INSERT(a)

DELETE (a)

SEARCH (a)

INSERT (a) - $O(1)$

DELETE (a) - $O(n)$

SEARCH (a) - $O(n)$

LINKED LIST

INSERT(a)

DELETE (a)

SEARCH (a)

Q: WHAT IF $a \in [1 \ldots B]$

INSERT(a)

DELETE(a)

SEARCH(a)

Q: WHAT IF $a \in [1 \dots B]$

MAKE AN ARRAY $A[1 \dots B]$

INSERT(a)
{

INSERT (a)

DELETE (a)

SEARCH (a)

Q: WHAT IF $a \in [1 .... B]$

MAKE AN ARRAY $A[1....B]$

```
INSERT(a)              DELETE (a)
{   A[a] ← 1           {  A[a] ← 0
}                      }
```

```
SEARCH (d)
{   RETURN A[a]
}
```

INSERT(a) — $O(1)$

DELETE (a) — $O(1)$

SEARCH (a) — $O(1)$

Q: WHAT IF $a \in [1 \ldots. B]$

MAKE AN ARRAY $A[1 \ldots. B]$

```
INSERT(a)
{   A[a] ← 1
}
```

```
DELETE (a)
{   A[a] ← 0
}
```

```
SEARCH (d)
{   RETURN A[a]
}
```

INSERT (a)
DELETE (a)
SEARCH (a)

NO INSERT & DELETE. YOU ARE GIVEN n NUMBERS AND YOU JUST WANT TO DO SEARCHES.

INSERT(a)
DELETE(a)
SEARCH(a)

NO INSERT & DELETE. YOU ARE GIVEN n NUMBERS AND YOU JUST WANT TO DO SEARCHES.

- SORT THE n NUMBERS
- DO BINARY SEARCH FOR ALL SEARCHES.

INSERT(a)
DELETE(a)
SEARCH(a)

NO INSERT & DELETE. YOU ARE GIVEN n NUMBERS AND YOU JUST WANT TO DO SEARCHES.

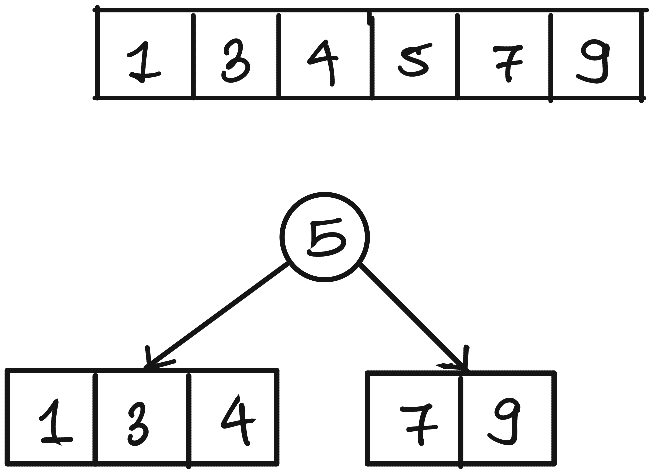
- SORT THE n NUMBERS $n\log n$
- DO BINARY SEARCH FOR ALL SEARCHES.

$\downarrow$ $\log n$ TIME PER SEARCH.

BINARY SEARCH CAN BE NICELY VISUALIZED USING A TREE

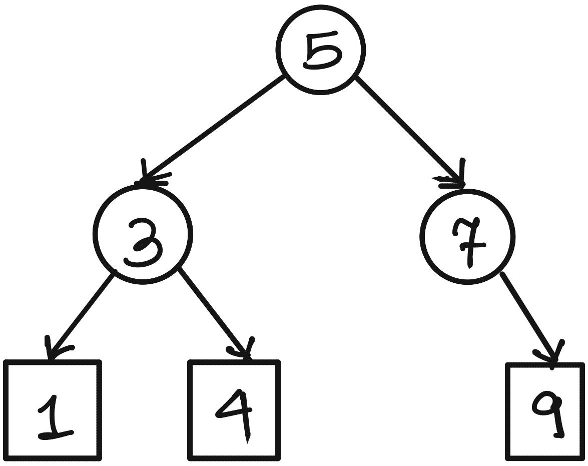
1	3	4	5	7	9

BINARY SEARCH CAN BE NICELY VISUALIZES USING A TREE



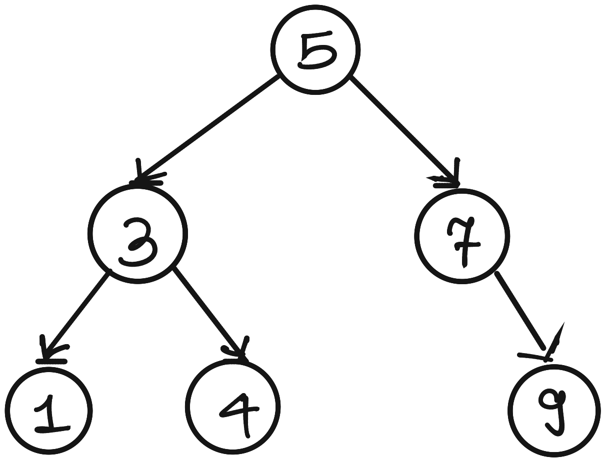
BINARY SEARCH CAN BE NICELY VISUALIZES USING A TREE

1	3	4	5	7	9



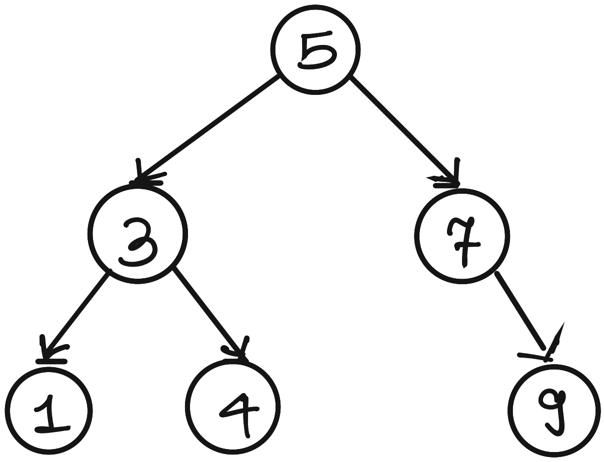
BINARY SEARCH CAN BE NICELY VISUALIZES USING A TREE

1	3	4	5	7	9



BINARY SEARCH CAN BE NICELY VISUALIZES USING A TREE

1	3	4	5	7	9

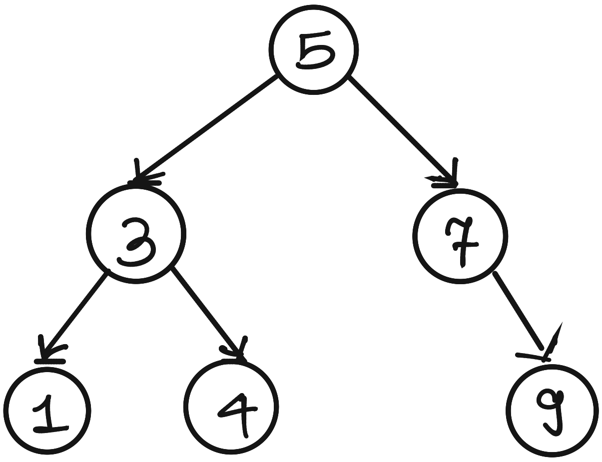


BINARY SEARCH TREE. (BST)

Q: WHAT IS THE HEIGHT OF THIS BST?

BINARY SEARCH CAN BE NICELY VISUALIZES USING A TREE

1	3	4	5	7	9



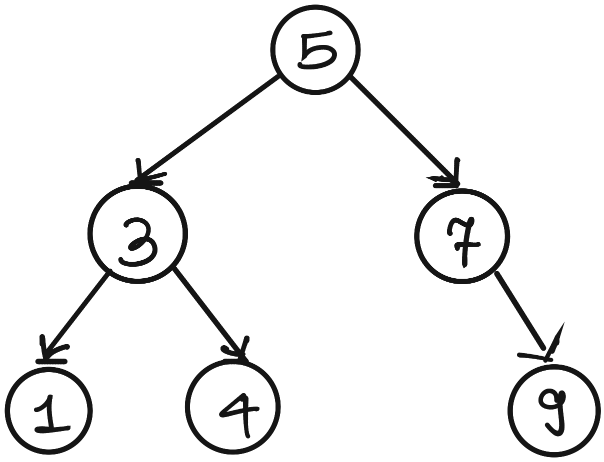
BINARY SEARCH TREE. (BST)

Q: WHAT IS THE HEIGHT OF THIS BST ?

A: $O(\log n)$

BINARY SEARCH CAN BE NICELY VISUALIZES USING A TREE

1	3	4	5	7	9



BINARY SEARCH TREE. (BST)

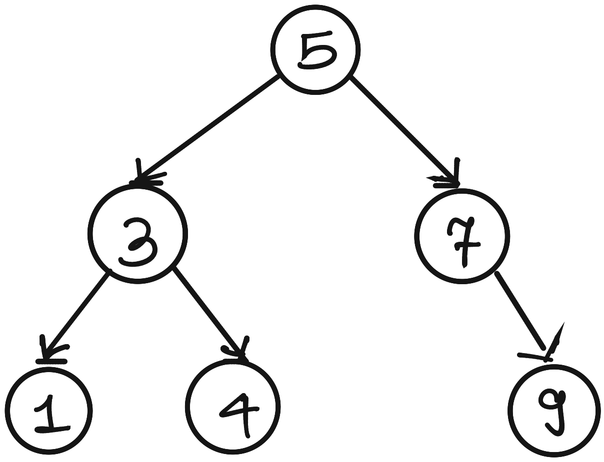
Q: WHAT IS THE HEIGHT OF THIS BST ?

A: $O(\log n)$

THIS TREE IS CALLED BALANCED BECAUSE THE HEIGHT OF LEFT SUBTREE OF EACH NODE $\approx$ HEIGHT OF ITS RIGHT SUBTREE

BINARY SEARCH CAN BE NICELY VISUALIZES USING A TREE

1	3	4	5	7	9



BINARY SEARCH TREE. (BST)

Q: WHAT IS THE HEIGHT OF THIS BST ?

A: $O(\log n)$

THIS TREE IS CALLED BALANCED BECAUSE THE HEIGHT OF LEFT SUBTREE OF EACH NODE ≈ HEIGHT OF ITS RIGHT SUBTREE

⇒ HEIGHT OF BALANCED BST = $O(\log n)$.

INSERT(a)

DELETE(a)

SEARCH(a)

IMPLEMENT A BINARY SEARCH TREE.

INSERT(a)

DELETE(a)

SEARCH(a)

IMPLEMENT A BINARY SEARCH TREE.

Def

<u>Def^n</u>: A BST IS A

(a) BINARY TREE (EACH NODE HAS AT MOST ONE LEFT & RIGHT CHILD)

(b) FOR EACH INTERNAL NODE v,

INSERT(a)

DELETE(a)

SEARCH(a)

IMPLEMENT A BINARY SEARCH TREE.

Def^n: A BST IS A

(a) BINARY TREE (EACH NODE HAS AT MOST ONE LEFT & RIGHT CHILD)

(b) FOR EACH INTERNAL NODE v,

v.value > VALUE OF ALL ELEMENTS IN LEFT SUBTREE OF v

&

v.value < VALUE OF ALL ELEMENTS IN RIGHT SUBTREE OF v

INSERT(a)

DELETE(a)

SEARCH(a)

IMPLEMENT A BINARY SEARCH TREE.

