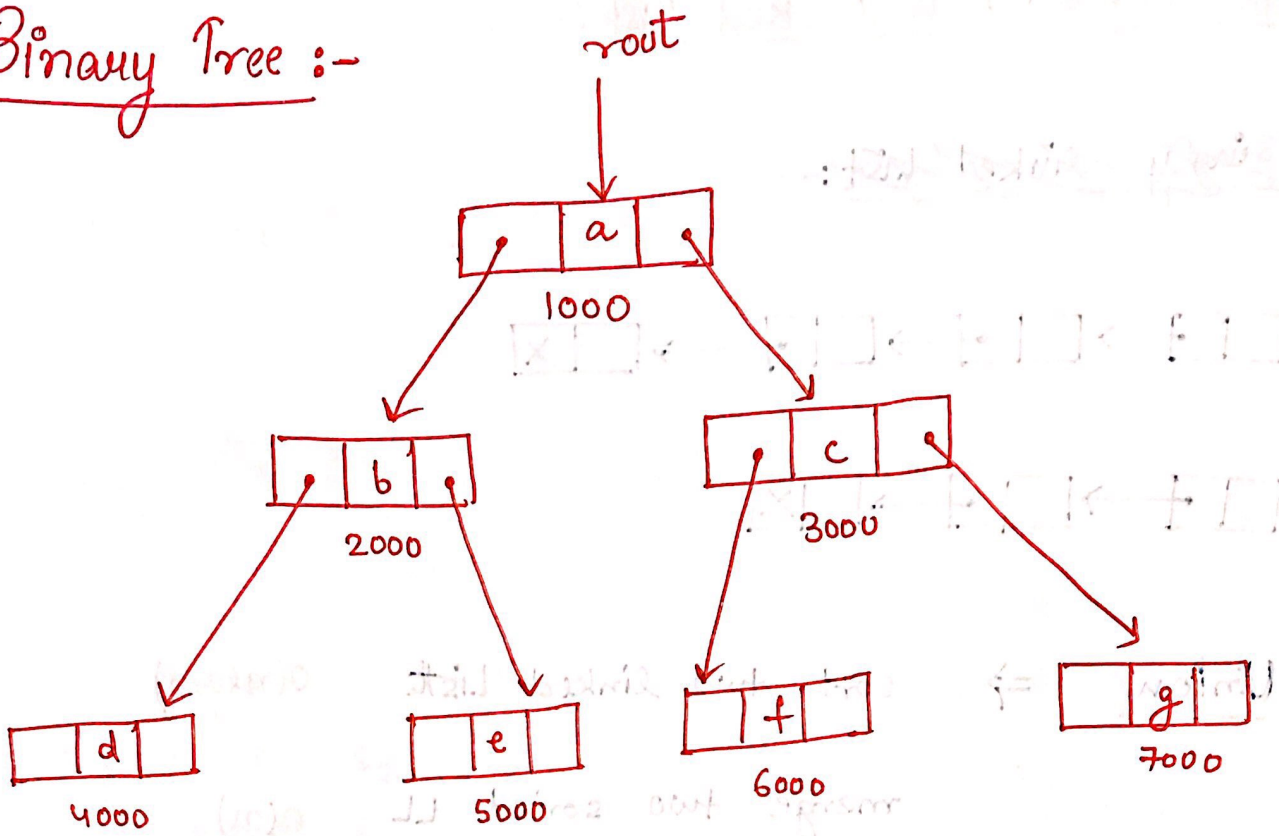
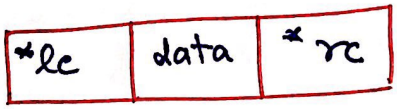
⇛ Binary Tree :-



- Binary tree are implimented as DLL.
- Each node of DLL points to 2 new child node.

```
Struct BTnode
{
    Struct BTnode   *lc;
    Int data;
    Struct BTnode   *rc;
};
```



Q. In a Binary tree :

no. of nodes of degree 1 = 5

" " 2 = 10

Find no. of leaf nodes

So, Sum of degree = no. of edges = 25

$e = 25$

So, total nodes = $e+1$ = 26 (Spanning Tree)

Leaf nodes = 26 − non leaf (5 + 10)

