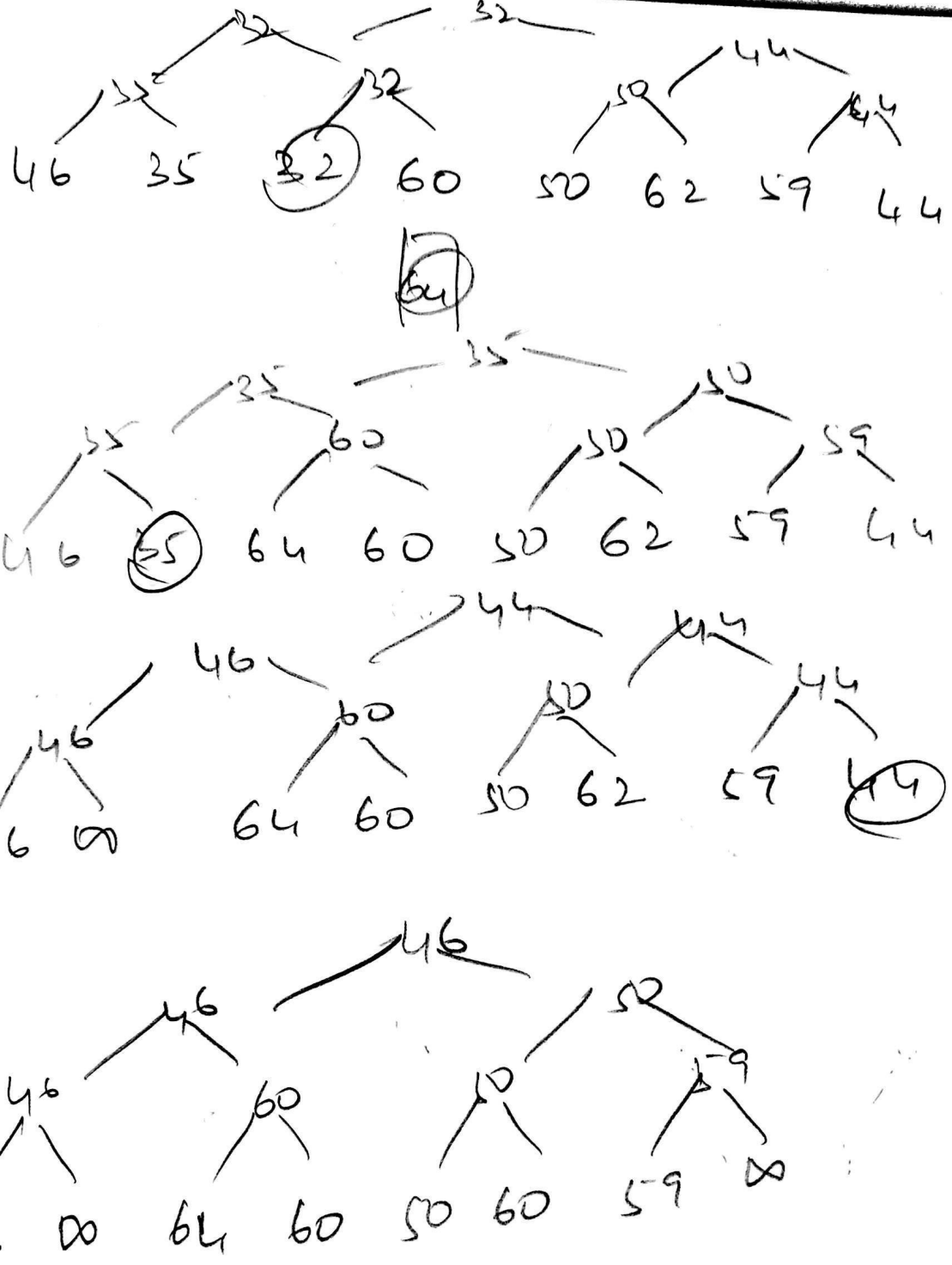
a[0] = 1
a[1] = 3
a[2] = 4
a[3] = 7
a[4] = 9
a[5] = 10
a[6] = 11
a[7] = 12
a[8] = 15
a[9] = 16
a[10] = 19
a[11] = 20
a[12] = 22
a[13] = 25
a[14] = 30
a[15] = 32
a[16] = 35
44
46