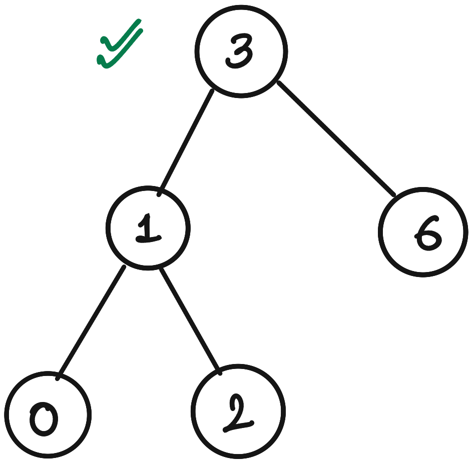
Def^n: A BST IS A

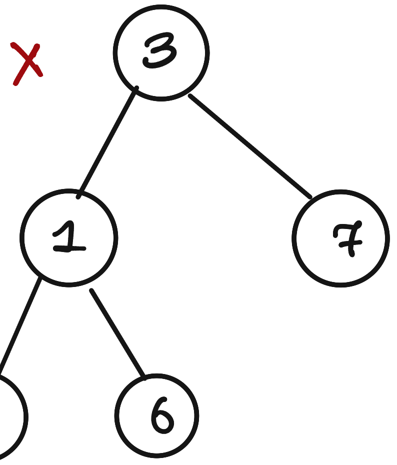
(a) BINARY TREE (EACH NODE HAS AT MOST ONE LEFT & RIGHT CHILD)

(b) FOR EACH INTERNAL NODE v, v.value > VALUE OF ALL ELEMENTS IN LEFT SUBTREE OF v

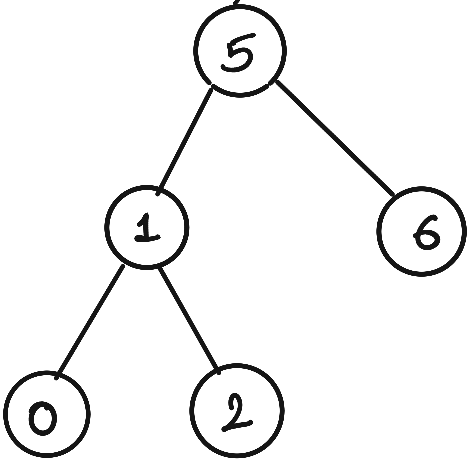
&

v.value < VALUE OF ALL ELEMENTS IN RIGHT SUBTREE OF v

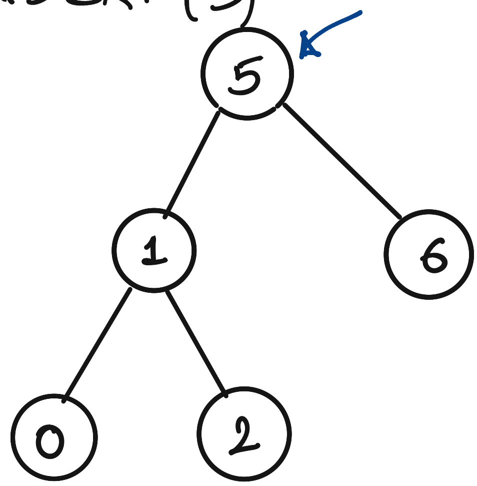




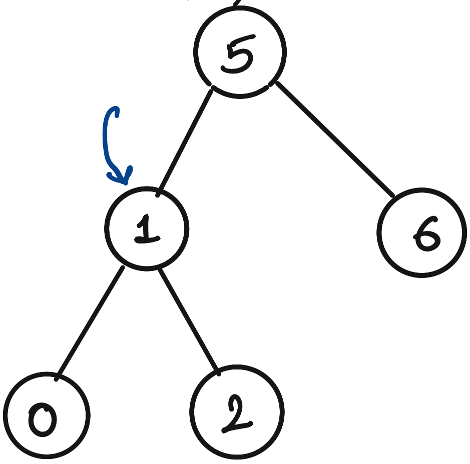
INSERT (3)



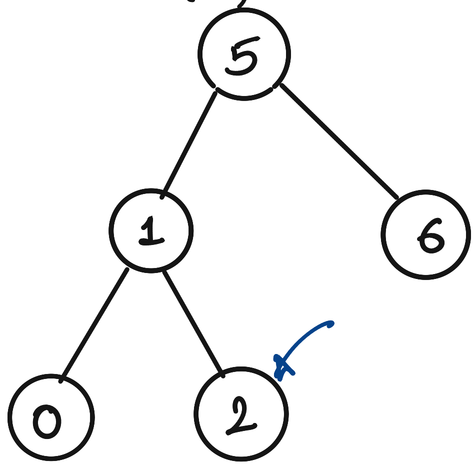
INSERT (3)



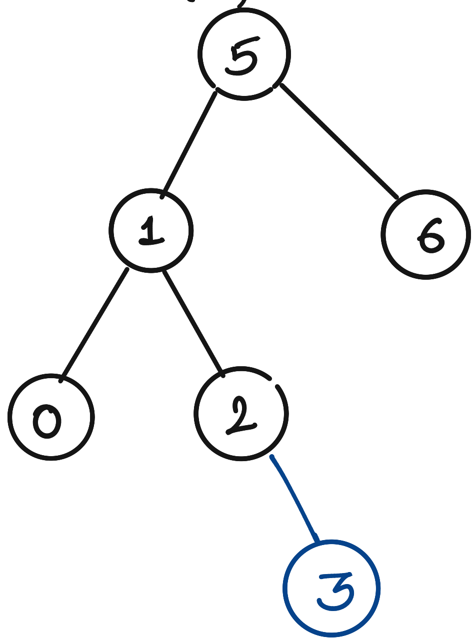
INSERT (3)



INSERT (3)



INSERT (3)



```
INSERT( root, a)
{   v ← A NEW NODE WITH v.value ← a,
        v.left ← Null & v.right ← NULL.

    If ( root = NULL)
    {   root ← v ;
        RETURN ;
    }
```

```
INSERT( root, a)
{   v ← A NEW NODE WITH v.value ← a,
        v.left ← Null & v.right ← NULL.

    If ( root = NULL)
    {   root ← v ;
    }   RETURN ;
    u ← root
    WHILE (TRUE)
    {   IF( v.value < u.value & u.left = NULL)
```

```
INSERT( root, a)
{   v ← A NEW NODE WITH v.value ← a,
        v.left ← Null & v.right ← NULL.

    If ( root = NULL)
    {   root ← v;
    }   RETURN;

    u ← root
    WHILE (TRUE)
    {   IF( v.value < u.value & u.left = NULL)
        {   u.left ← v;
        }   break;
        ELSE
```

```
INSERT( root, a)
{   v ← A NEW NODE WITH v.value ← a,
        v.left ← Null & v.right ← NULL.

    If ( root = NULL)
    {   root ← v;
    }   RETURN;
    u ← root
    WHILE (TRUE)
    {   IF( v.value < u.value & u.left = NULL)
        {  u.left ← v;
        }   break;
        ELSE   u ← u.left;
```

```
INSERT( root, a)
{   v ← A NEW NODE WITH v.value ← a,
        v.left ← Null & v.right ← NULL.

    If ( root = NULL)
    {   root ← v ;
    }   RETURN ;
    u ← root
    WHILE (TRUE)
    {   IF( v.value < u.value & u.left = Null)
        {   u.left ← v ;
        }   break ;
        ELSE   u ← u.left ;

        If ( v.value > u.value & u.right = Null)
        {   u.right ← v ;
            break ;
        }
        ELSE   u ← u.right
    }
}
```

```
INSERT( root, a)
{   v ← A NEW NODE WITH v.value ← a,
          v.left ← NULL & v.right ← NULL.

      If ( root = NULL)
      {    root ← v ;
      }    RETURN ;
      u ← root
      WHILE (TRUE)
      {    IF( v.value < u.value & u.left = NULL)
           {  u.left ← v ;
           }   break ;
           ELSE   u ← u.left ;

           If ( v.value > u.value & u.right = NULL)
           {   u.right ← v ;
                break ;
           }
           ELSE   u ← u.right
      }
}
```

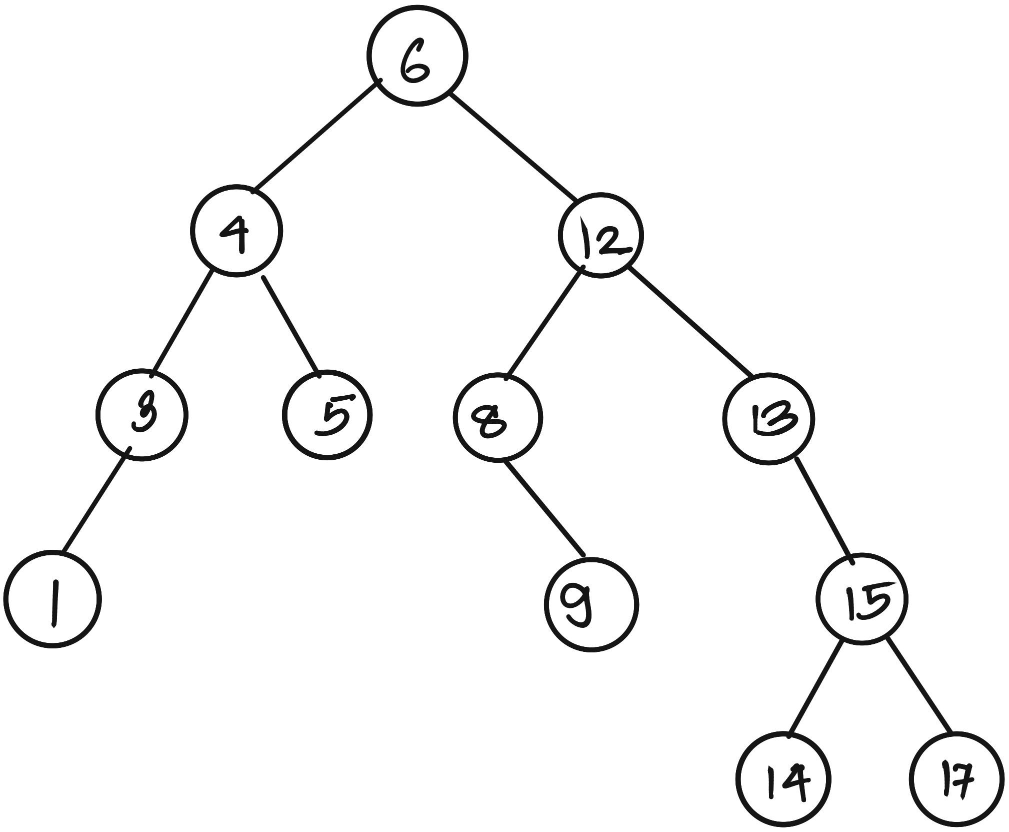
RUNNING TIME :

```
INSERT( root, a)
{   v ← A NEW NODE WITH v.value ← a,
            v.left ← Null & v.right ← NULL.

    If ( root = NULL)
    {   root ← v;
    }   RETURN;
    u ← root
    WHILE (TRUE)
    {   IF( v.value < u.value & u.left = Null)
        {   u.left ← v;
        }   break;
        ELSE   u ← u.left;

        If ( v.value > u.value & u.right = Null)
        {   u.right ← v;
            break;
        }
        ELSE   u ← u.right
    }
}
```

RUNNING TIME : O(h)

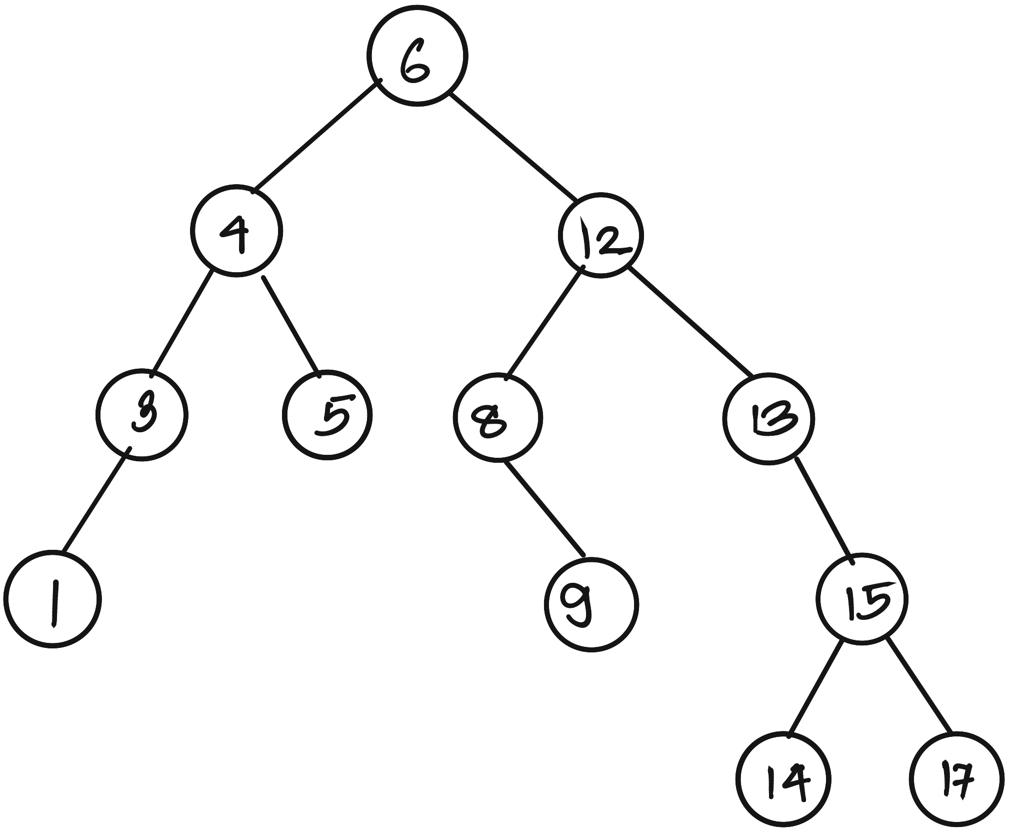
h : HEIGHT OF TREE.