= 26 − 15

= 11

Q. Write a c program to count no. of leaf nodes in the given B-Tree. (Recursive program)

```
int countleafnode ( struct BTnode *r)        —— T(n)
{
  if(r == NULL)
    return (0);                              —— O(1)

  if(r->lc == NULL && r->rc == NULL)
    return (1);                              —— O(1)

  else
  {
    a = countleafnode (r->lc);               —— T(n/2)
    b = countleafnode (r->rc);               —— T(n/2)
    c = a+b;                                 —— O(1)
    return c;
  }
}
```

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

$T(n) = O(n)$ every case

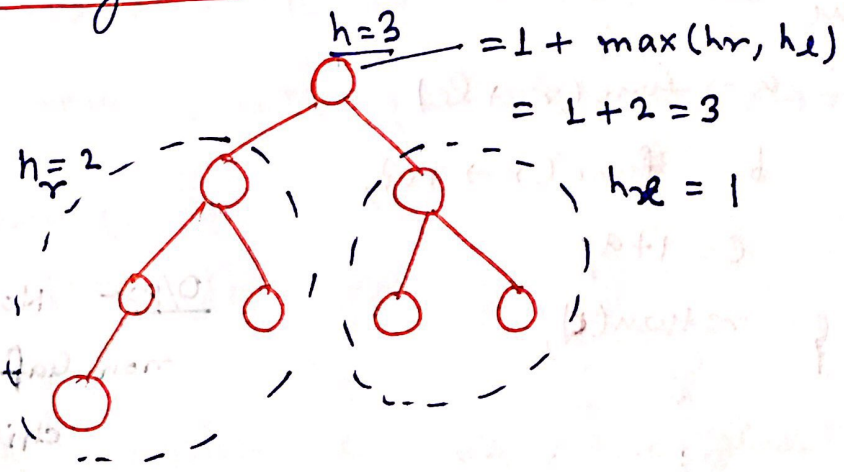
Leaf count Algo B-Tree	Time complexity	space complexity (STACK)
BEST	O(n)	O(logn)
AVG	O(n)	O(logn)
WORST	O(n)	O(n)

Q. Write a c program to count no. of non-leaf nodes.

```
int count_nonleaf ( Struct BTree *r)
{  if ( r == NULL)
   {
      return (0);
   }
   if ( r->lc == NULL && r->rc == NULL)
      return (0);
   else
   {
      a = count_nonleaf ( r->lc);
      b = count_nonleaf ( r->rc);
      c = 1 + a+b;
      return c;
   }
}
```

Q. Write a c program to find height of a given B-Tree.

- Height of a node is equal to the longest distance to reach leaf node.
- Height of root node is known as height of ~~root~~ Binary tree.




```
int Height ( struct BTree * r)
{
    if (r == NULL)
        return (0);
    if (r->lc ==0 && r->rc ==0)
        return (0);

    else
    {
        a = Height (r->lc);
        b = Height (r->rc);
        c = 1 + max (a,b);
        return (c);
    }
}
```

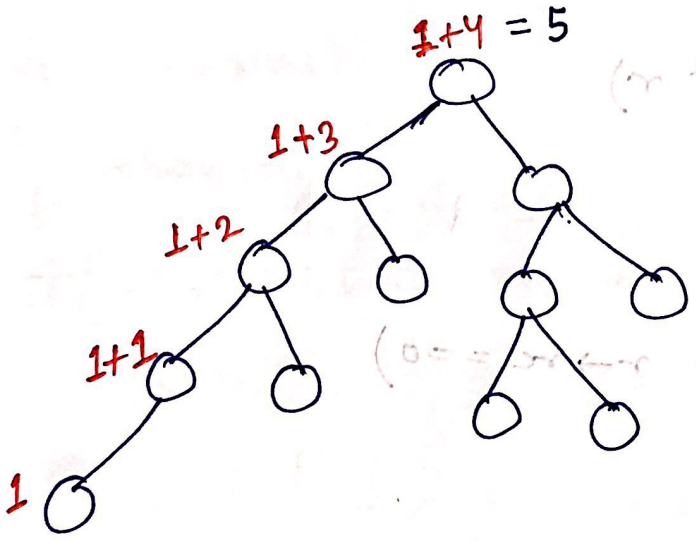
[T(n) = O(n)]

Q. what is the o/p:-

```
int func(r)
{  if( r == NULL)
       return(0);
   if (r→lc == NULL && r→rc == NULL)
     return(1);

   else
   {
       a = func(r→lc);
       b = func(r→rc);
       c = 1+a;
       return(c);
   }
}
```

