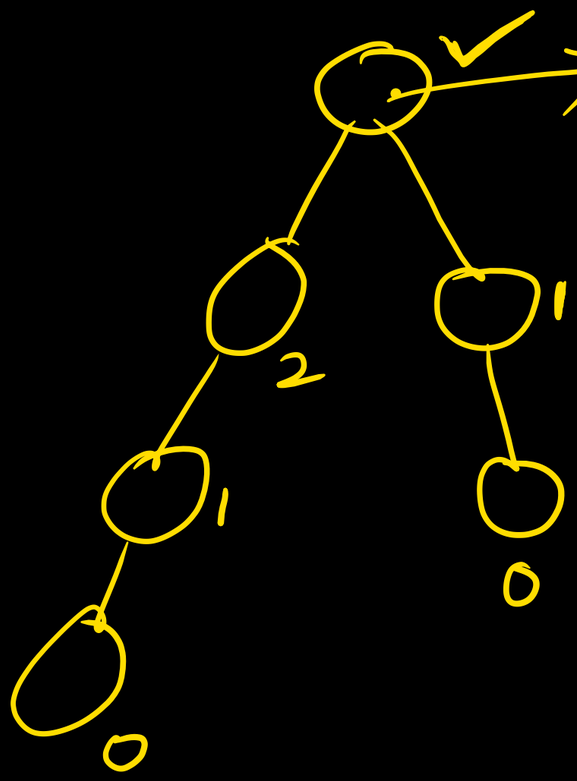
Write an algo to find the height of a tree. assuming height of a leaf is '0'.

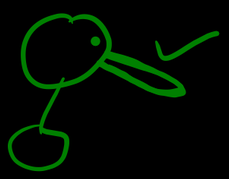


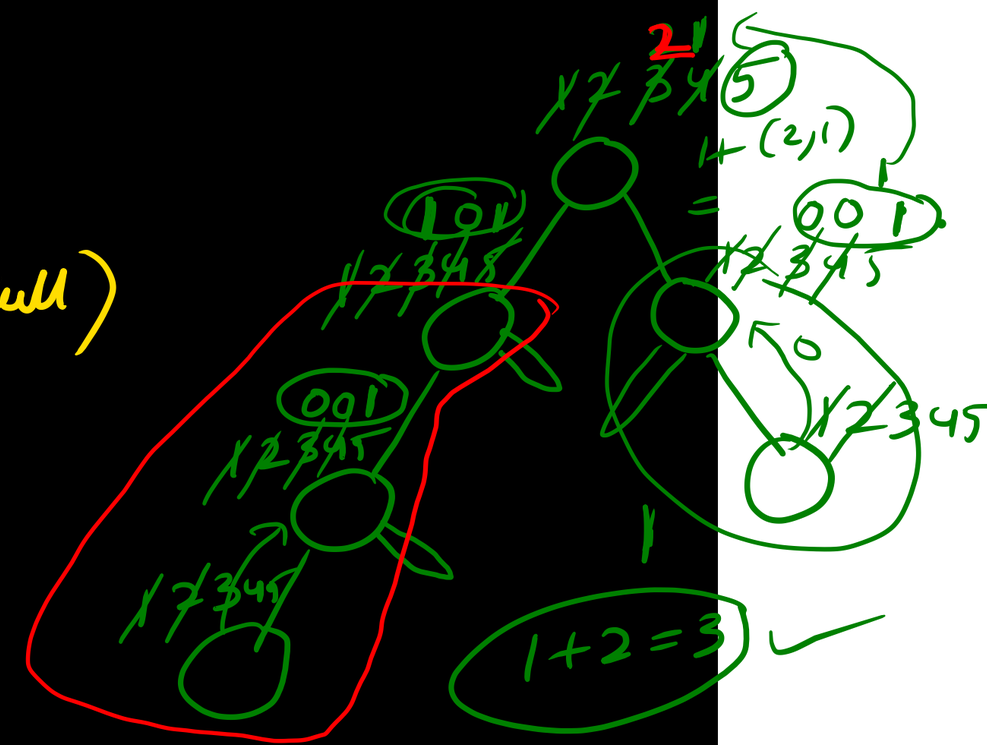
$H(T) = 1 + \max(H(LST), H(RST))$

$= 1 + \max(2, 1)$

$= 1 + 2 = 3.$

```
Height ( Struct node *t)
{
1) if (t == null) return 0;
2) if (t->left == null && t->right == null)
     return 0
   else {
3)   HL = Height (t->left);
4)   HR = Height (t->right);
5)   H = 1 + max (HL, HR);
     Return H;
   }
}
```

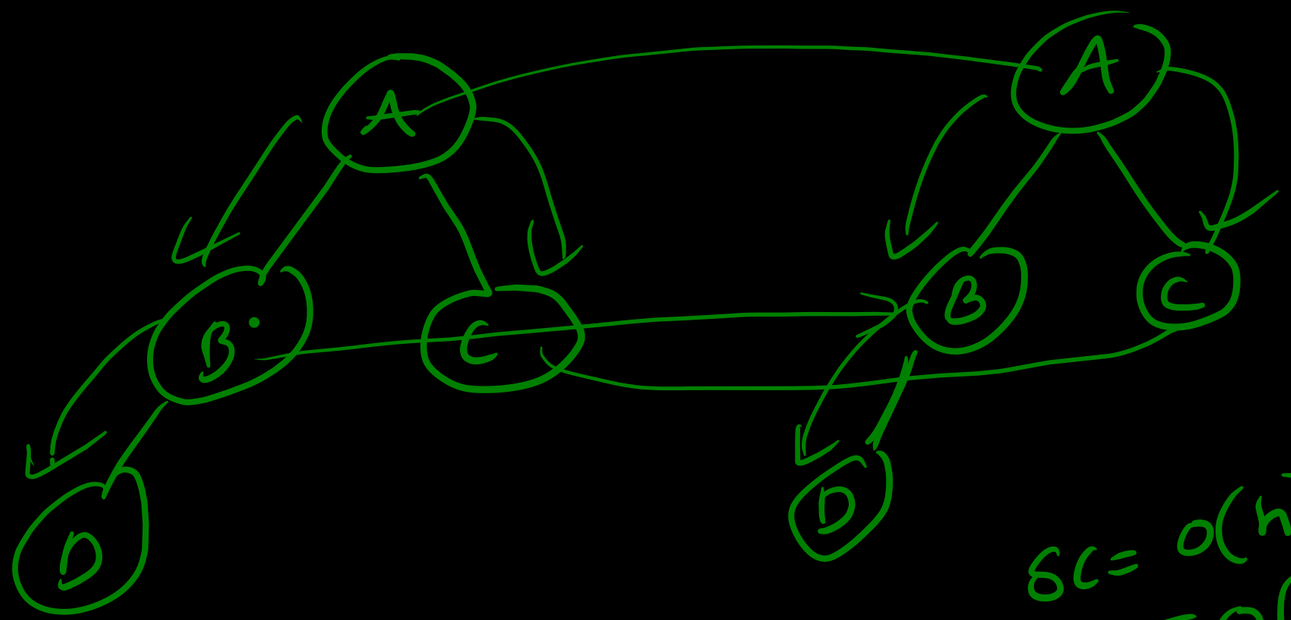




SC = O(h)
= O(n)

TC
= we are visiting all the nodes once = O(n)

Write an algo to check if two BT are identical.



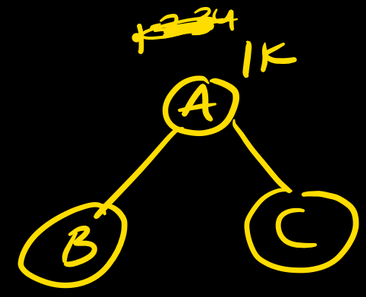
Visiting all the nodes → TC = O(n)

In order
Pre order
post order
} → O(n).

O(3*n)
O(3n)
= O(n).