DELETE



DELETE 9

DELETE

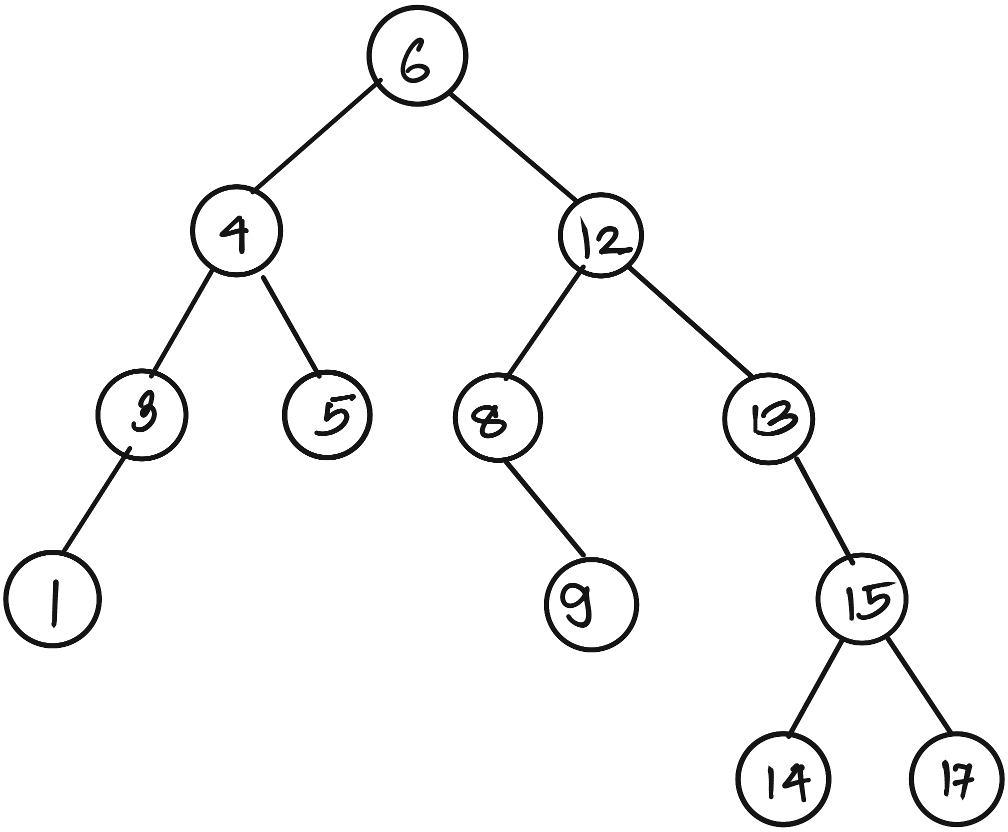


DELETE 9

SEARCH NODE 9.

NODE 9 IS A LEAF

DELETE



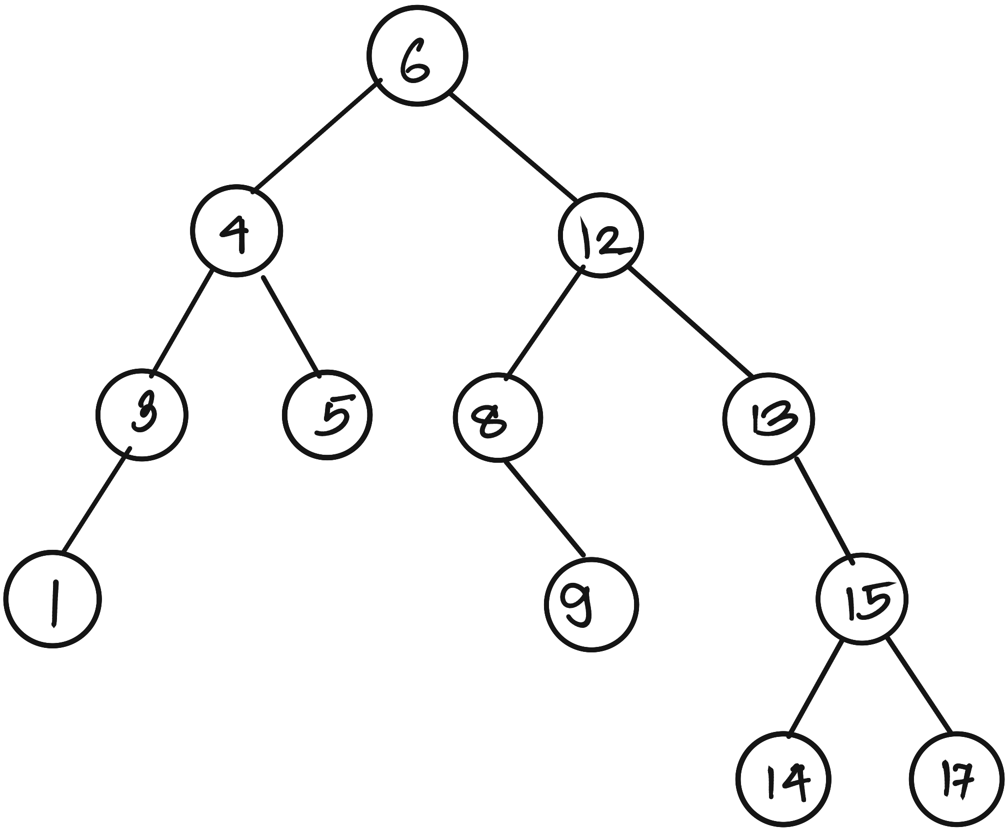
DELETE 9

SEARCH NODE 9.

NODE 9 IS A LEAF

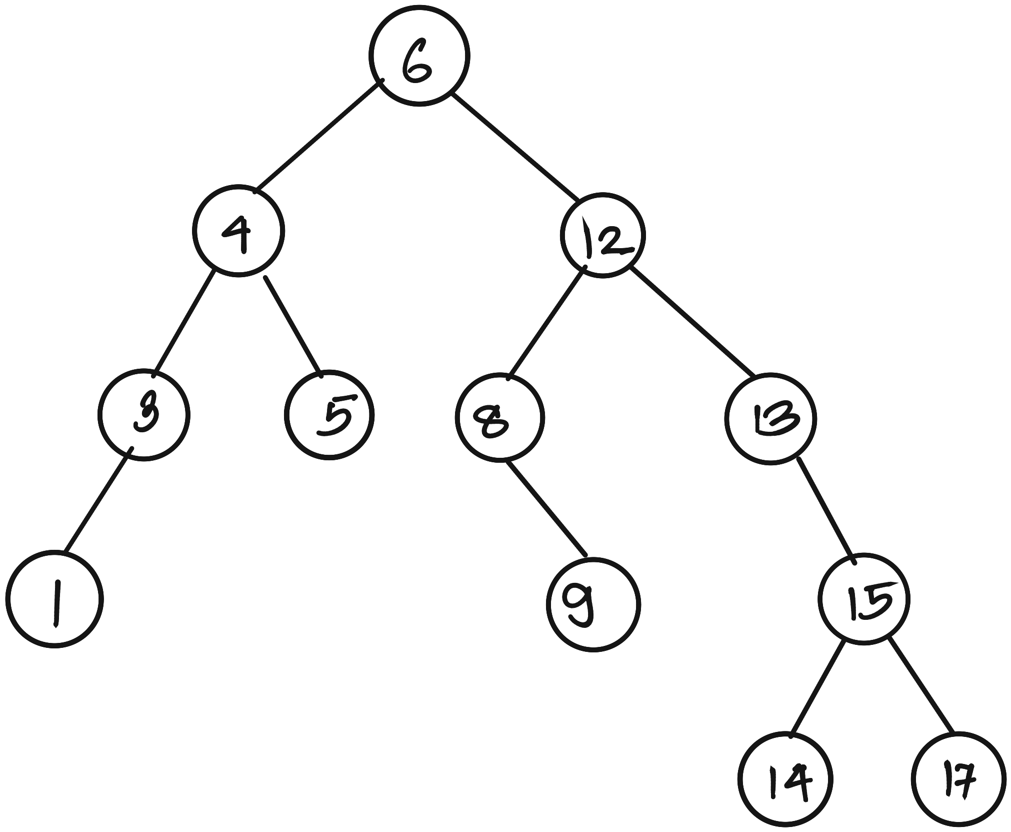
JUST DELETE IT.

DELETE



DELETE 13

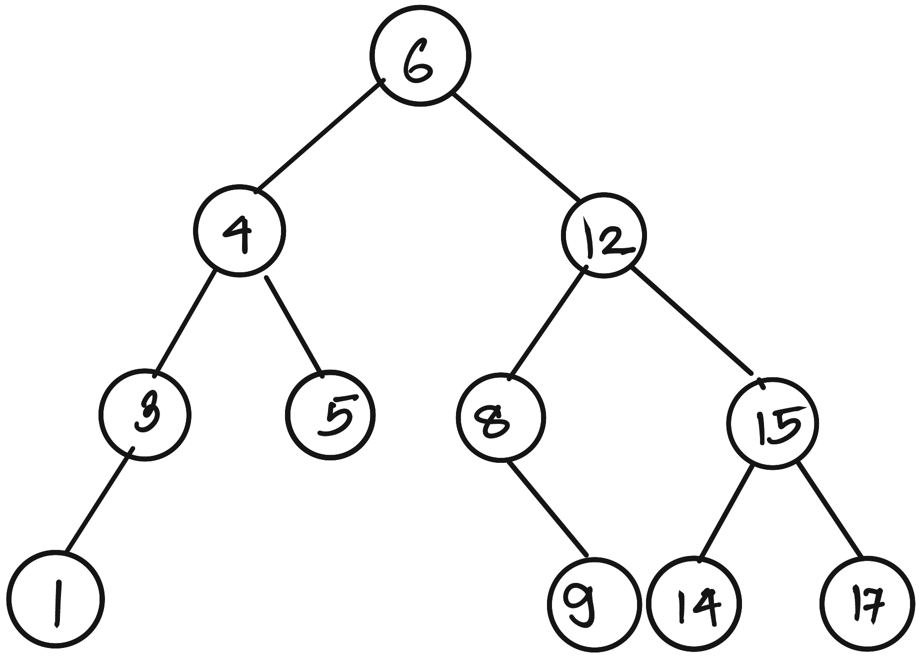
DELETE



DELETE 13

NODE 13 HAS JUST ONE CHILD.

DELETE

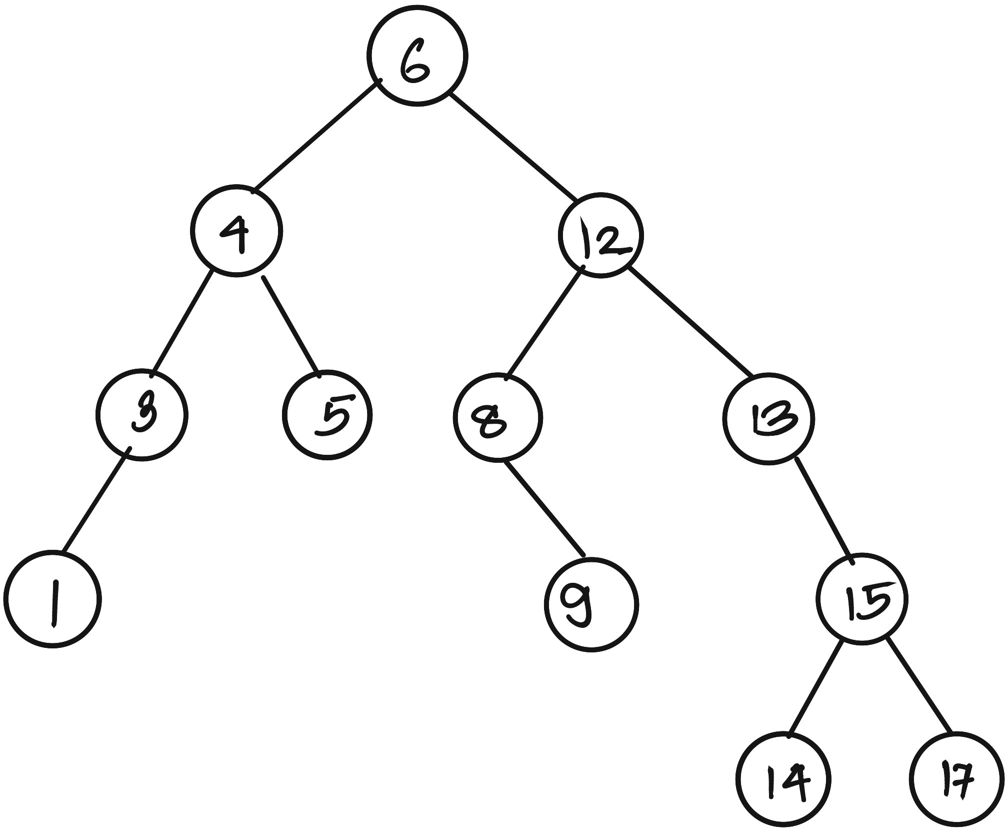


DELETE 13

NODE 13 HAS JUST ONE CHILD.