O/P:- No. of non leaf nodes in left child + 1



Q. Write a c program to verify given B Tree is strict B Tree or not.

[strict B-Tree : 0 or 2 child]

```
int
    SBT ( Struct Btree * r)
{
        if ( r == NULL)
            return (1);
        if ( r -> lc == NULL  &&  r -> rc == NULL)
            return (1);
        if ( r -> lc ! NULL && r -> lc ! = NULL)
            return ( SBT(r->lc) && SBT( r -> rc));
        return (0);
}
```

Q. Write a program to ver two given B-Trees equal or not

```
Equal ( r1 , r2)
{
    if( r1 == NULL && r2 == NULL)
        return(1);
    if( (r1 == null && r2 != null) || (r1 != null && r2 == null))
        return (0);
    if( r1 -> data == r2 -> data)
        return ( Equal(r1->lc, r2->lc) && Equal
                                    (r1->rc, r2->rc));
    return (0);
```

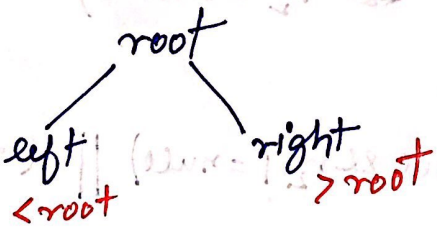
Q. Write a program to convert the given binary tree in its mirror image.

```
mirror_img(r)
{
    if ( r == null)
        return(r);

    if( r→lc == null && r→rc == null)
        return(r);
    else
    {
        r→rc = mirror_img(r→lc);
        r→lc = mirror_img(r→rc);
        return r;
    }
}
```

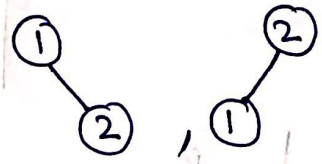
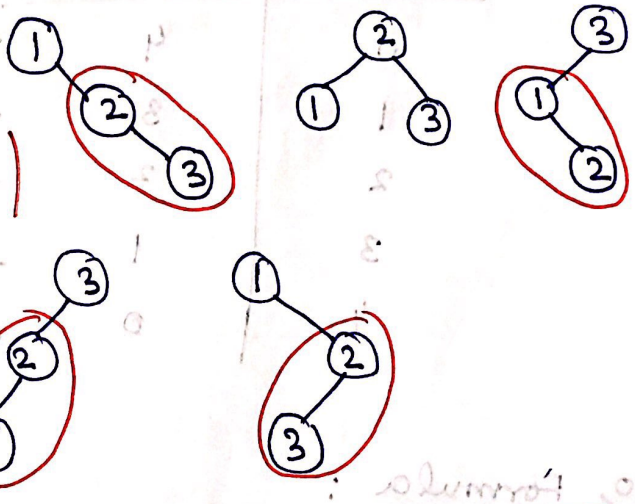
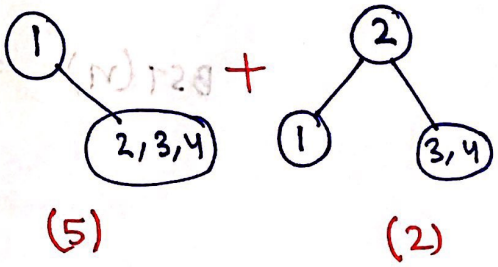
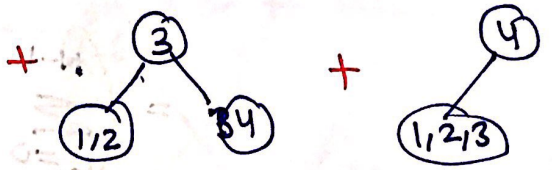
O(n)