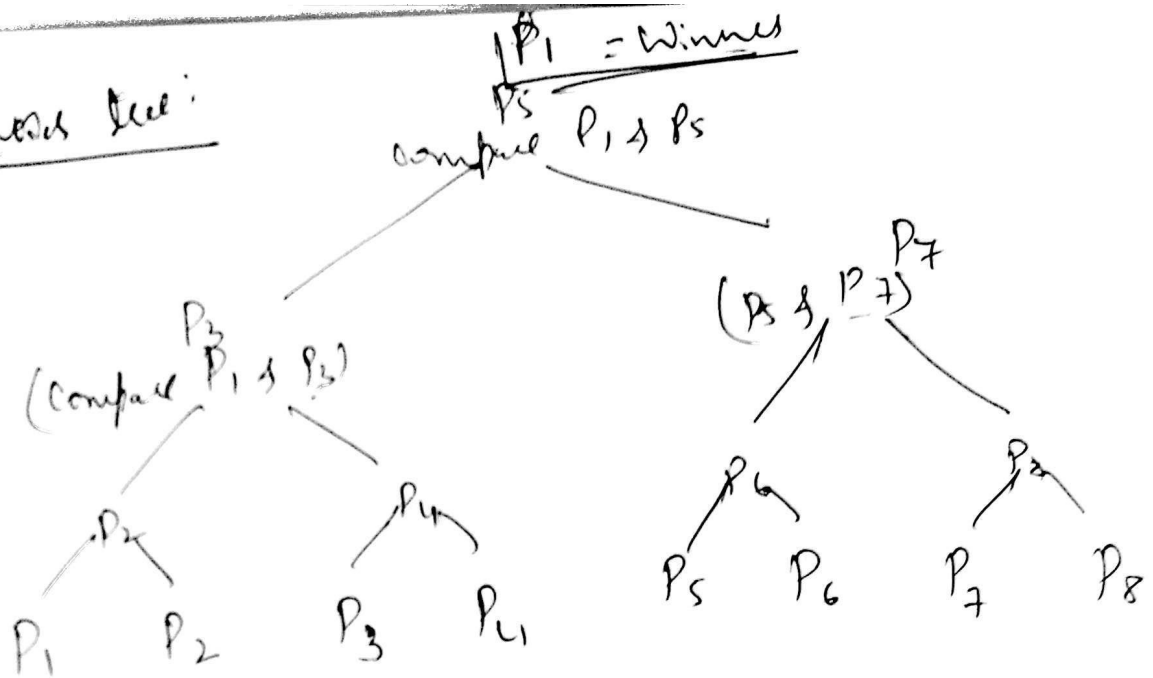


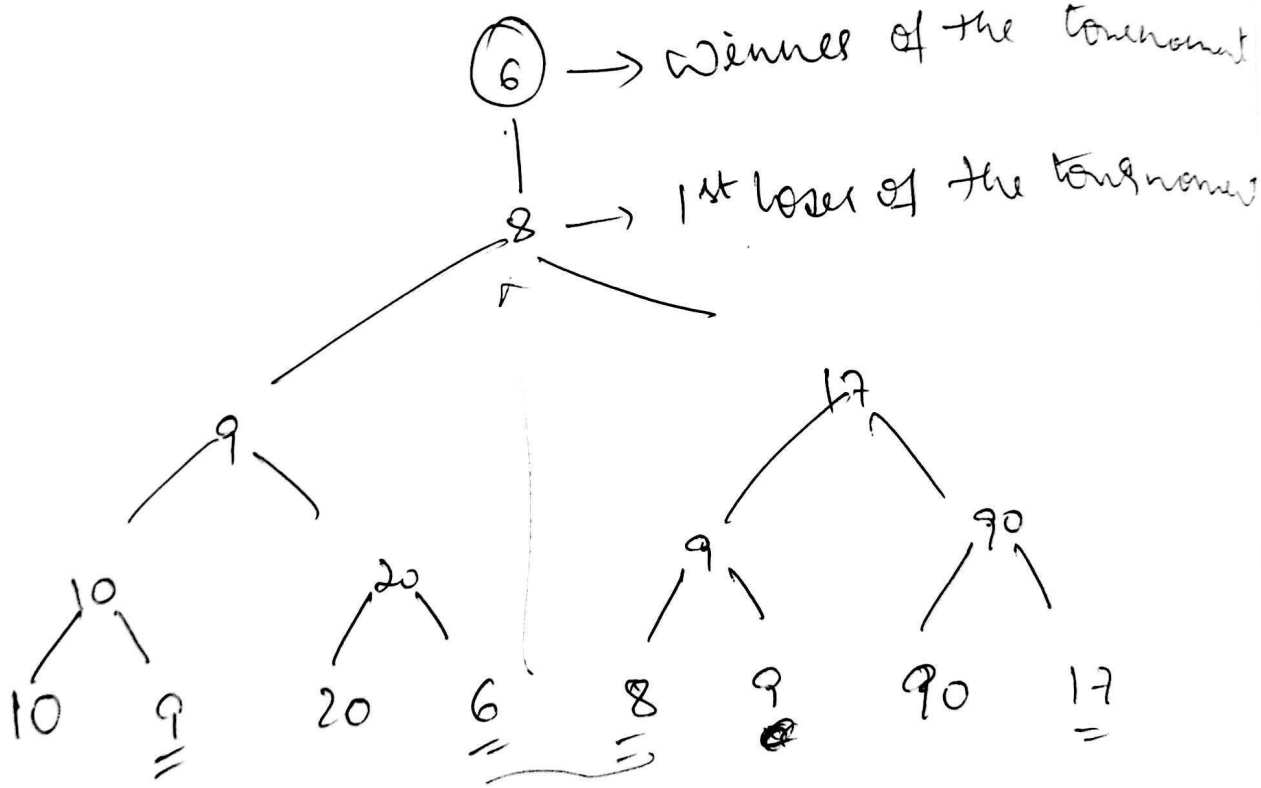






Loser tree:





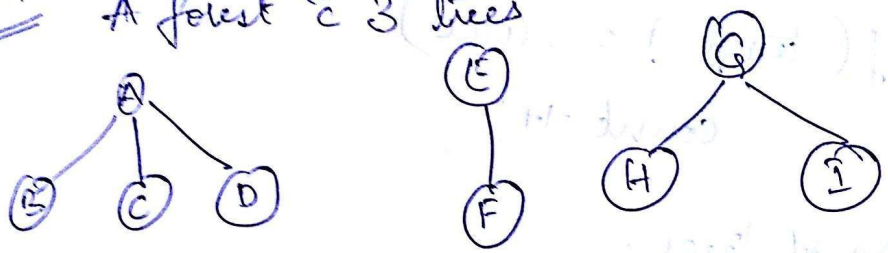
A loser tree for n players is also a complete binary tree with 'n' leaf nodes & n-1 internal nodes.