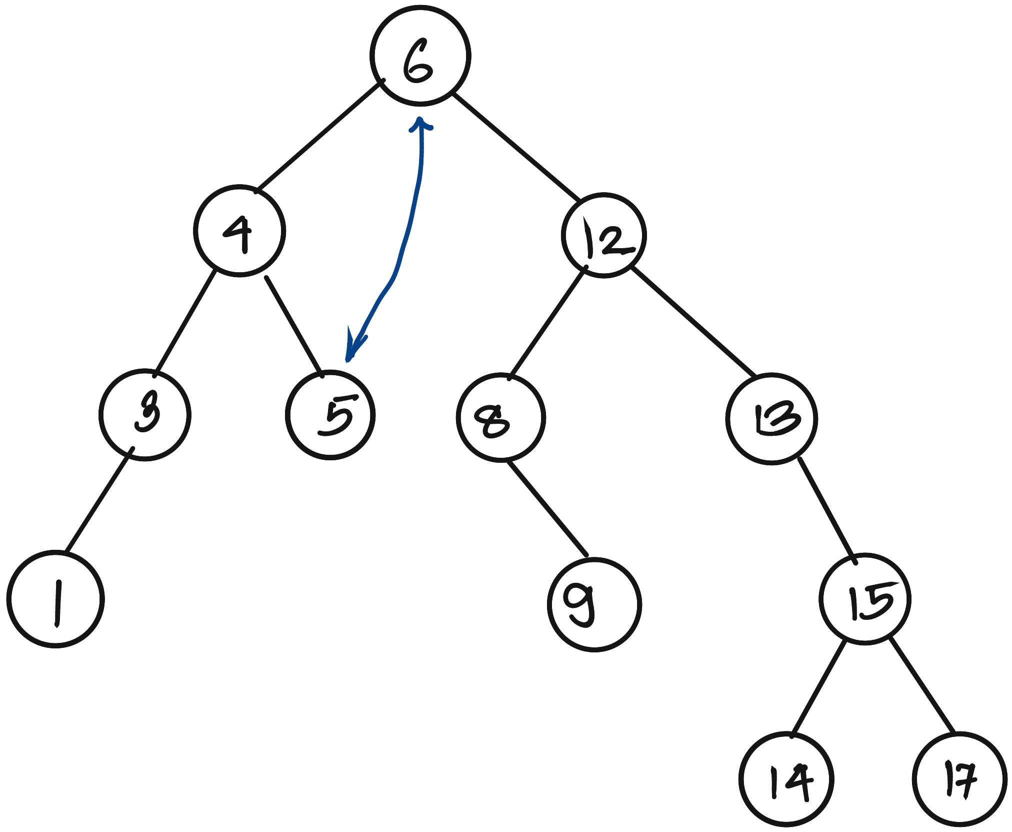
JUST DELETE IT & ATTACH ITS ONLY CHILD TO ITS PARENT