Binary Search Tree :-

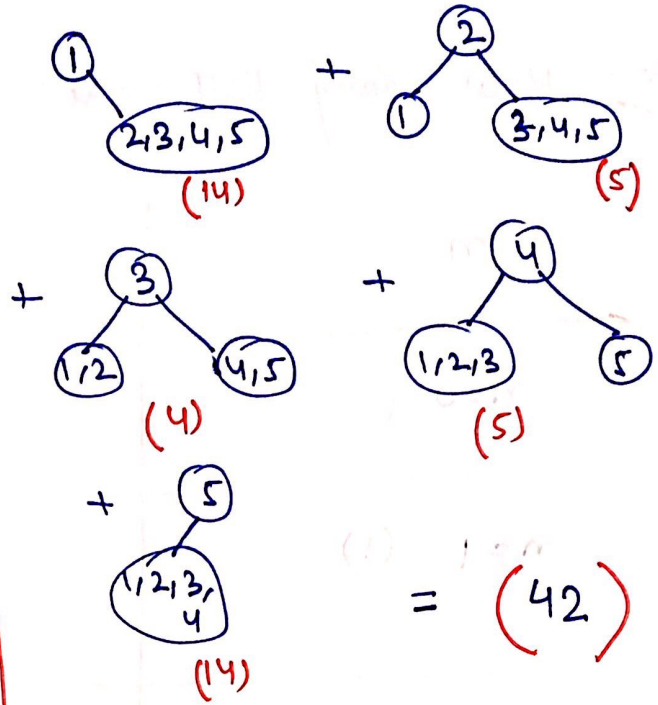


- All elements on left of root are smaller than root
- " right " larger than root

Q. How many BST are possible with n nodes.

n	no. of BST
n=0	(1) empty BST
n=1 (1)	(1) BST with root only
n=2 (1,2)	(2) 
n=3 (1,2,3)	(5) 
n=4 (1,2,3,4)	(14)  (5) (2)  (2) (5)

$n = 5$ (1, 2, 3, 4, 5)



$= (42)$

So,

L	R		
0	4	→	BST(0) * BST(4)
1	3	→	BST(1) * BST(3)
2	2	→	BST(2) * BST(2)
3	1	→	BST(3) * BST(1)
4	0	→	BST(4) * BST(0)

So, Formula :