Each internal node records the loser of the match. The final player who has not lost any match is the Winner.

Forests:

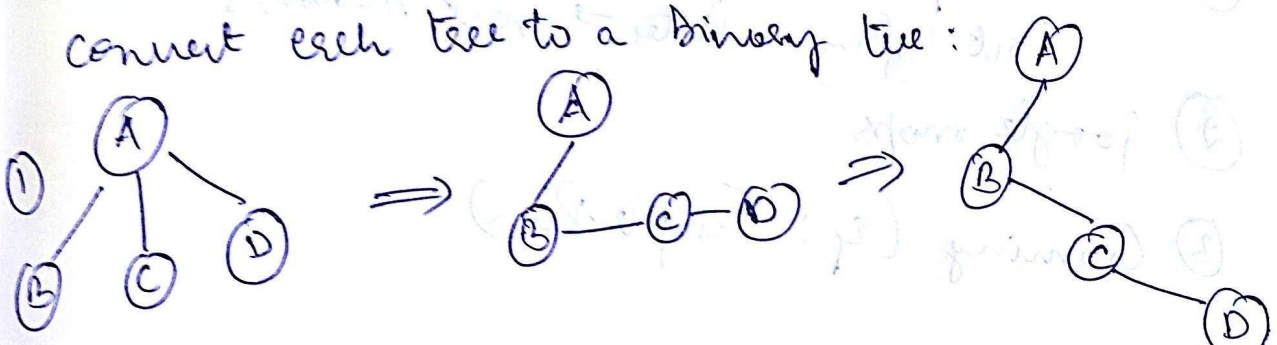
Collection of zero/more trees is called a forest.