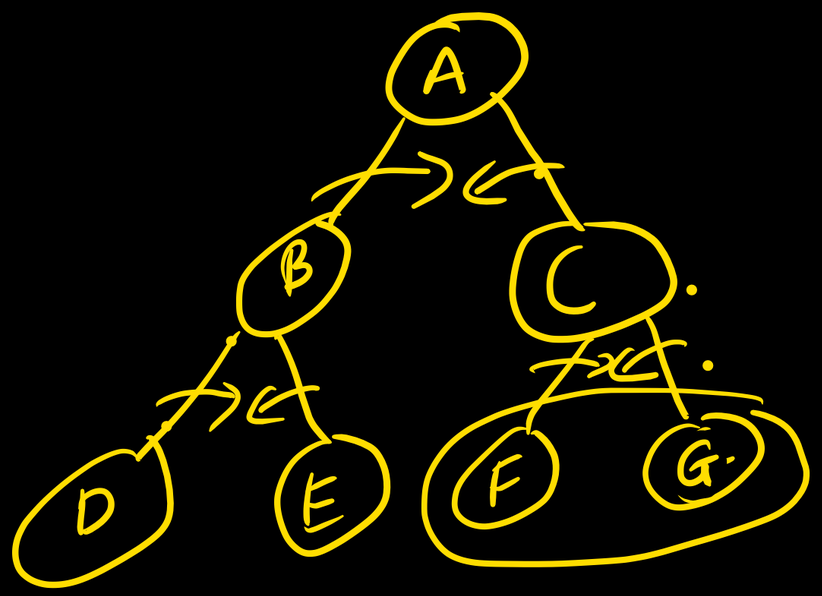
SC = O(n)
= O(n)

```
identical trees ( struct node *a, struct node *b)
{
① if (a == null && b == null)
      return true;
② if (a != null && b != null)
  {
    return (a->data == b->data)
    && identical (a->left, b->left)
    && identical (a->right, b->right)
  }
  return false.
}
```

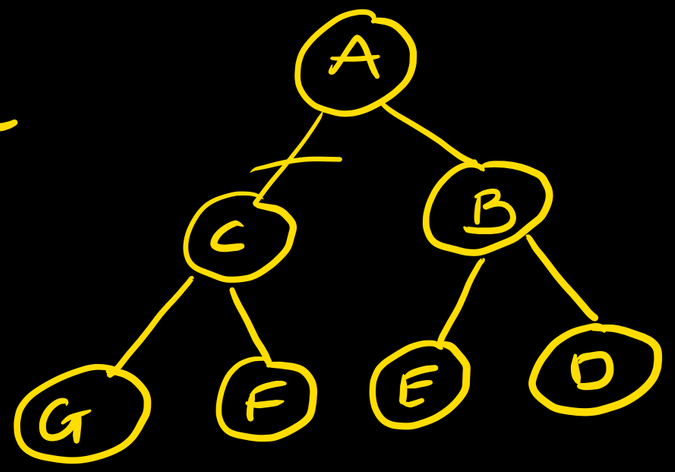


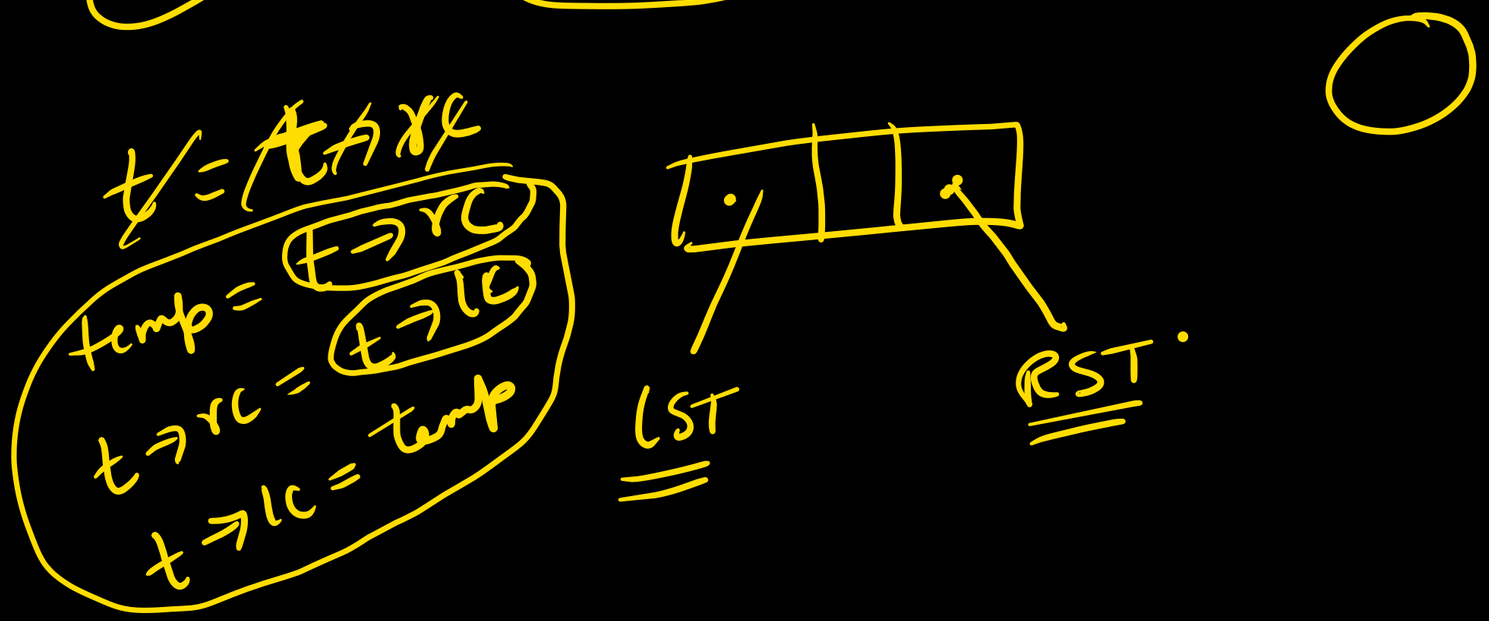
recursion tree in a single true. we have two tree.

a = 1k			
b = 2k			

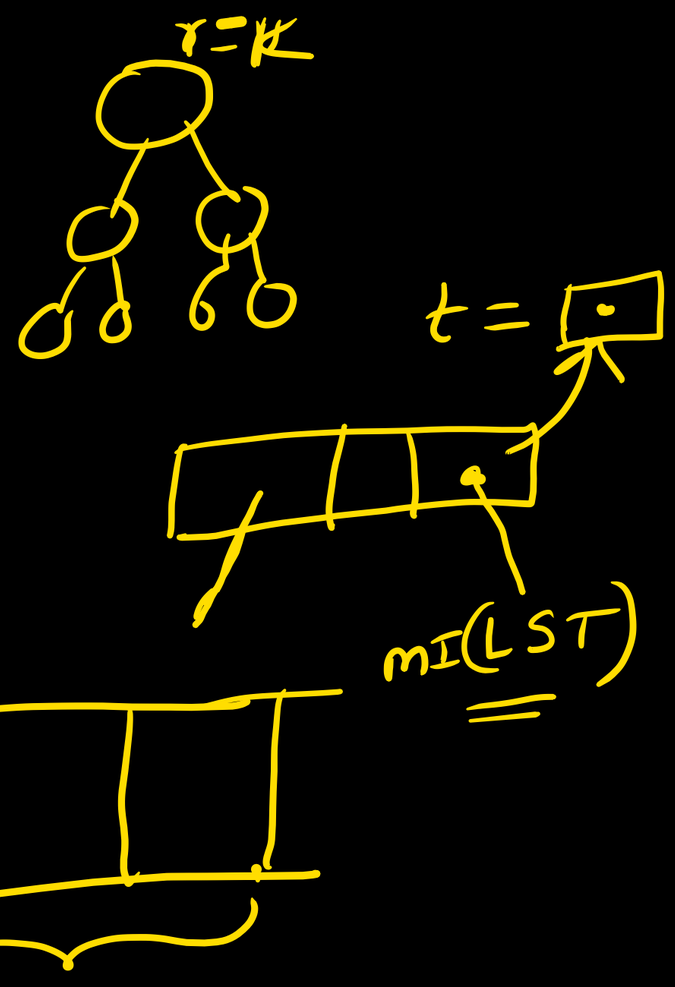


mirror image





```
mI(r)
{
  if (r== null) return(r);
  if ( r->lc ==null) && (r->rc ==null)
     return(r);
  else { t = r->rc;
       {
         r->rc = mI(r->lc)
         r->lc = mI(t)
          return(r)
       }
}
```



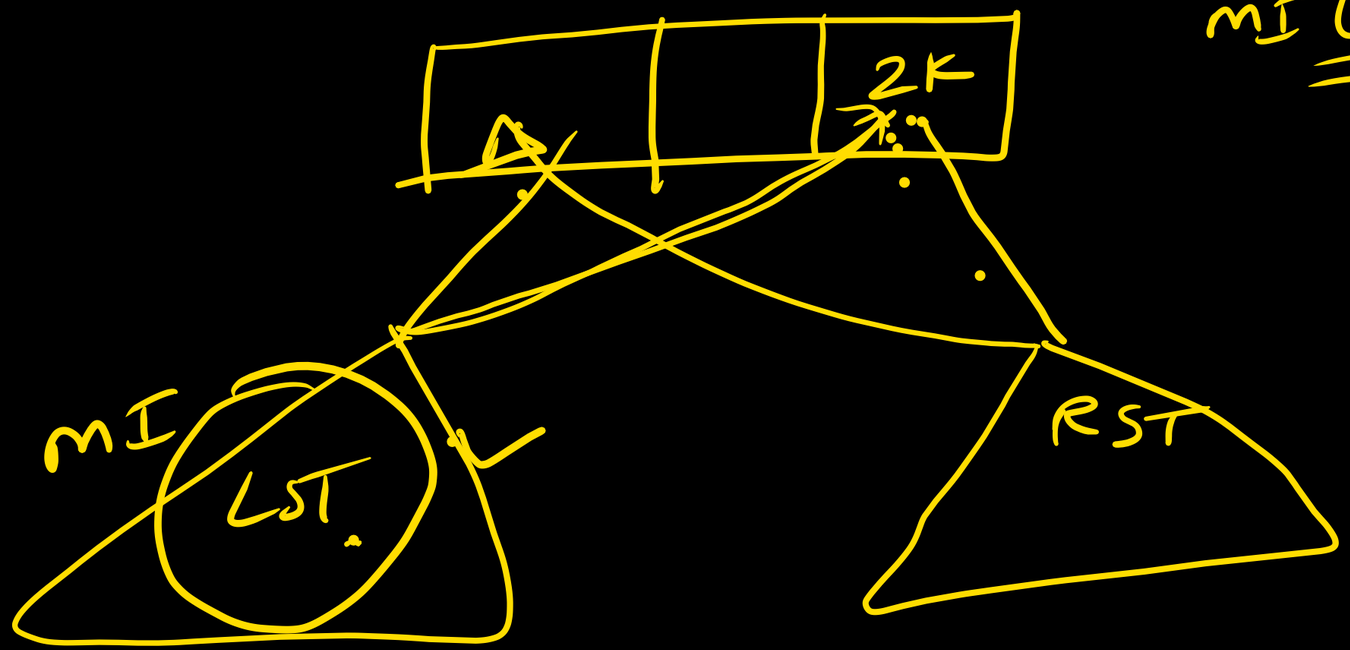
Whether two trees are mirror of each other.