$$BST(n) = \sum_{L=0}^{n-1} BST(L) * BST(R)$$

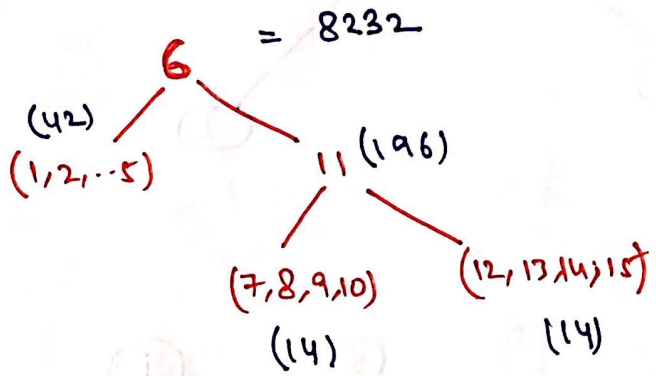
$$L + R + 1 = N$$

$$R = N - (L+1)$$

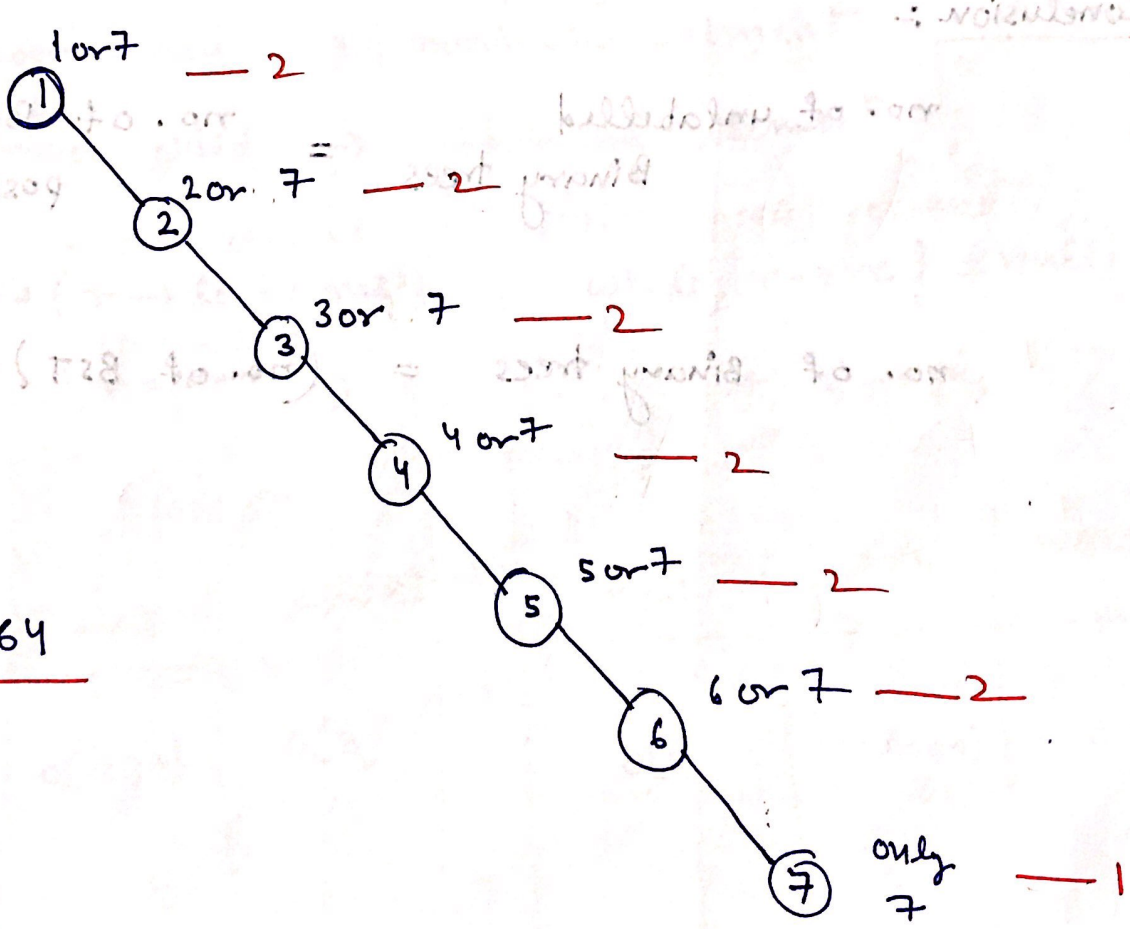
$$= \sum_{L=0}^{N-1} BST(L) * BST(N-L-1)$$

$$= \frac{2n C_n}{n+1}$$

Q. No. of BST possible in 15 nodes (1, 2, 3, ... 15). in such a way that 6 will be the root such that



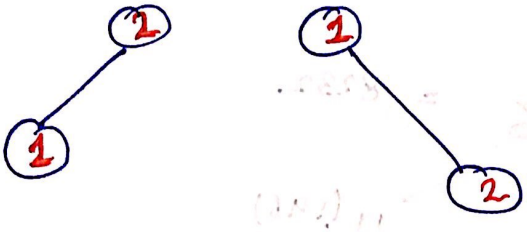
Q. No. of BST possible with 7 nodes, where height of every BST is 6.



$2^6 = 64$

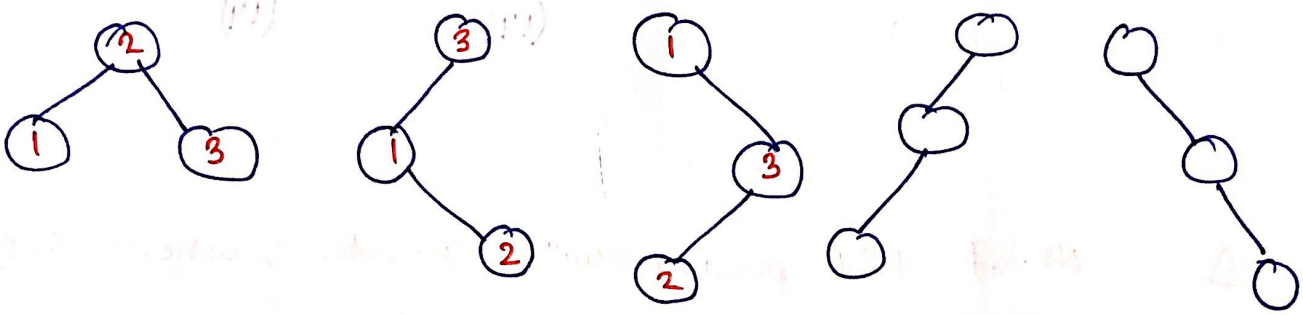
Q. How many unlabelled Binary trees possible with n nodes.