DELETE



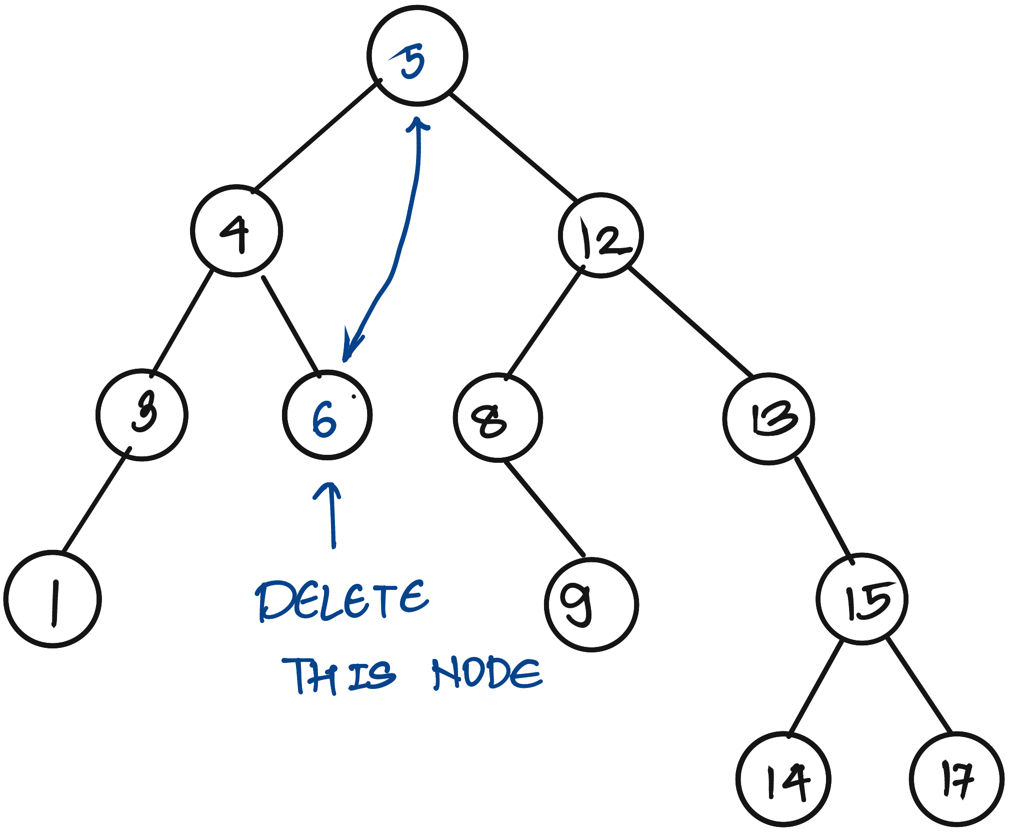
DELETE 6

DELETE

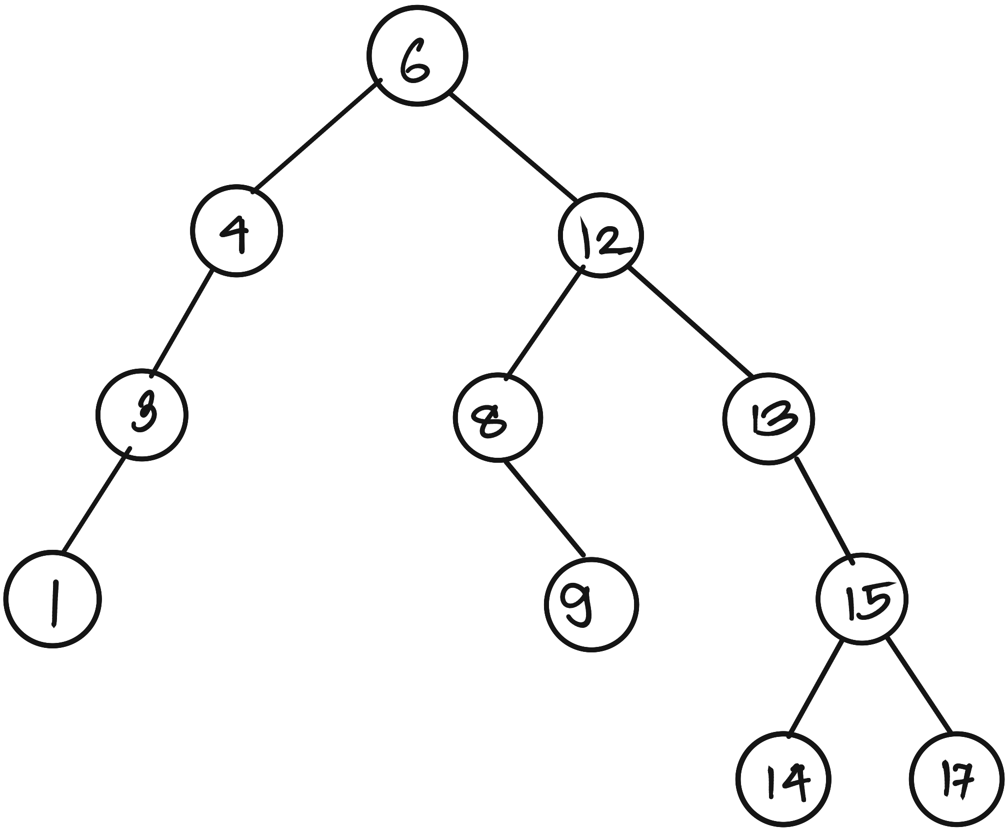


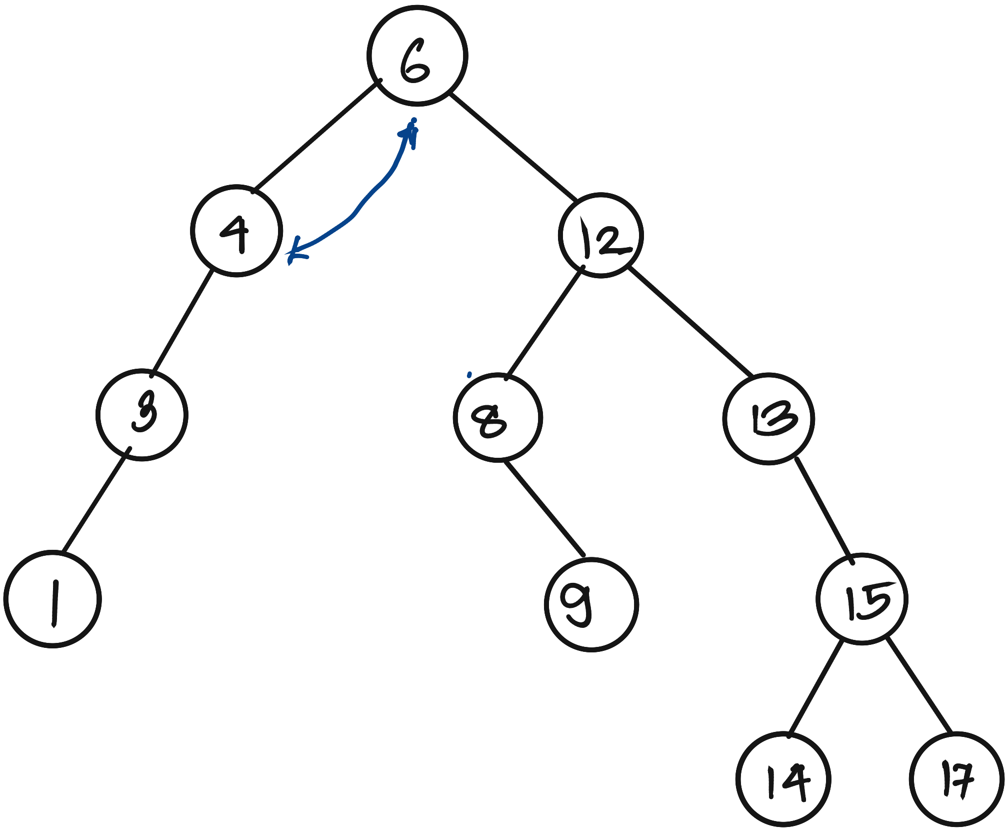
DELETE 6

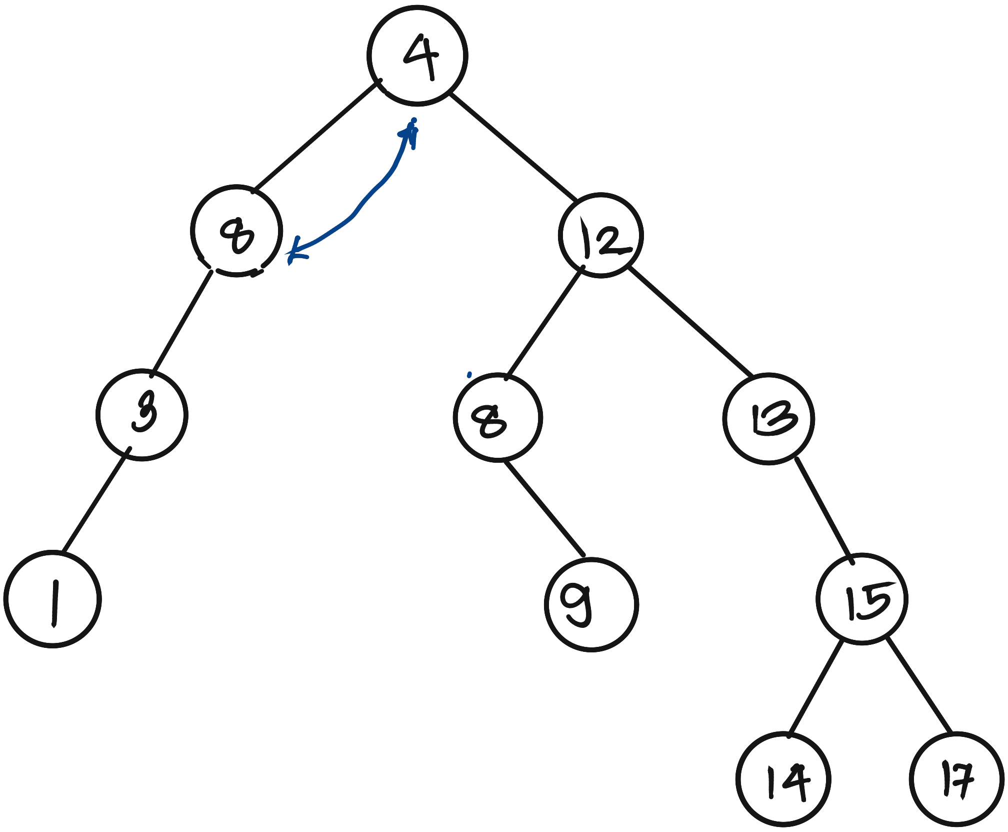
DELETE

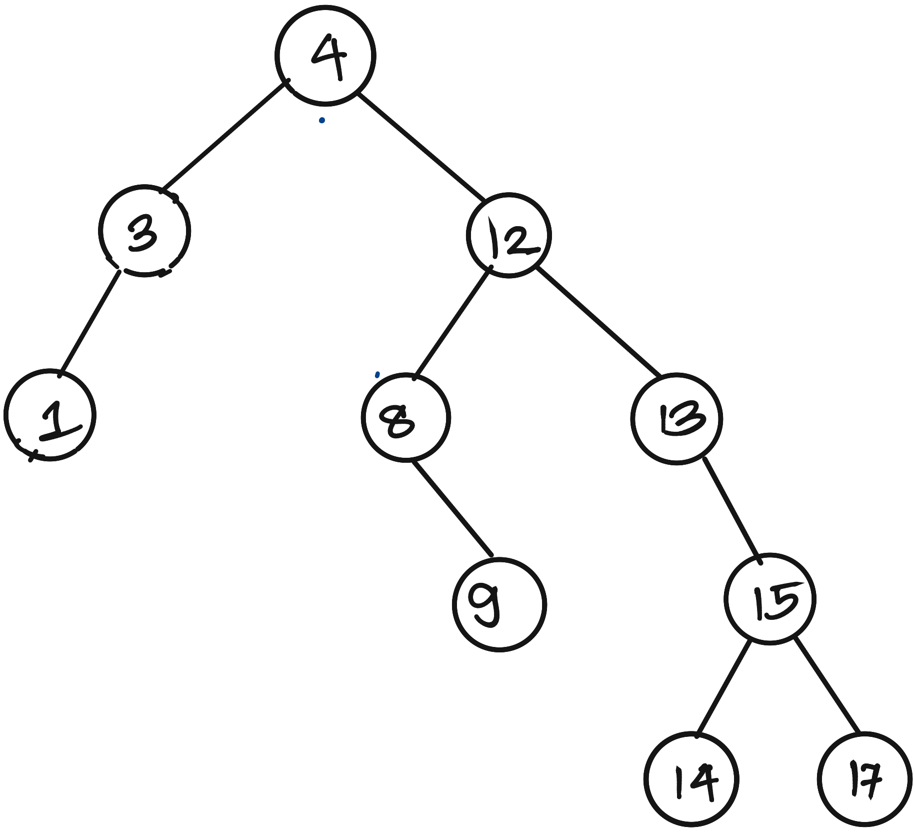


DELETE 6


6
4
12
3
8
13
1
9
15
14
17


6
4
12
3
8
13
1
9
15
14
17


4
8
12
3
8
13
1
9
15
14
17


4
3
12
1
8
13
9
15
14
17

DELETE (a)

OBSERVATION : (1) IF NODE a IS A LEAF OR IT HAS A SINGLE CHILD THEN DELETING IT IS EASY

(2) IF NODE a HAS BOTH LEFT AND RIGHT CHILD THEN

DELETE (a)

OBSERVATION : ① IF NODE a IS A LEAF OR IT HAS A SINGLE CHILD THEN DELETING IT IS EASY

② IF NODE a HAS BOTH LEFT AND RIGHT CHILD THEN

(a) u ← MAXIMUM ELEMENT IN THE LEFT SUBTREE OF NODE a.

<u>Q</u>: WHAT IS THE PROPERTY OF u?

DELETE (a)

OBSERVATION : ① IF NODE a IS A LEAF OR IT HAS A SINGLE CHILD THEN DELETING IT IS EASY

② IF NODE a HAS BOTH LEFT AND RIGHT CHILD THEN

(a) $u \leftarrow$ MAXIMUM ELEMENT IN THE LEFT SUBTREE OF NODE a.

<u>Q</u>: WHAT IS THE PROPERTY OF u?

<u>A</u> : u HAS NO RIGHT CHILD

⇒ u HAS A LEFT CHILD OR u IS A LEAF

DELETE(a)

OBSERVATION : ① IF NODE a IS A LEAF OR IT HAS A SINGLE CHILD THEN DELETING IT IS EASY

② IF NODE a HAS BOTH LEFT AND RIGHT CHILD THEN

(a) u ← MAXIMUM ELEMENT IN THE LEFT SUBTREE OF NODE a.

EASY CASE

Q: WHAT IS THE PROPERTY OF u?

A: u HAS NO RIGHT CHILD

⇒ u HAS A LEFT CHILD OR u IS A LEAF

RUNNING TIME:

DELETE (a)

OBSERVATION : ① IF NODE a IS A LEAF OR IT HAS A SINGLE CHILD THEN DELETING IT IS EASY

② IF NODE a HAS BOTH LEFT AND RIGHT CHILD THEN

(a) u ← MAXIMUM ELEMENT IN THE LEFT SUBTREE OF NODE a.

EASY CASE

Q: WHAT IS THE PROPERTY OF u?

A: u HAS NO RIGHT CHILD
⇒ u HAS A LEFT CHILD OR u IS A LEAF

RUNNING TIME: DOMINATED BY SEARCH TIME:

DELETE (a)

OBSERVATION : ① IF NODE a IS A LEAF OR IT HAS A SINGLE CHILD THEN DELETING IT IS EASY

② IF NODE a HAS BOTH LEFT AND RIGHT CHILD THEN

(a) u ← MAXIMUM ELEMENT IN THE LEFT SUBTREE OF NODE a.

EASY CASE

Q: WHAT IS THE PROPERTY OF u?

A: u HAS NO RIGHT CHILD
⇒ u HAS A LEFT CHILD OR u IS A LEAF

RUNNING TIME: DOMINATED BY SEARCH TIME: $O(h)$.

INSERT : $O(h)$

DELETE : $O(h)$

SEARCH : $O(h)$

h ← MAXIMUM HEIGHT OF BST.

INSERT : $O(h)$
DELETE : $O(h)$
SEARCH : $O(h)$

$h \leftarrow$ MAXIMUM HEIGHT OF BST.

HEIGHT BALANCED BST — HEIGHT $O(\log n)$

AVL TREE
RED-BLACK TREE
2-3-4 TREE

INSERT
DELETE $O(\log n)$
SEARCH

INSERT : $O(h)$
DELETE : $O(h)$
SEARCH : $O(h)$

$h \leftarrow$ MAXIMUM HEIGHT OF BST.

HEIGHT BALANCED BST — HEIGHT $O(\log n)$

$$\left.\begin{array}{l}\text{AVL TREE}\\ \text{RED-BLACK TREE}\\ \text{2-3-4 TREE}\end{array}\right\} \begin{array}{l}\text{INSERT}\\ \text{DELETE}\\ \text{SEARCH}\end{array} \quad O(\log n)$$

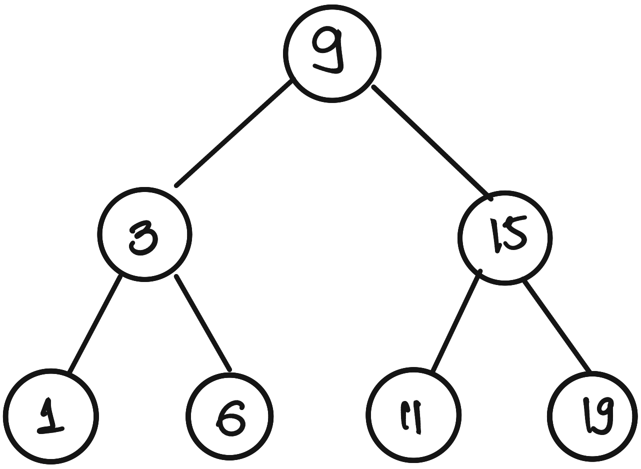
HARD TO IMPLEMENT
HARD TO ANALYSE
FAST IN PRACTICE.

⇒ GIVEN A SORTED ARRAY, WE CAN MAKE A BALANCED BST FROM IT

1	3	6	9	11	15	19

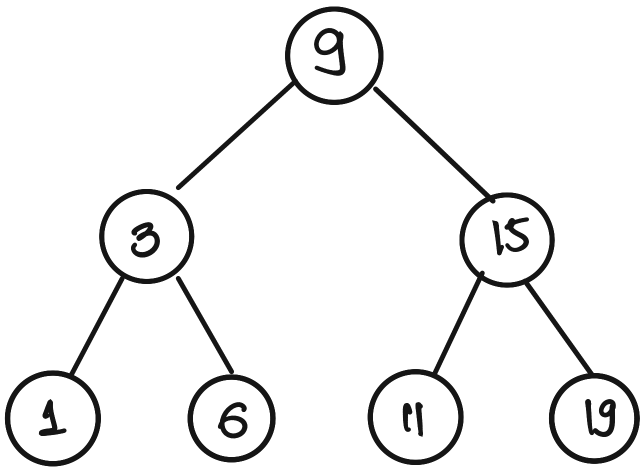
⇒ GIVEN A SORTED ARRAY, WE CAN MAKE A BALANCED BST FROM IT