(1) Take a tree and find its mirror image.

(2) Compare the mirror image with other tree.

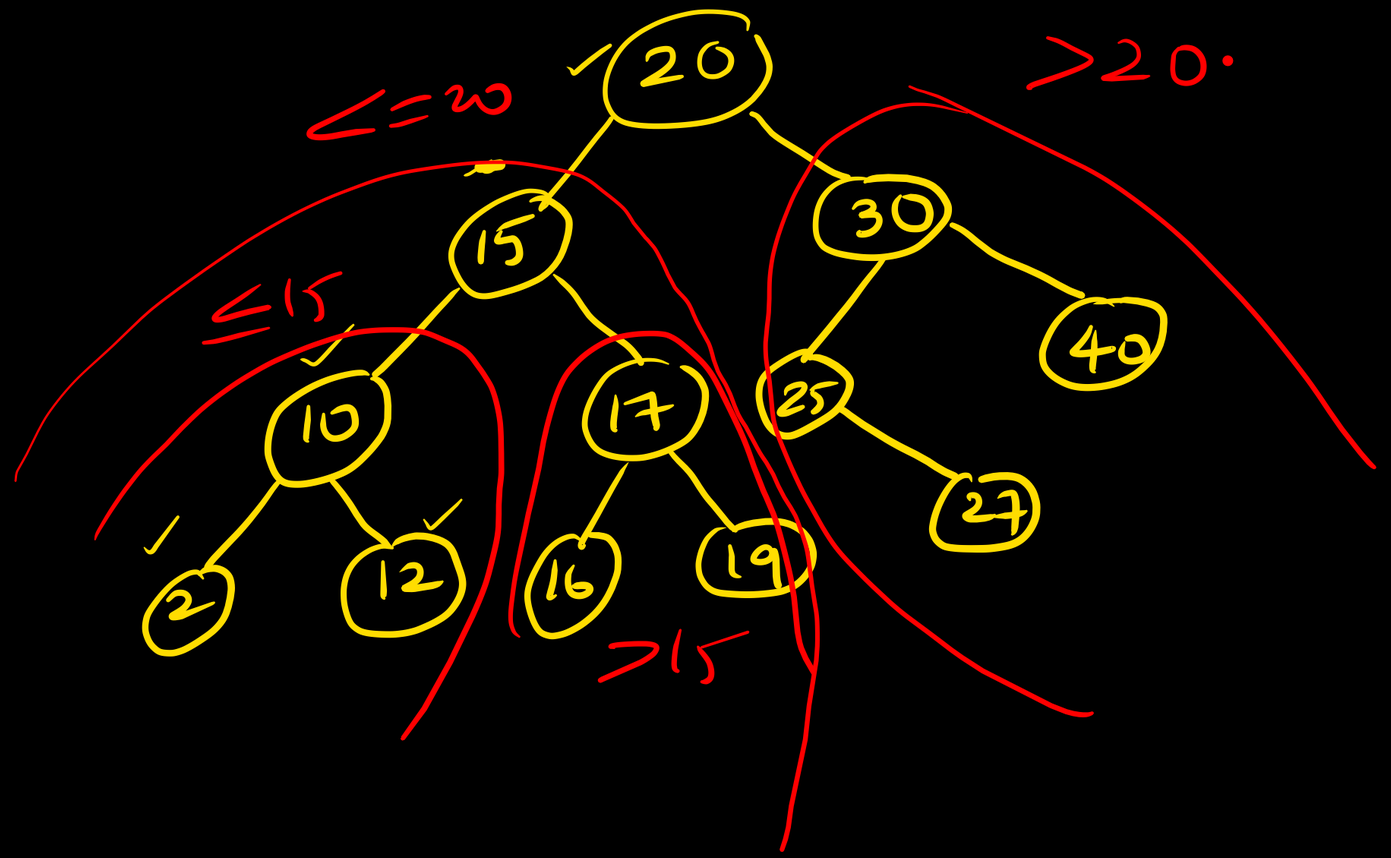
$t = \lceil 2K \rceil$.

$m_I(t)$.

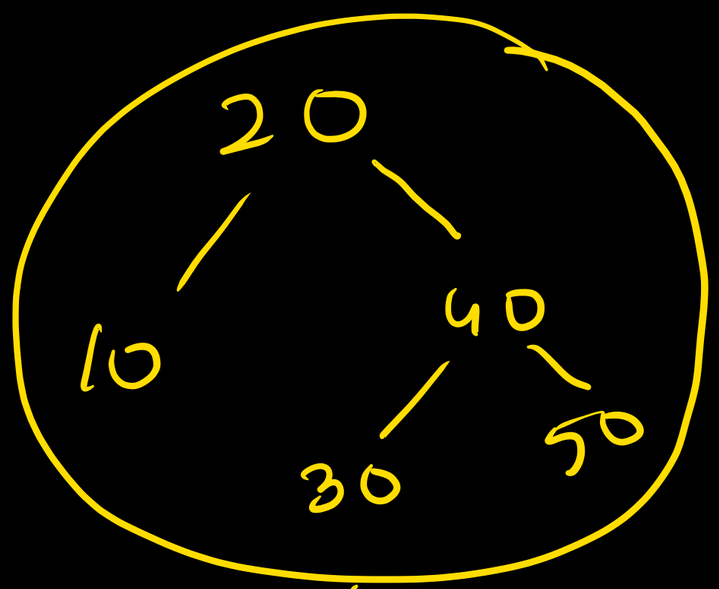


Binary search tree :-

A BST is a BT in which at every node, the number in LST are <= node and number in RST > node.

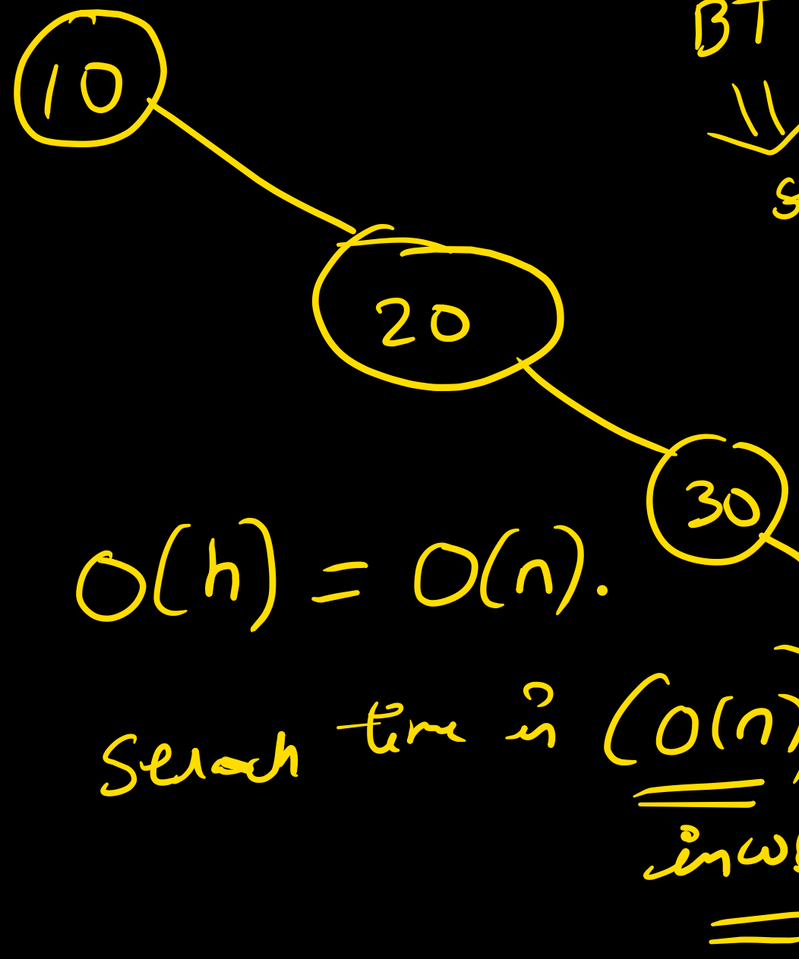


Searching BST.