n=2



(2)

n=3

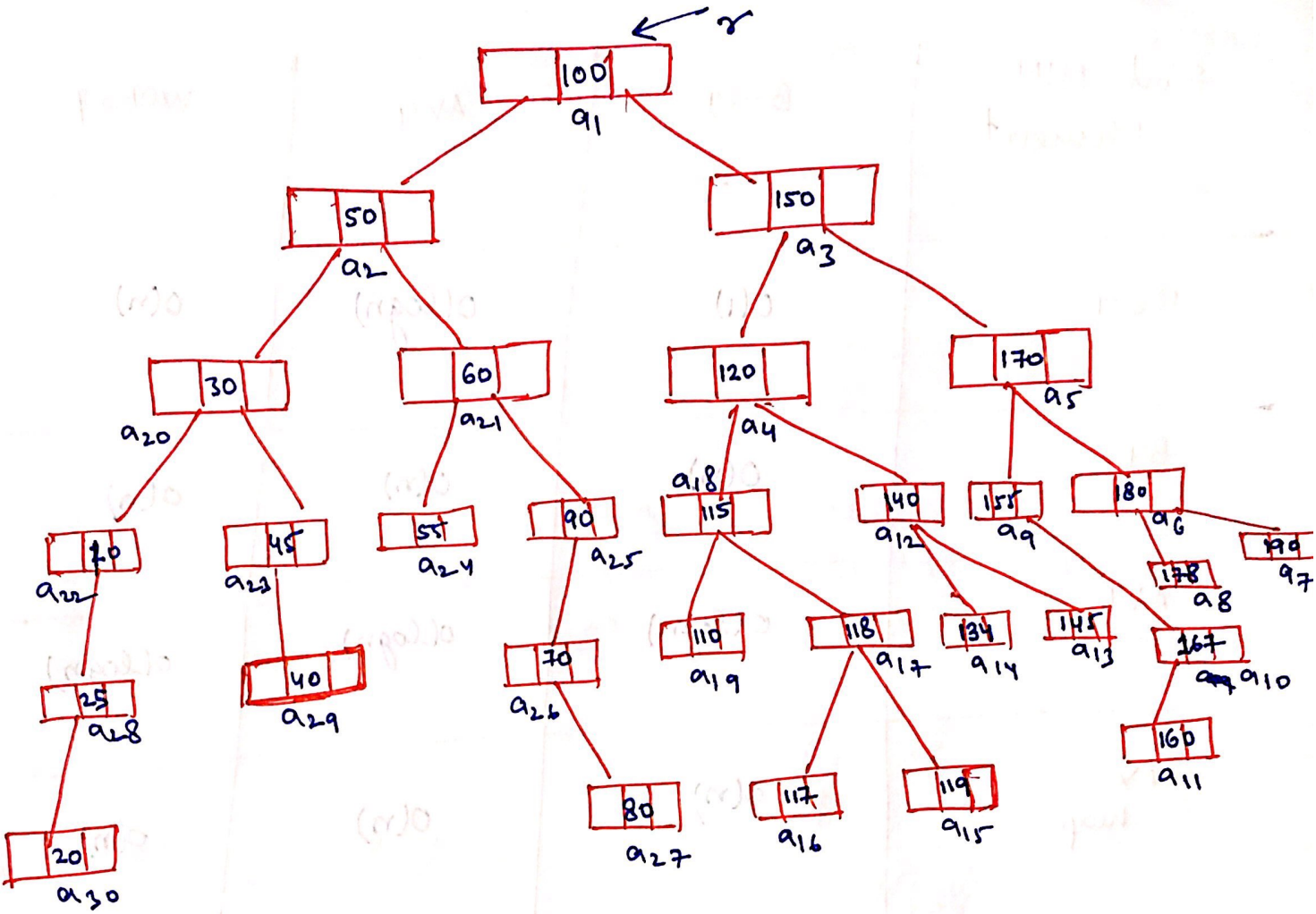


(5)

Conclusion :-

- no. of unlabelled Binary trees = no. of BST possible

- no. of Binary trees = (no. of BST) * n!