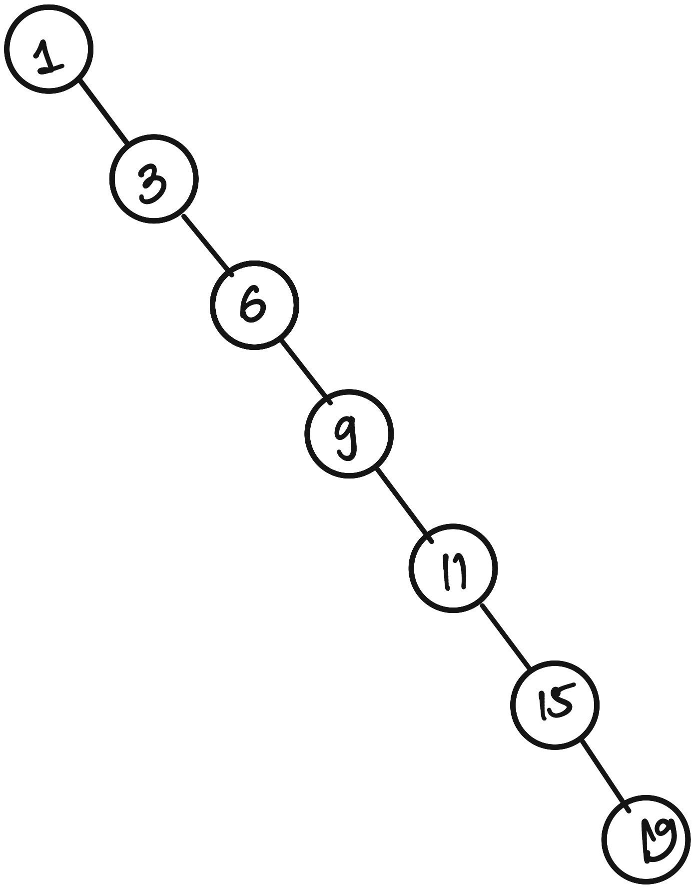
1	3	6	9	11	15	19



⇒ GIVEN A SORTED ARRAY, WE CAN MAKE A BALANCED BST FROM IT

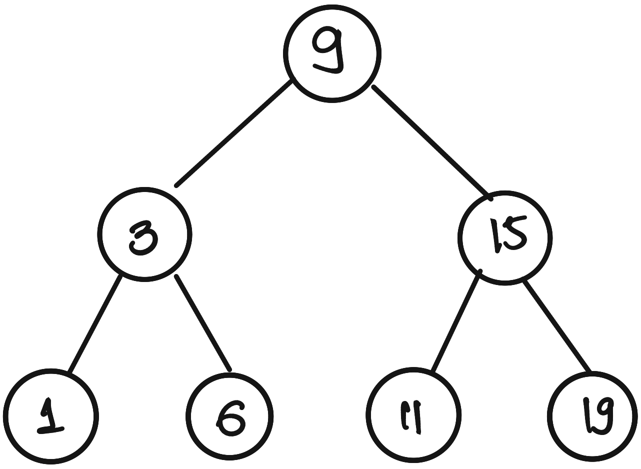
1	3	6	9	11	15	19

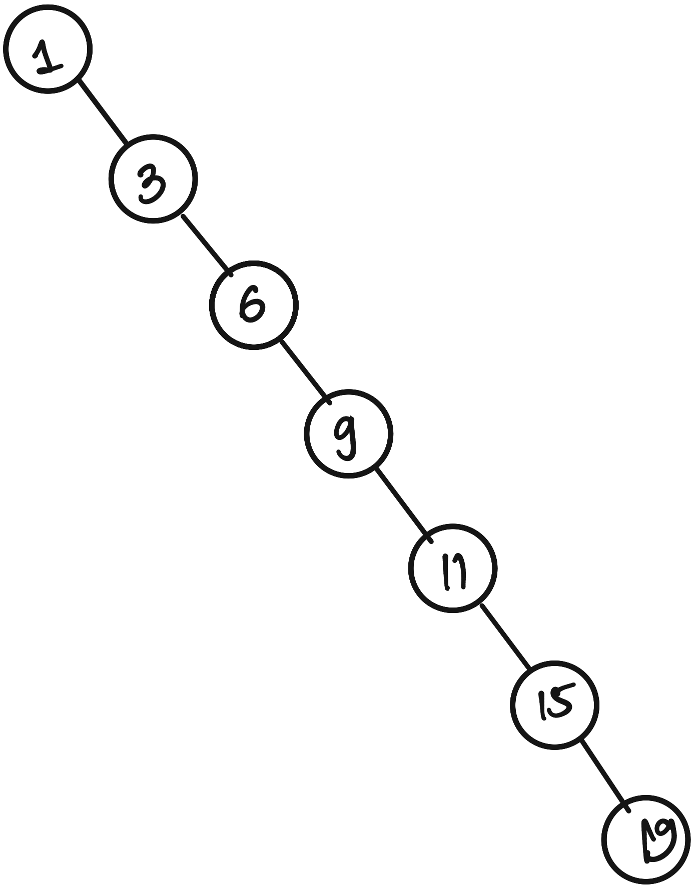




⇒ GIVEN A SORTED ARRAY, WE CAN MAKE A ~~BALANCED~~ BST FROM IT

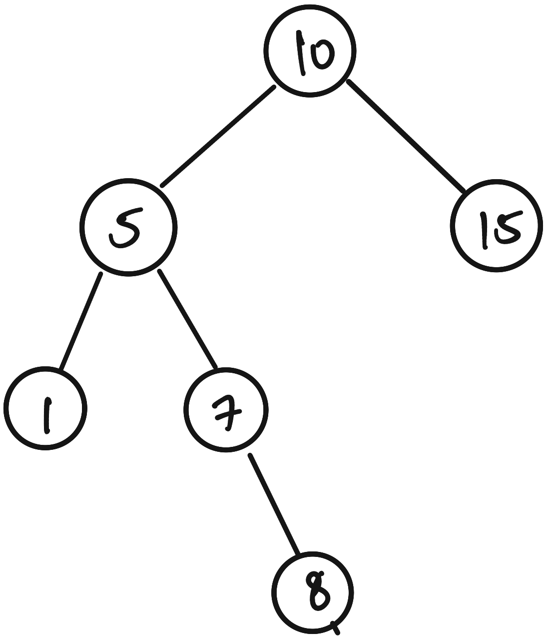
1	3	6	9	11	15	19





⇒ GIVEN A SORTED ARRAY, WE CAN MAKE A ~~BALANCED~~ BST FROM IT

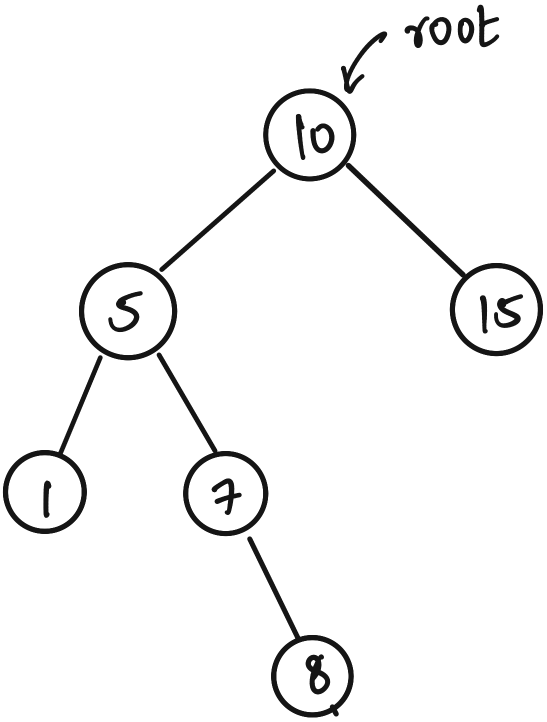
⇐ GIVEN A BST, PRINT ITS CORRESPONDING SORTED ARRAY.



1 5 7 8 10 15

⇒ GIVEN A SORTED ARRAY, WE CAN MAKE A ~~BALANCED~~ BST FROM IT

⇐ GIVEN A BST, PRINT ITS CORRESPONDING SORTED ARRAY.

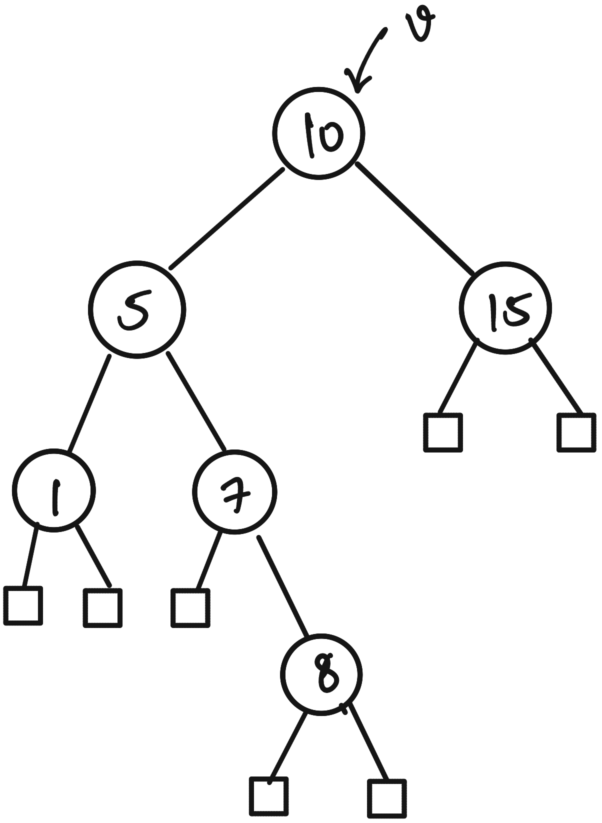


1 5 7 8 10 15

```
INORDER(v)
{  If (v IS NULL)
        RETURN
    INORDER (v.left)
    PRINT (v.value)
    INORDER (v.right)
}
```

⇒ GIVEN A SORTED ARRAY, WE CAN MAKE A ~~BALANCED~~ BST FROM IT

⇐ GIVEN A BST, PRINT ITS CORRESPONDING SORTED ARRAY.

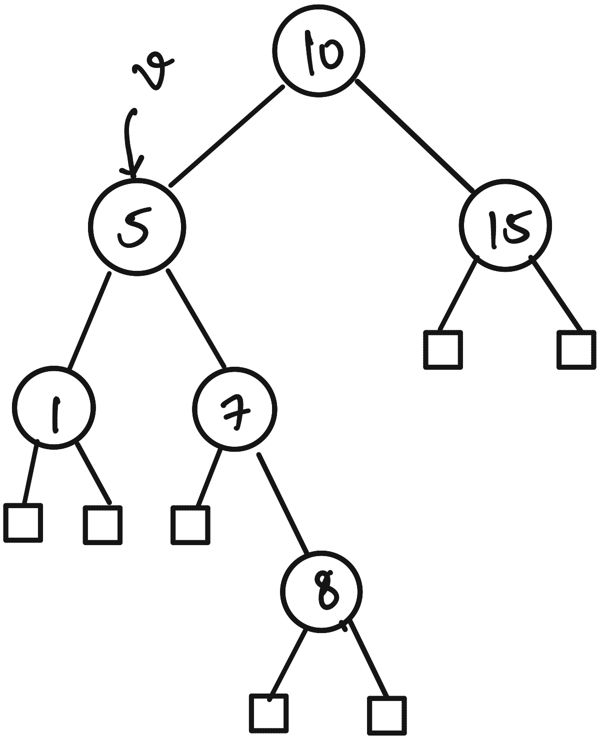


1 5 7 8 10 15

```
INORDER(v)
{  If (v IS NULL)
       RETURN
    INORDER (v.left)
    PRINT (v.value)
    INORDER (v.right)
}
```

⇒ GIVEN A SORTED ARRAY, WE CAN MAKE A ~~BALANCED~~ BST FROM IT

⇐ GIVEN A BST, PRINT ITS CORRESPONDING SORTED ARRAY.

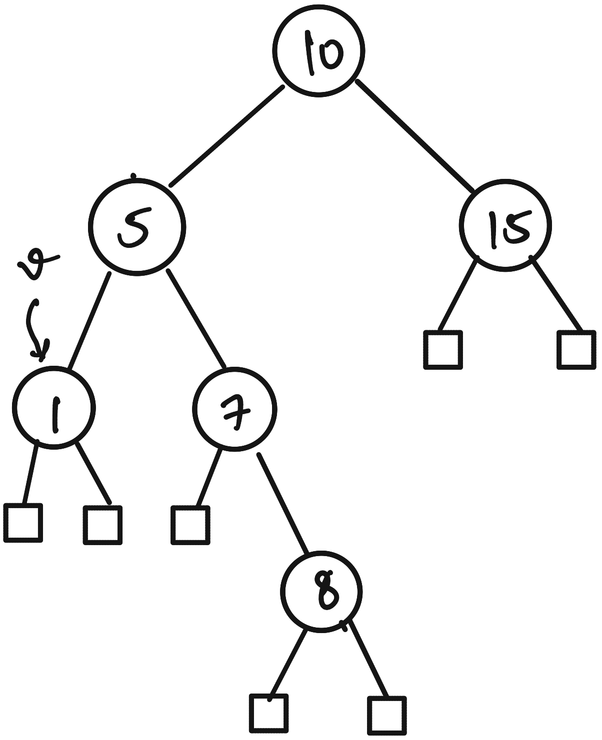


1 5 7 8 10 15

```
INORDER(v)
{  If (v IS NULL)
        RETURN
     INORDER (v.left)
     PRINT (v.value)
     INORDER (v.right)
}
```

⇒ GIVEN A SORTED ARRAY, WE CAN MAKE A ~~BALANCED~~ BST FROM IT

⇐ GIVEN A BST, PRINT ITS CORRESPONDING SORTED ARRAY.