$TC = \underbrace{O(\log n)}_{O(h)}$

Best, average case.



$O(h) = O(n)$.

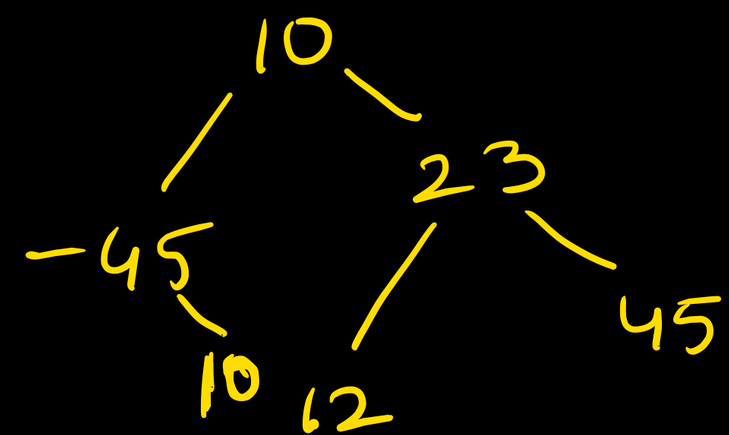
Search time in $O(n)$ in worst

BT ⟹ BST

Pre order

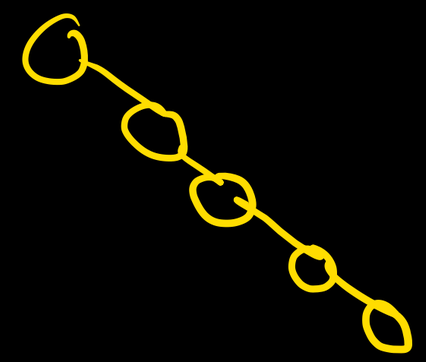
SAE

10, 23, 45, -45, 12, 11, 10



Construct a BT ✓.

Skewed.



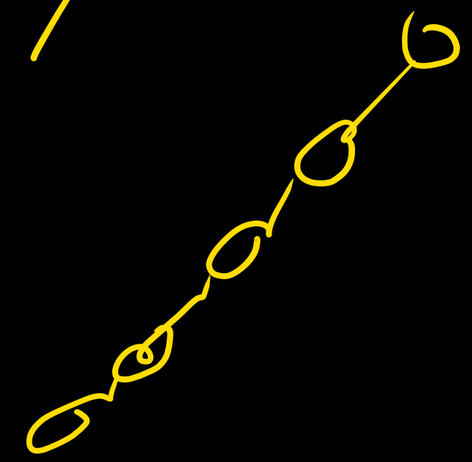
$n \times O(n)$.

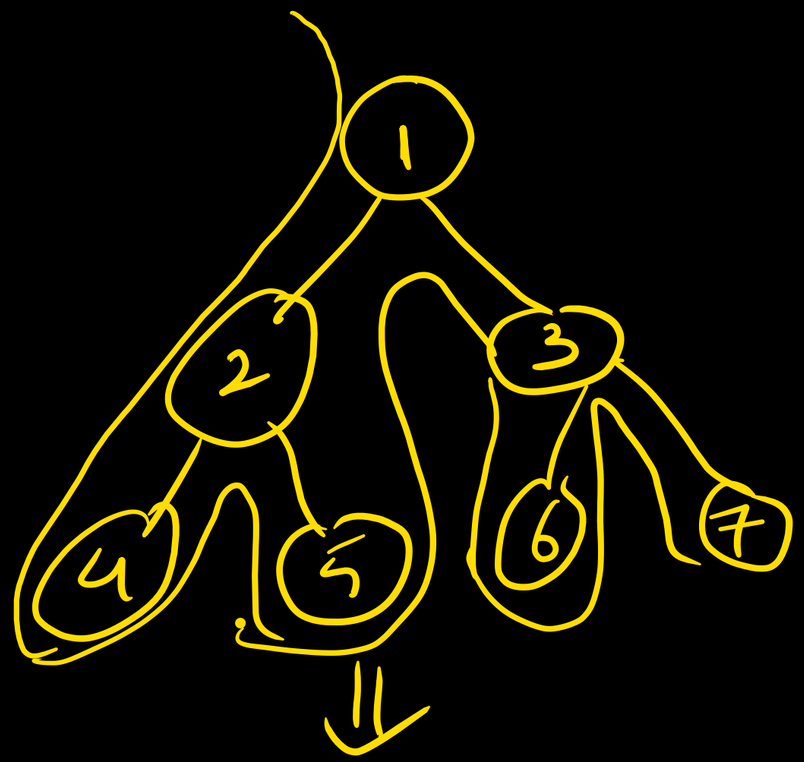
$WC = O(n^2)$ ✓

$BC = O(n \log n)$

$AC =$

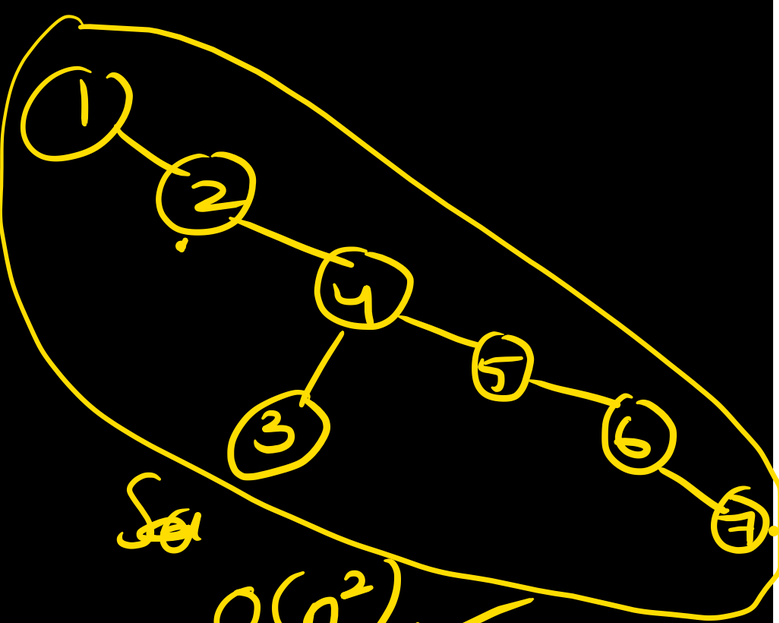
balance.





⇒

BST.



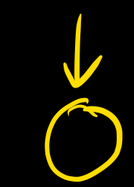
$O(n^2)$ ✓

Traversal → preorder

→ 1 2 4 5 3 6 7

Build a BST → $O(n^2)$

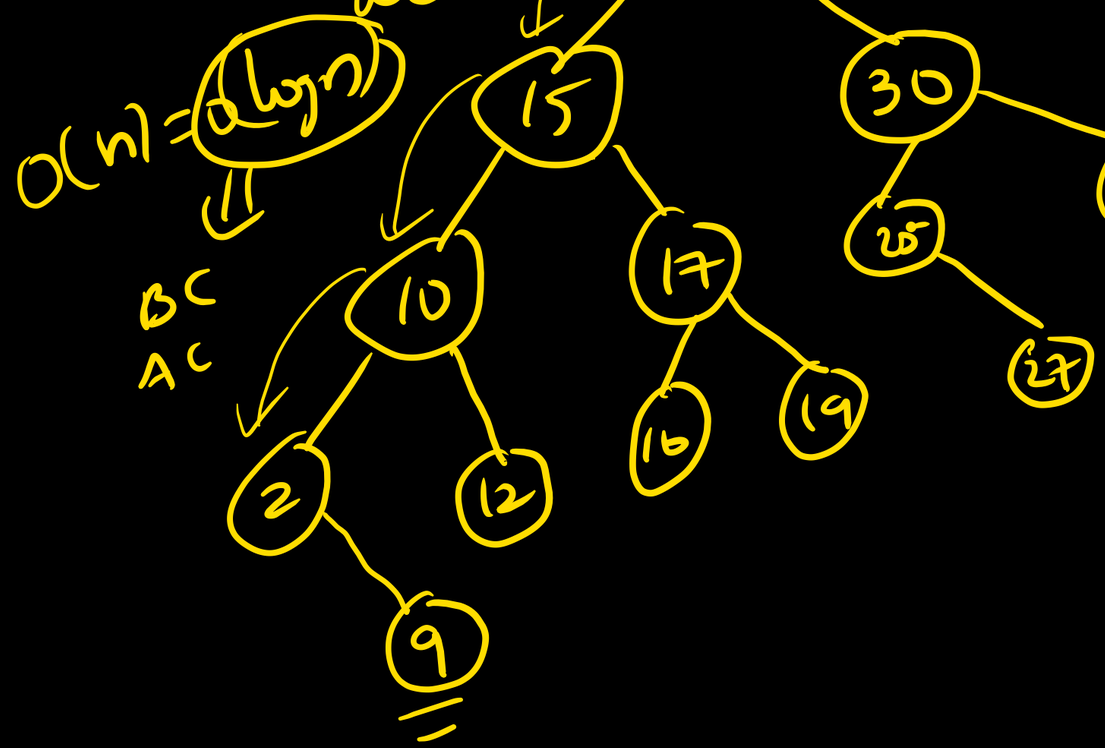
Insert a into a BST → AC $O(\log n)$, BC $O(n)$



$O(1)$.