- Leftmost child → minimum element

  Rightmost child → maximum element

T(n) — MIN element

```
while (r → lc != null)
{
    r = r → lc;
}
```

BEST	AVG	WORST
O(1)	O(logn)	O(n)

T(n) — MAX element

```
while (r → rc != null)
{
    r = r → rc;
}
```

BEST	AVG	WORST
O(1)	O(logn)	O(n)

Find MIN Element	BEST	AVG	WORST
BST	O(1)	O(logn)	O(n)
BT	O(n)	O(n)	O(n)
AVL	O(logn)	O(logn)	O(logn)
MAX heap	O(n)	O(n)	O(n)

Find element X	BEST	AVG	WORST
BST	O(1)	O(logn)	O(n)
BT	O(1)	O(n)	O(n)
AVL	O(1)	O(logn)	O(logn)
MAX heap	O(1)	O(n)	O(n)

Find Element X not exist	BEST	AVG	WORST
BST	O(1)	O(logn)	O(n)
BT	O(n)	O(n)	O(n)
AVL	O(logn)	O(logn)	O(logn)
MAX heap	O(1)	O(n)	O(n)

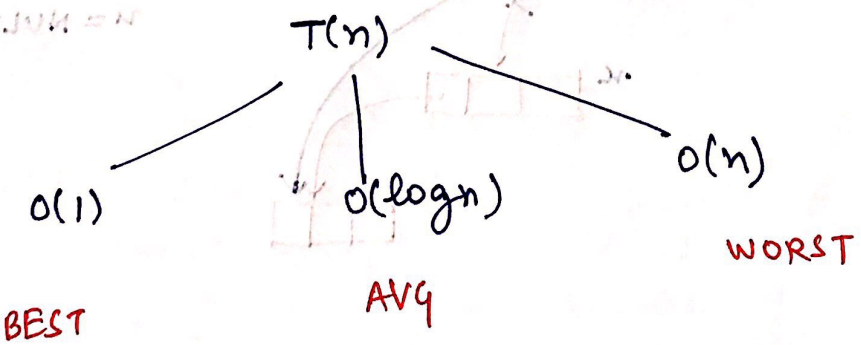
Insertion in BST:-

(i) Create node - x — O(1)

(ii) Find correct place for x to be inserted — O(1) BEST, O(logn) AVG, O(n) WORST

(iii) Connet node - x with pointer. — O(1)

So,



= Insertion will be done at NULL, whether x (is value close to root or not.

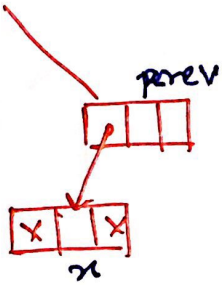
Deletion in BST :-

Inorder successor → Left most element of right side

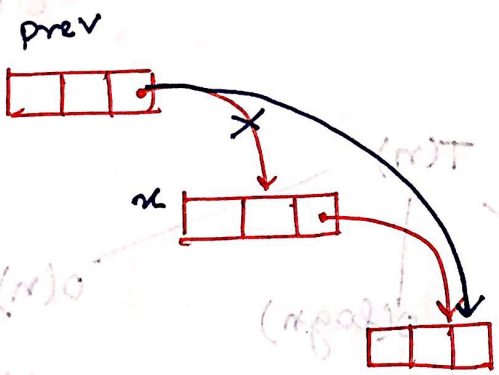
Inorder predecessor → Rightmost element of left side

(i) **Delete leaf node:-**

```
prev → left = NULL;
free (x);
x = NULL;
```

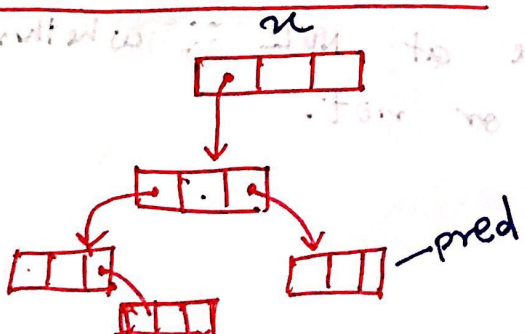


(ii) **Delete node with 1 child :-**




```
prev → rc = x → rc;
free (x);
x = NULL;
```

(iii) **Delete node with 2 child:-**



predecessor / succes

```
x → data = pred → data;
delete (pred);
```

Algo:-

(i) find - x;

(ii) 0 - child ⇒ simply delete by adjusting two links

(iii) 1 - child ⇒ simply delete by connecting grandfather with grandchild

(iv) 2 - child ⇒ replace x data by precessor / successor then delete (predecessor / successor).