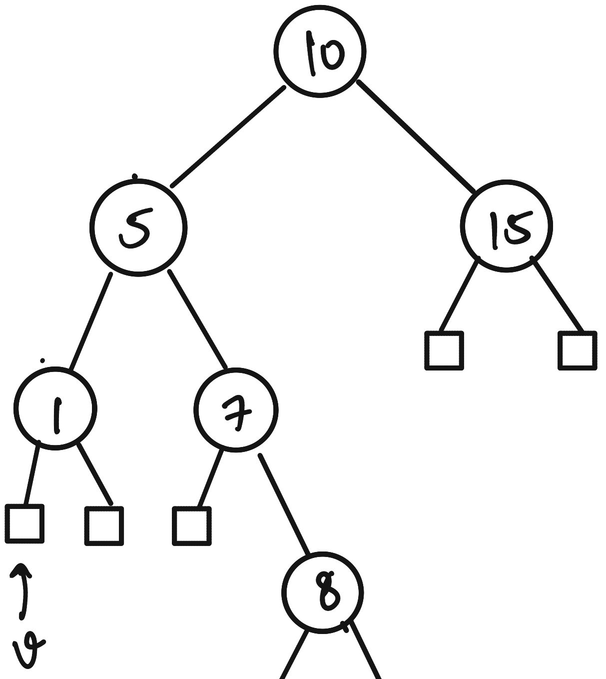


1 5 7 8 10 15

```
INORDER(v)
{   If(v IS NULL)
        RETURN
    INORDER(v.left)
    PRINT(v.value)
    INORDER(v.right)
}
```

⇒ GIVEN A SORTED ARRAY, WE CAN MAKE A ~~BALANCED~~ BST FROM IT

⇐ GIVEN A BST, PRINT ITS CORRESPONDING SORTED ARRAY.

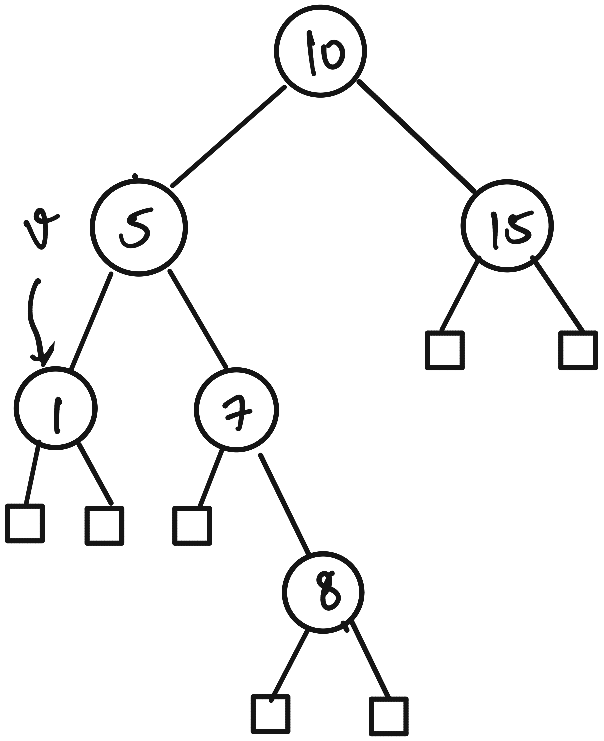


1 5 7 8 10 15

```
INORDER(v)
{  If (v IS NULL)
        RETURN
    INORDER (v.left)
    PRINT (v.value)
    INORDER (v.right)
}
```

⇒ GIVEN A SORTED ARRAY, WE CAN MAKE A ~~BALANCED~~ BST FROM IT

⇐ GIVEN A BST, PRINT ITS CORRESPONDING SORTED ARRAY.



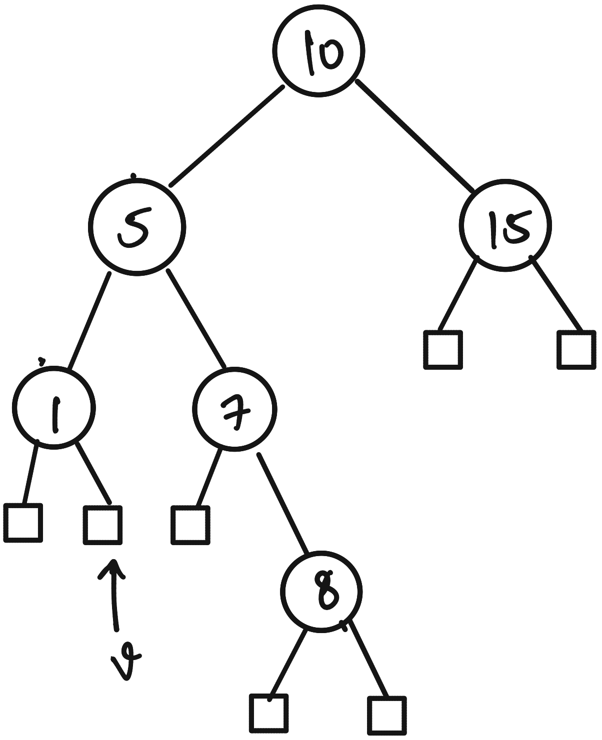
1 5 7 8 10 15

```
INORDER(v)
{  If(v IS NULL)
       RETURN
   INORDER(v.left)
   PRINT(v.value)
   INORDER(v.right)
}
```

1

⇒ GIVEN A SORTED ARRAY, WE CAN MAKE A ~~BALANCED~~ BST FROM IT

⇐ GIVEN A BST, PRINT ITS CORRESPONDING SORTED ARRAY.



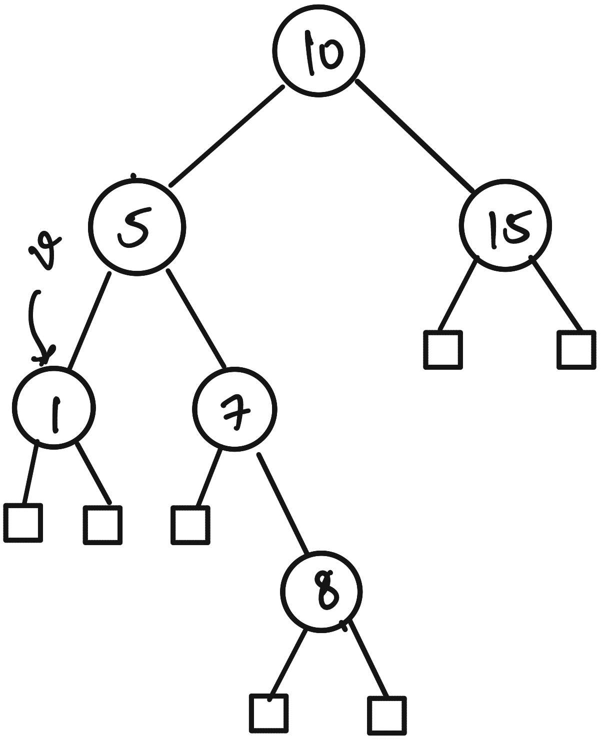
1 5 7 8 10 15

```
INORDER(v)
{ If (v IS NULL)
      RETURN
   INORDER (v.left)
   PRINT (v.value)
   INORDER (v.right)
}
```

1

⇒ GIVEN A SORTED ARRAY, WE CAN MAKE A ~~BALANCED~~ BST FROM IT

⇐ GIVEN A BST, PRINT ITS CORRESPONDING SORTED ARRAY.



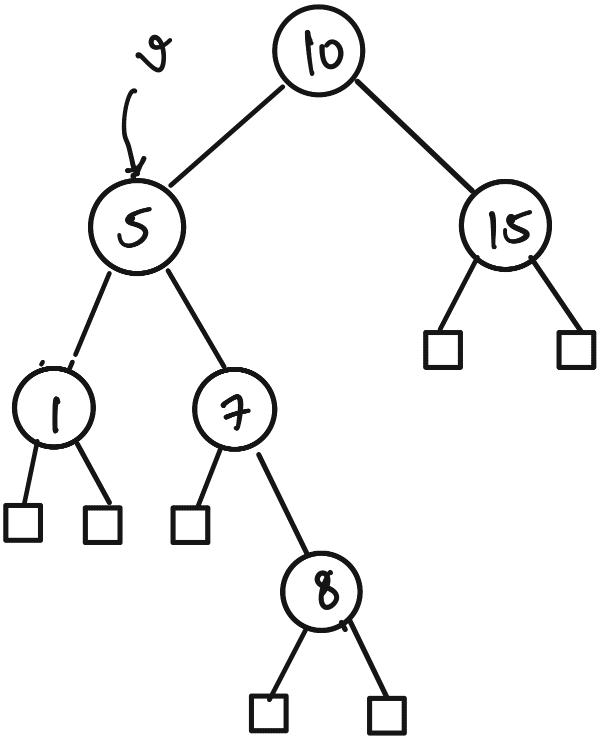
1 5 7 8 10 15

```
INORDER(v)
{  If (v IS NULL)
        RETURN
   INORDER (v.left)
   PRINT (v.value)
   INORDER (v.right)
}
```

1

⇒ GIVEN A SORTED ARRAY, WE CAN MAKE A ~~BALANCED~~ BST FROM IT

⇐ GIVEN A BST, PRINT ITS CORRESPONDING SORTED ARRAY.



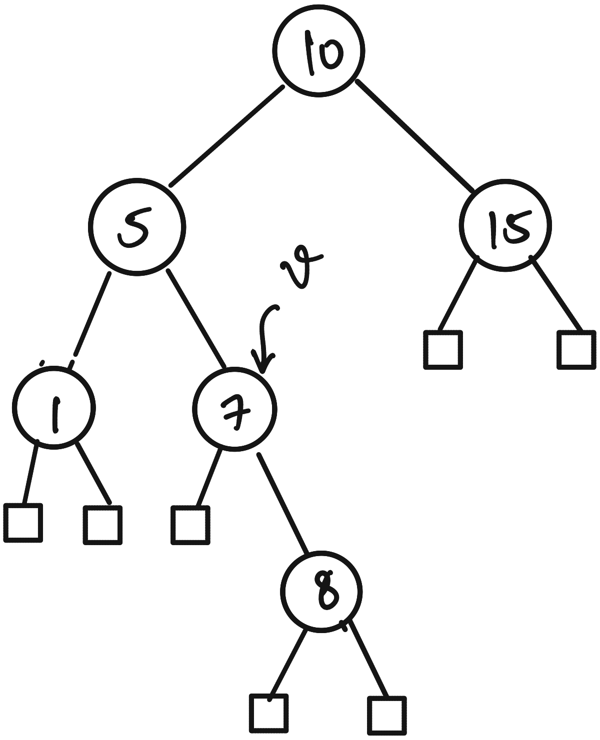
1 5 7 8 10 15

```
INORDER(v)
{   If (v IS NULL)
        RETURN
    INORDER (v.left)
    PRINT (v.value)
    INORDER (v.right)
}
```

1 5

⇒ GIVEN A SORTED ARRAY, WE CAN MAKE A ~~BALANCED~~ BST FROM IT

⇐ GIVEN A BST, PRINT ITS CORRESPONDING SORTED ARRAY.



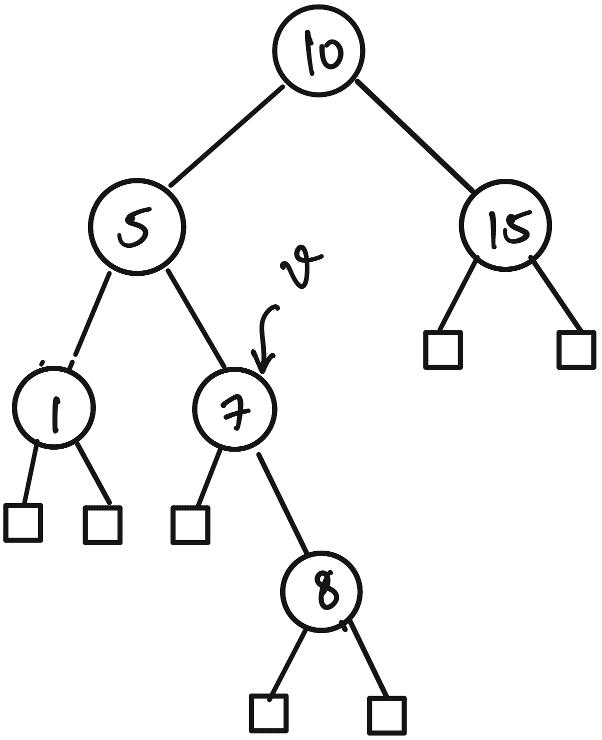
1 5 7 8 10 15

```
INORDER(v)
{  If (v IS NULL)
        RETURN
    INORDER (v.left)
    PRINT (v.value)
    INORDER (v.right)
}
```

1 5

⇒ GIVEN A SORTED ARRAY, WE CAN MAKE A ~~BALANCED~~ BST FROM IT

⇐ GIVEN A BST, PRINT ITS CORRESPONDING SORTED ARRAY.