$O(h) = O(n)$

WC



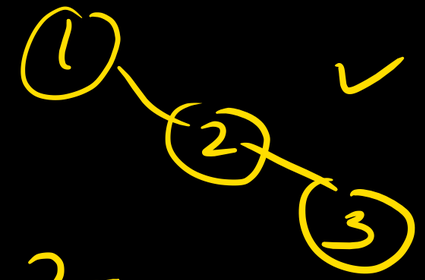
$O(n) = O(\log n)$

BC

AC

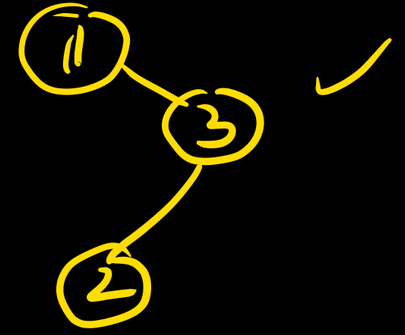
(9) ✓

1 2 3 — $O(1)$ – BC ✓, AC = $O(\log n)$ ✓, WC = $O(n^2)$.



✓

1 3 2



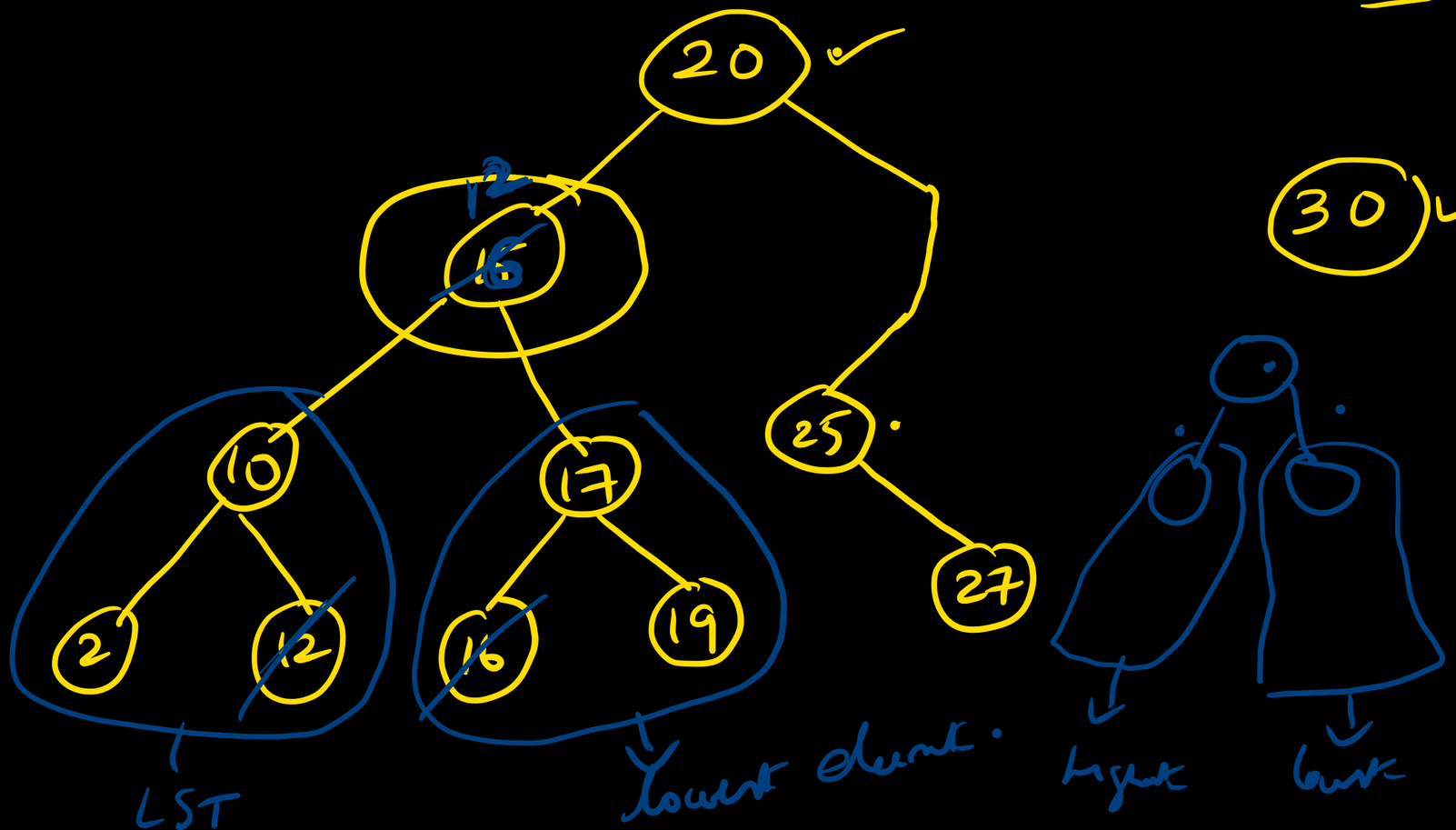
✓

3 2 1

✓

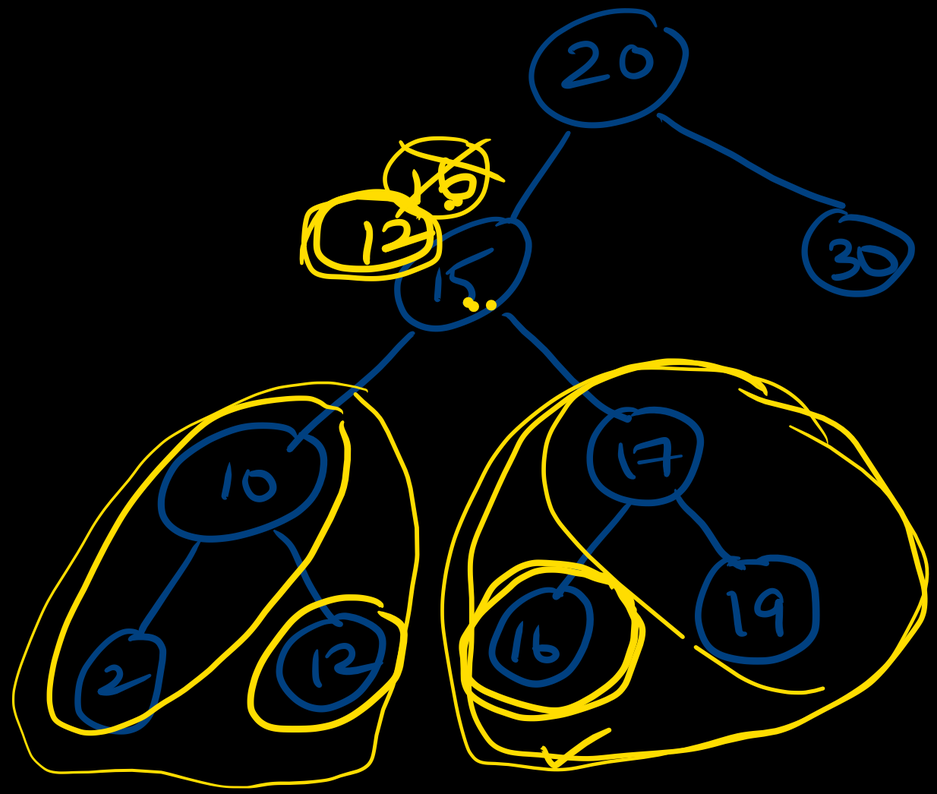
Deletion from a BST.