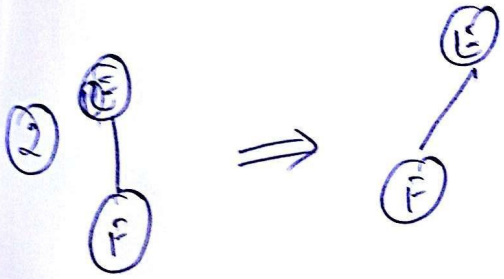
Eg: A forest of 3 trees

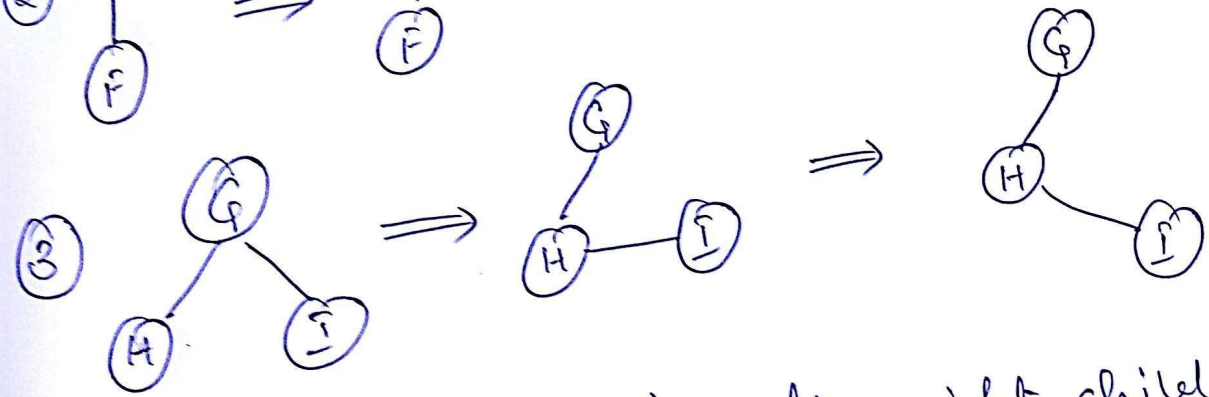


Transforming a forest into a binary tree:

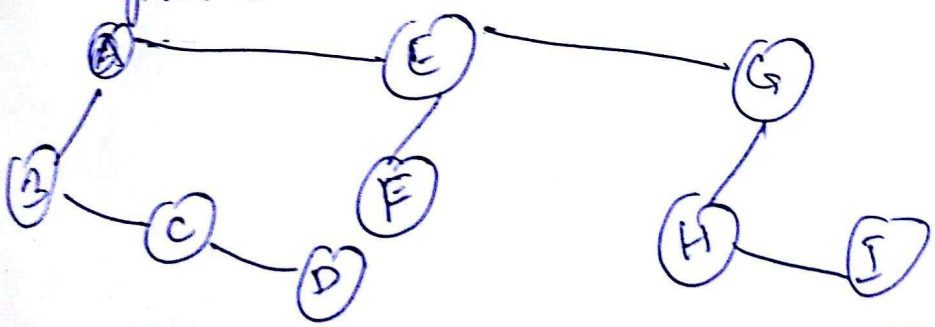
Convert each tree to a binary tree:







Attach each tree (root) as the right child of the previous tree.



Counting Binary trees:

Counting the no of nodes in a tree.

→ Use "count" variable.

```
count = 0;
temp = root;
if( temp != NULL)
    count ++;
```

Application of trees:

1. Search operation
   Start typing a letter → rest follows
2. Google maps
3. Gaming (Eg: Temple Run)