Delete 40

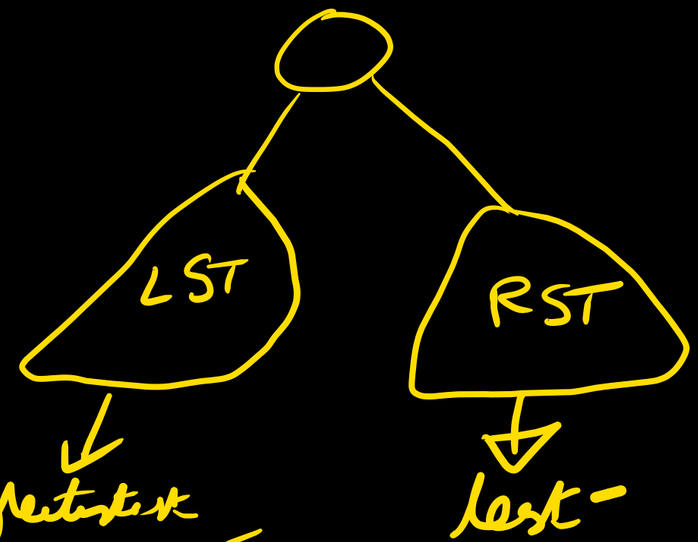


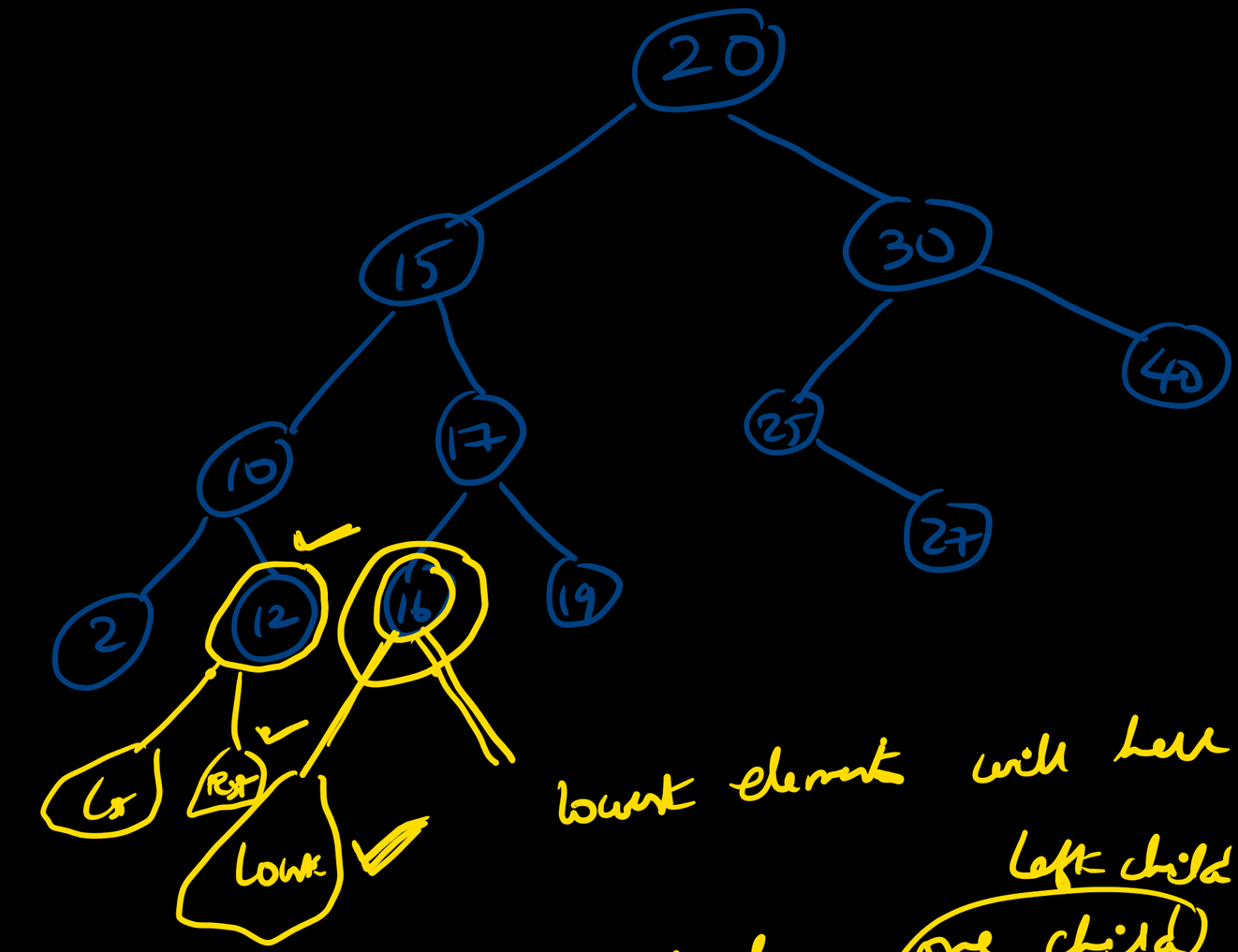
30 → 1 child.

Connect the child to grandparent.



what if these element have two children.





Zero
one
two children ✓

lowest element will have either 1 or 2 children

largest element will have one child (left child) or 0 children.

∴ BST → Inorder
→ Sorted

O(n) ✓

20
15
30
10
17
25
40
2
12
16
19
27

O(n)

O(n)

Inorder predecessor.

Inorder successor

2, 10, 12, 15, 16, 17 19 20 25 27

30, 40

BST deletion algo:

1) Find node x to be delete $\Rightarrow$ $O(n)$ time WC

if 0 children $\Rightarrow$ simply delete it

1 child $\Rightarrow$ connect it to grandparent

2 children $\Rightarrow$ delete node and replace by

Inorder predecessor $\Rightarrow O(n)$

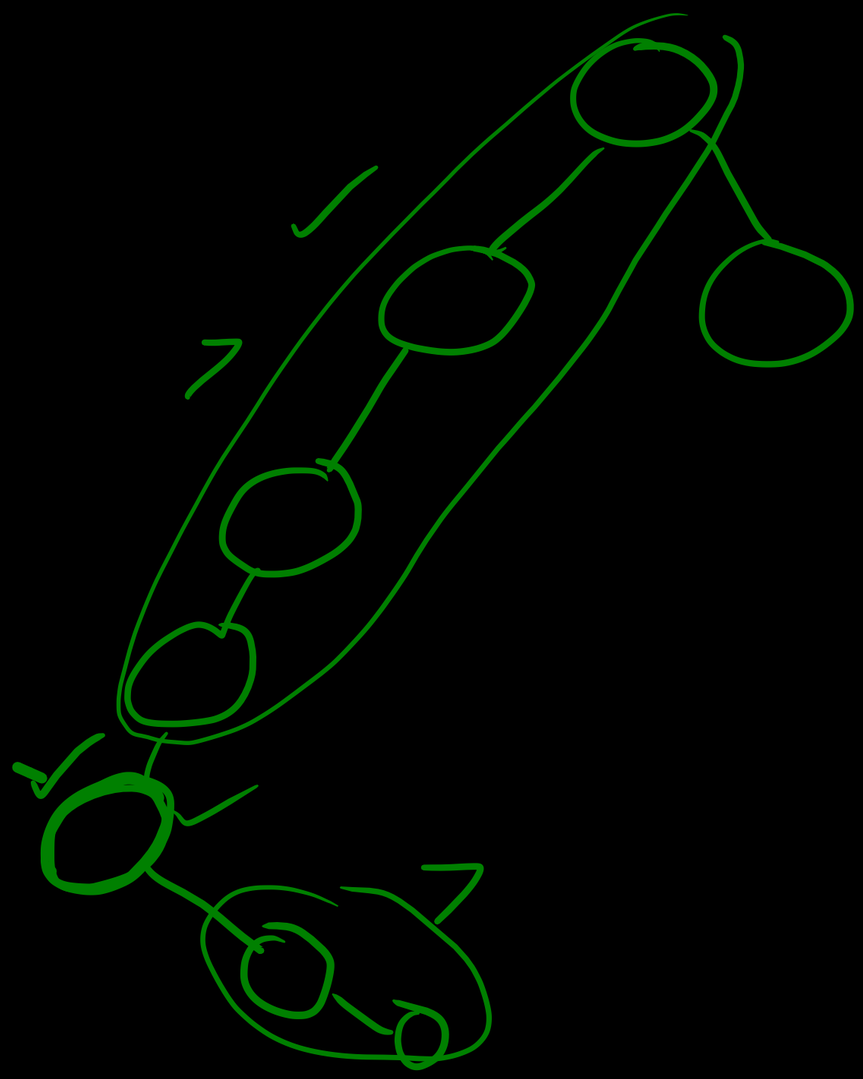
or

Inorder successor $\Rightarrow O(n)$

$O(n) + O(n)$

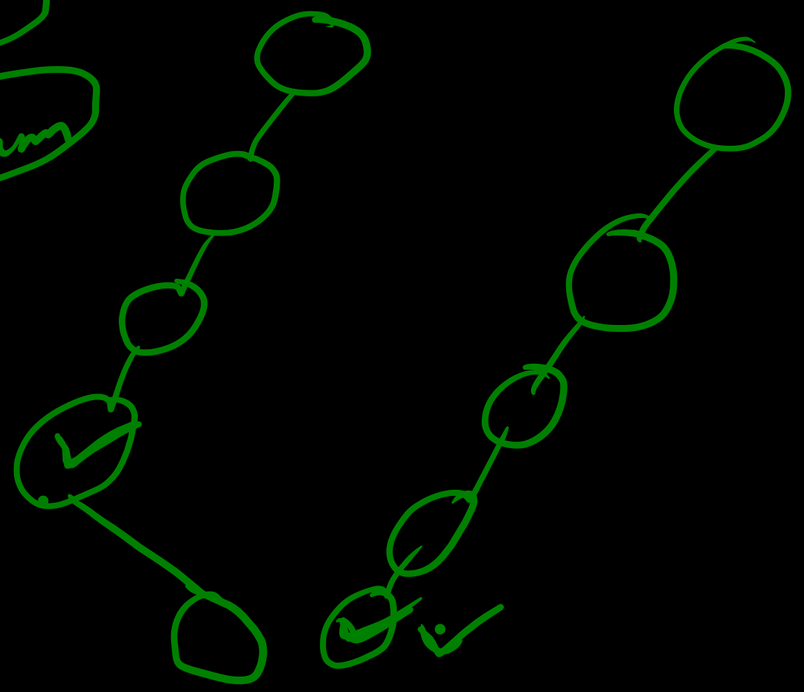
$= O(n)$ WC

find min BST



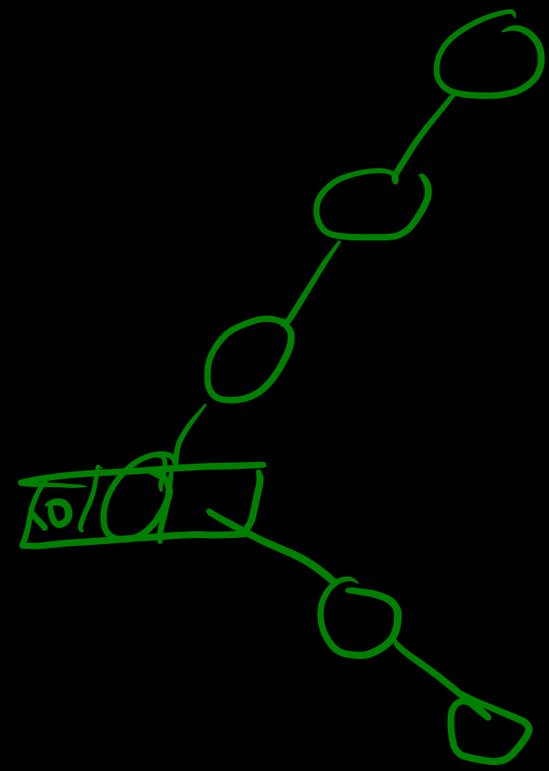
In order

I elem



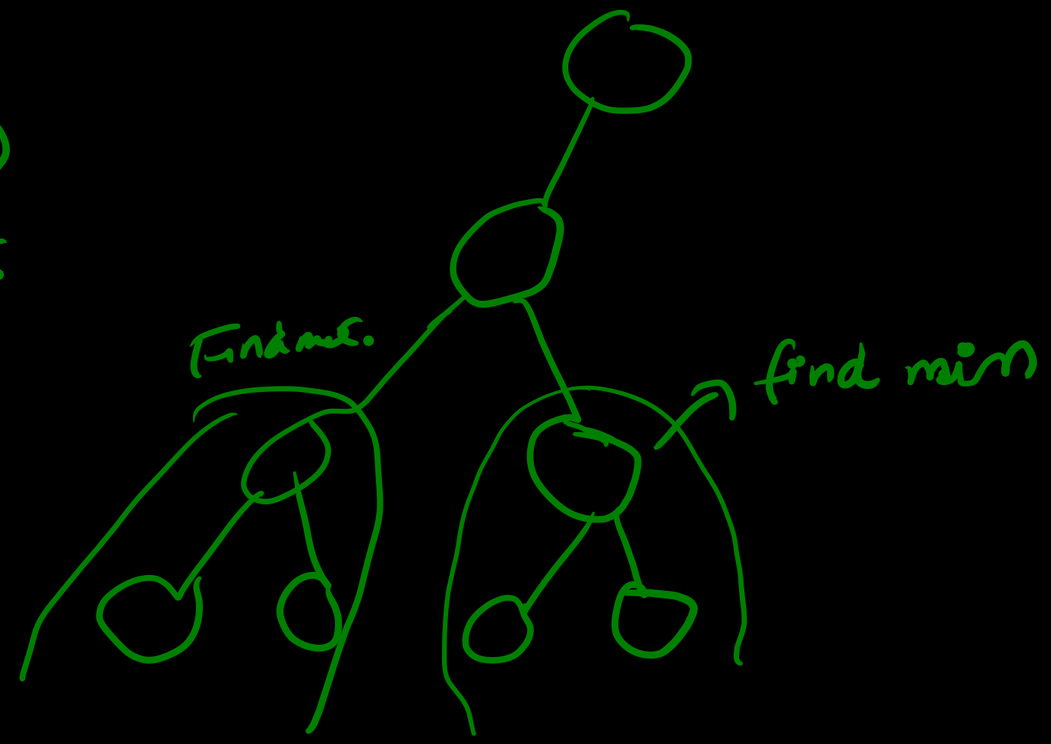
T(n) = O(n)

```
find min (struct node *t)
{
    while (t->left)
        t = t->left;
}
```



```
find_max (struct node*)
{
    while (t -> right)
        t = t -> right
}
```

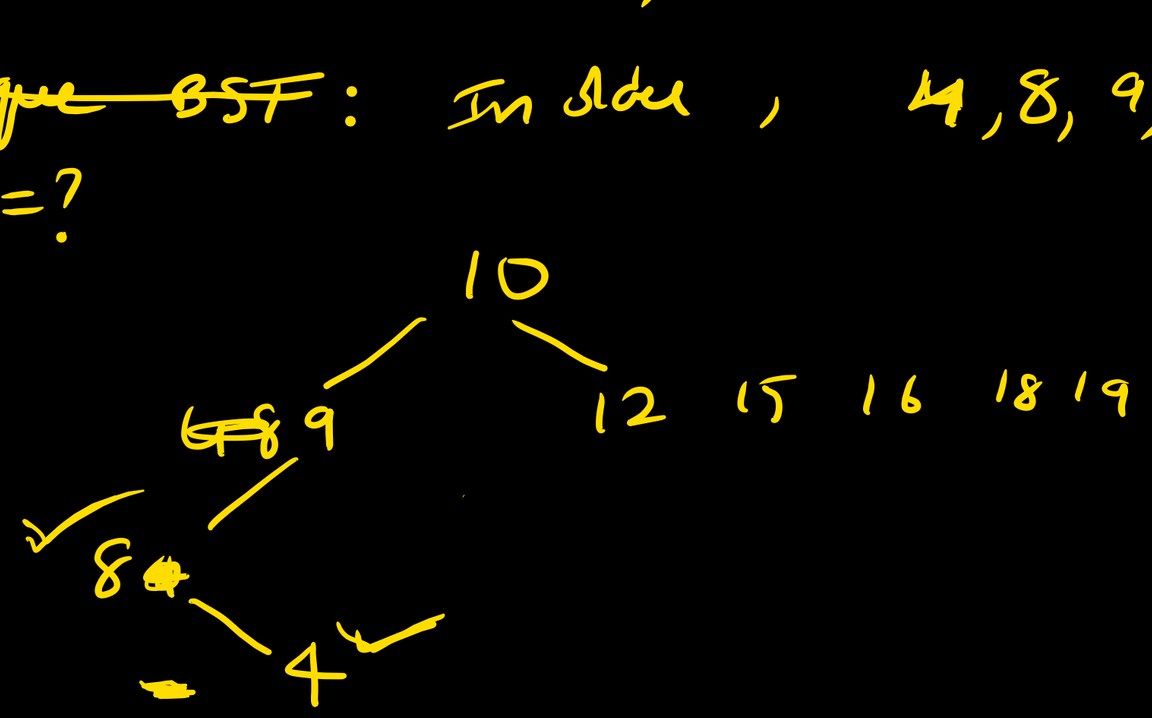
O(n)



Pre order of an BST is (10), 12, 9, 8, 4, 16, 15, 19, 18.

~~Draw the unique BST~~: In order, 4, 8, 9, (10), 12, 15, 16, 18, 19.

Post order = ?



Gate:

How many BST are possible with 4 distinct keys.

labeled tree

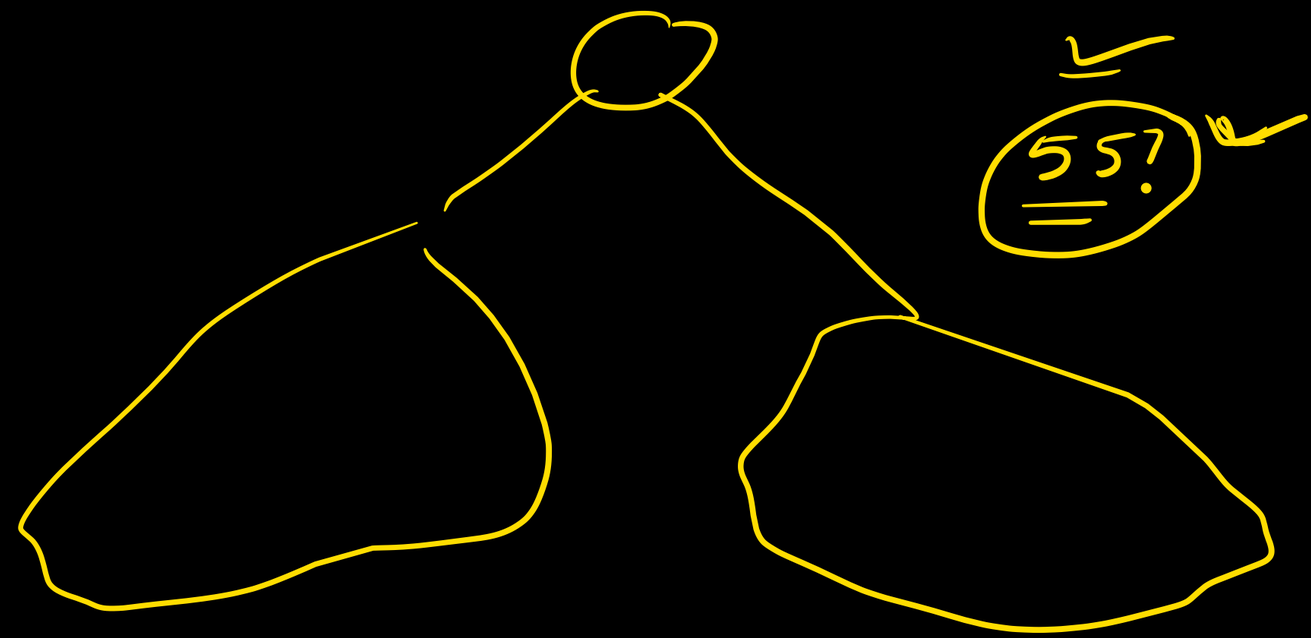
$$\frac{2nCn}{(n+1)} \times \cancel{n!}$$

3 nodes

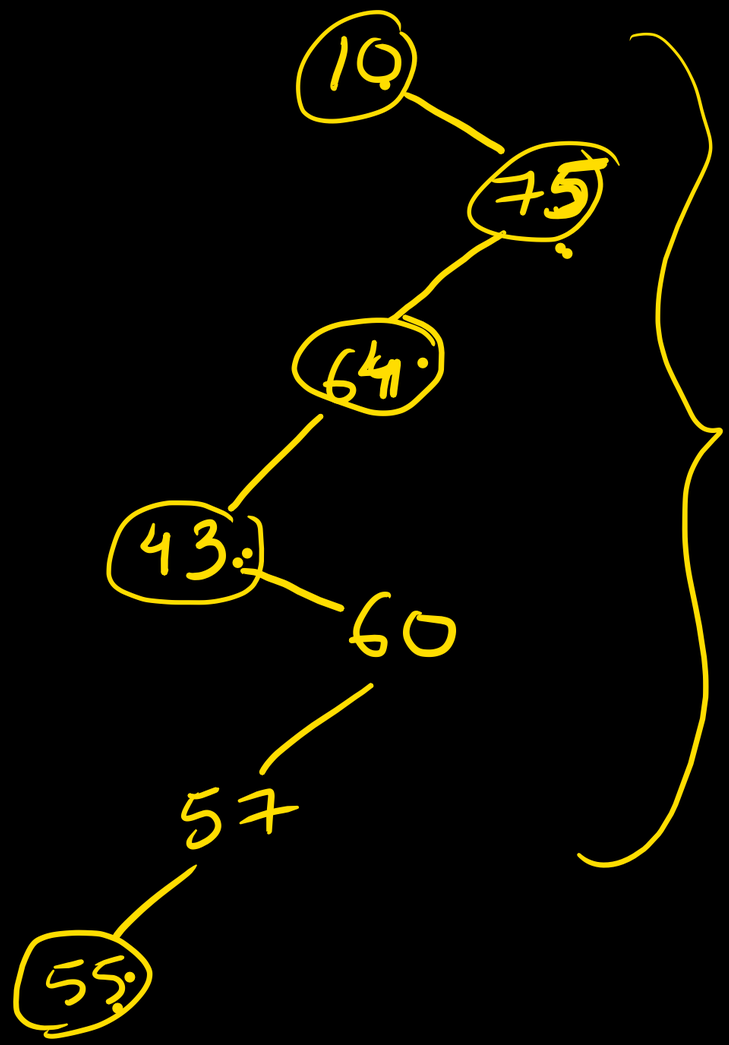
1 2 3

BST = 5

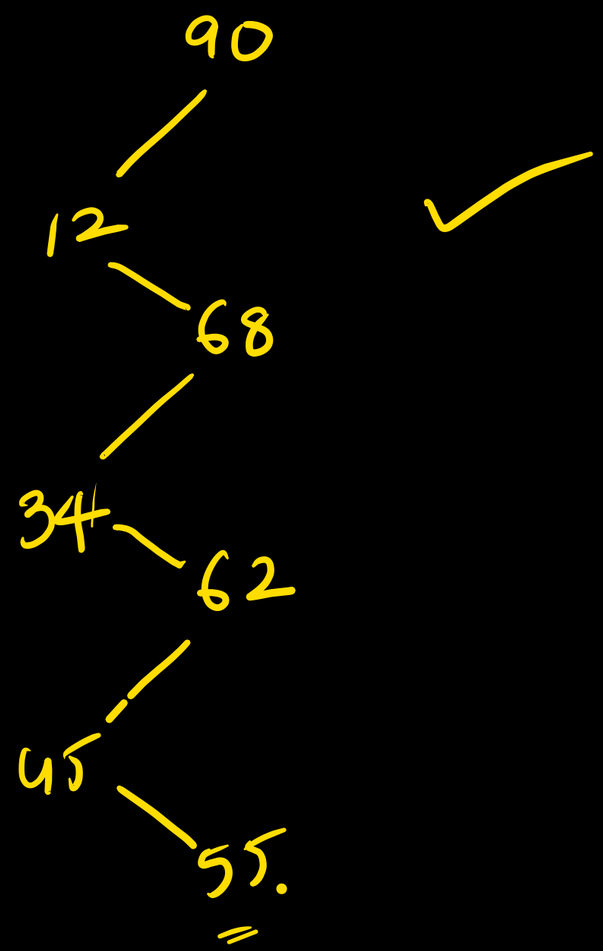
Ques: (1–100) numbers in a BST which of the following sequences cannot be the sequence of nodes visited in a search for 55.



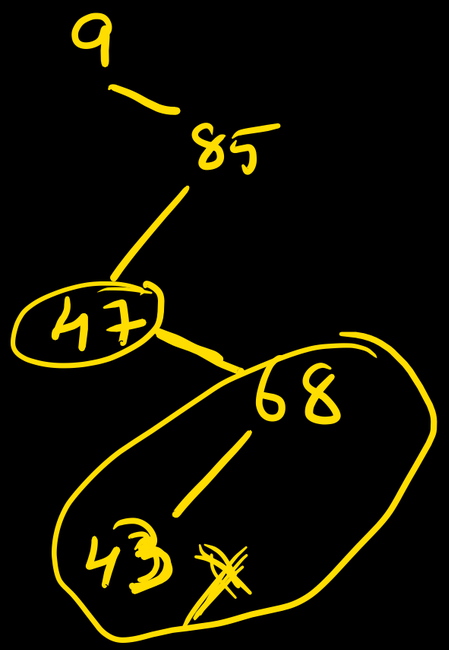
10 75 64 43 60 57 55



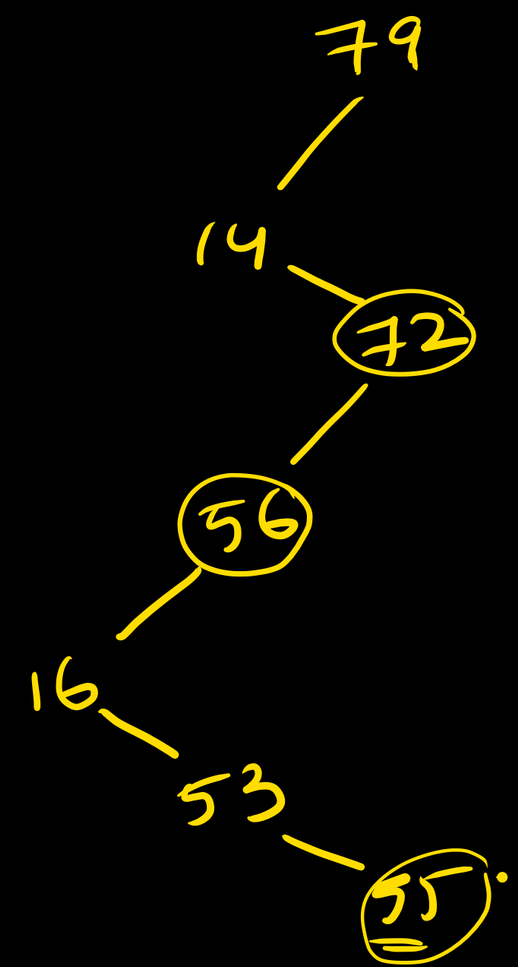
90 12 68 34 62 45 55



× 9, 85, 47, 68, 43, 57, 55



d) 79 14 72 56 16 53 55



Code:

Pork ← I : MBCAFHPYE

✓ Pre ← II : K AMCBYPFH

✓ In ← III : MABCKYFPH

Pre pork In.

MBCAFHPYK ✓

MABC K YFPH.