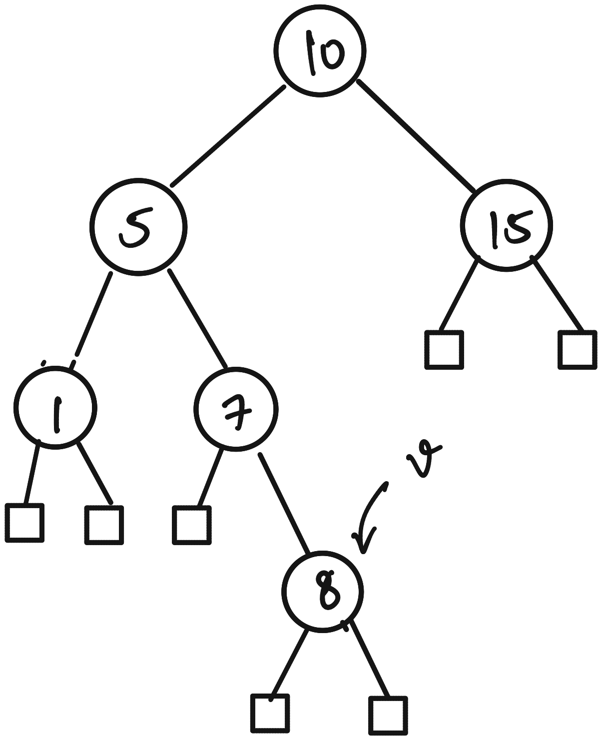
1 5 7 8 10 15

```
INORDER(v)
{  If (v IS NULL)
       RETURN
   INORDER (v.left)
   PRINT (v.value)
   INORDER (v.right)
}
```

1 5 7

⇒ GIVEN A SORTED ARRAY, WE CAN MAKE A ~~BALANCED~~ BST FROM IT

⇐ GIVEN A BST, PRINT ITS CORRESPONDING SORTED ARRAY.



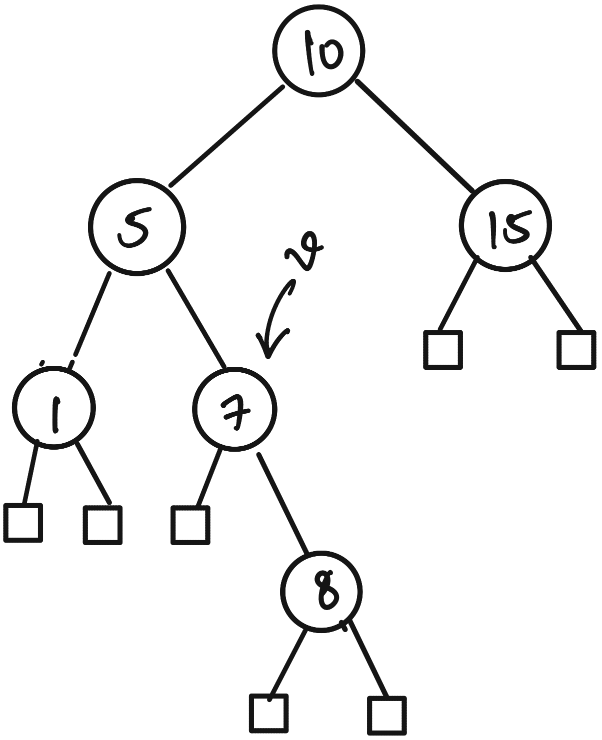
1 5 7 8 10 15

```
INORDER(v)
{   If (v IS NULL)
        RETURN
    INORDER (v.left)
    PRINT (v.value)
    INORDER (v.right)
}
```

1 5 7

⇒ GIVEN A SORTED ARRAY, WE CAN MAKE A ~~BALANCED~~ BST FROM IT

⇐ GIVEN A BST, PRINT ITS CORRESPONDING SORTED ARRAY.



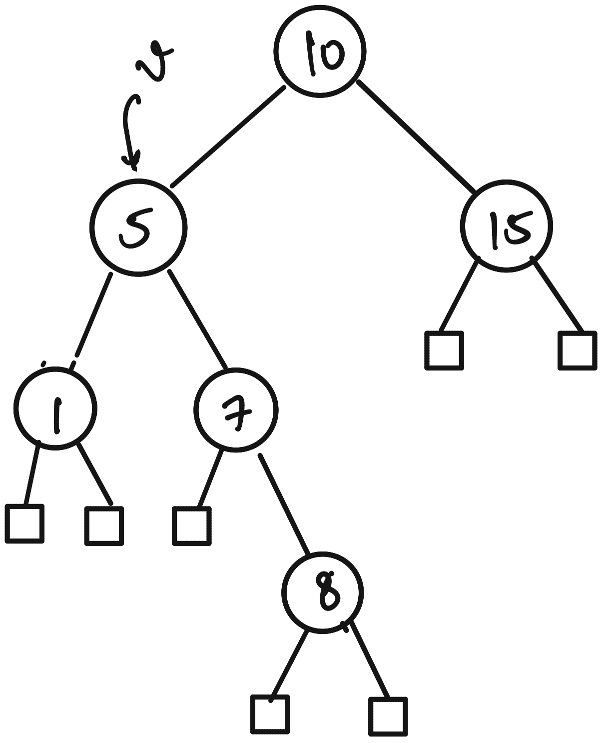
1 5 7 8 10 15

```
INORDER(v)
{  If (v IS NULL)
       RETURN
   INORDER(v.left)
   PRINT(v.value)
   INORDER(v.right)
}
```

1 5 7 8

⇒ GIVEN A SORTED ARRAY, WE CAN MAKE A ~~BALANCED~~ BST FROM IT

⇐ GIVEN A BST, PRINT ITS CORRESPONDING SORTED ARRAY.



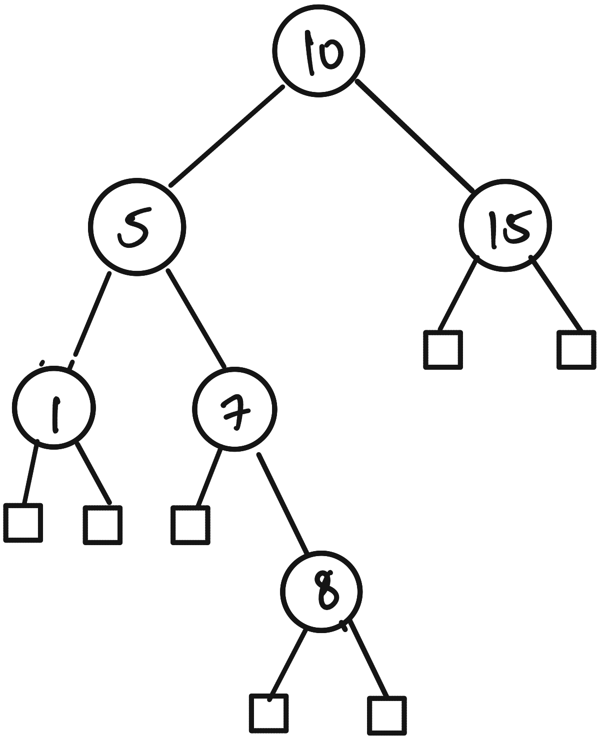
1 5 7 8 10 15

```
INORDER(v)
{  If (v IS NULL)
        RETURN
     INORDER (v.left)
     PRINT (v.value)
     INORDER (v.right)
}
```

1 5 7 8

⇒ GIVEN A SORTED ARRAY, WE CAN MAKE A ~~BALANCED~~ BST FROM IT

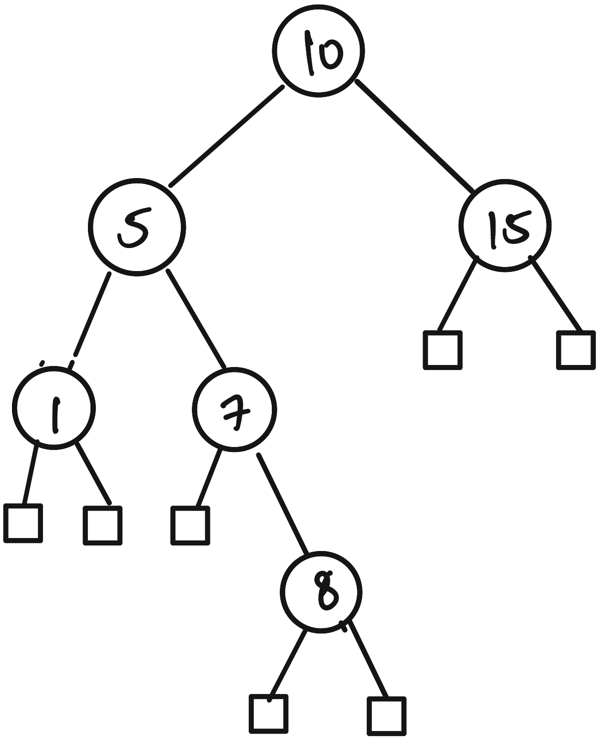
⇐ GIVEN A BST, PRINT ITS CORRESPONDING SORTED ARRAY.



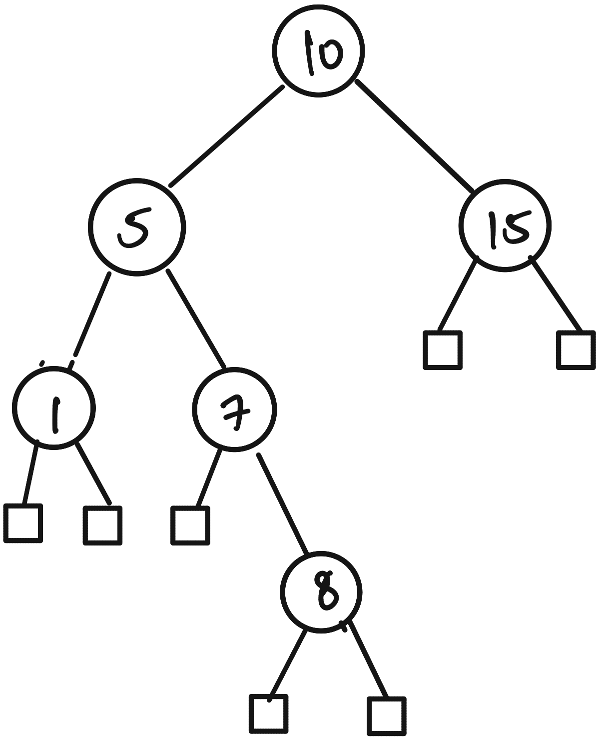
1 5 7 8 10 15

```
INORDER(v)
{  If (v IS NULL)
       RETURN
   INORDER (v.left)
   PRINT (v.value)
   INORDER (v.right)
}
```

1 5 7 8 10 15

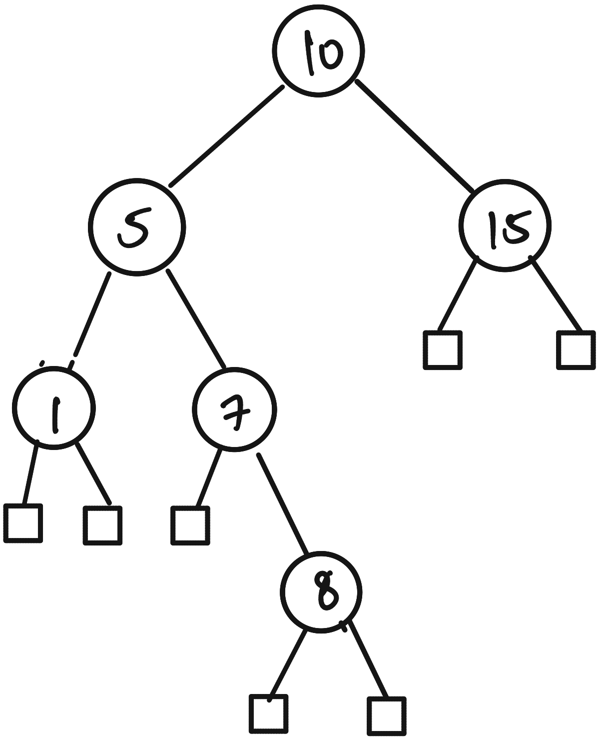


```
PREORDER(v)
{   If (v IS NULL)
        RETURN
    PRINT (v.value)
    PRE-ORDER (v.left)
    PRE-ORDER (v.right)
}
```

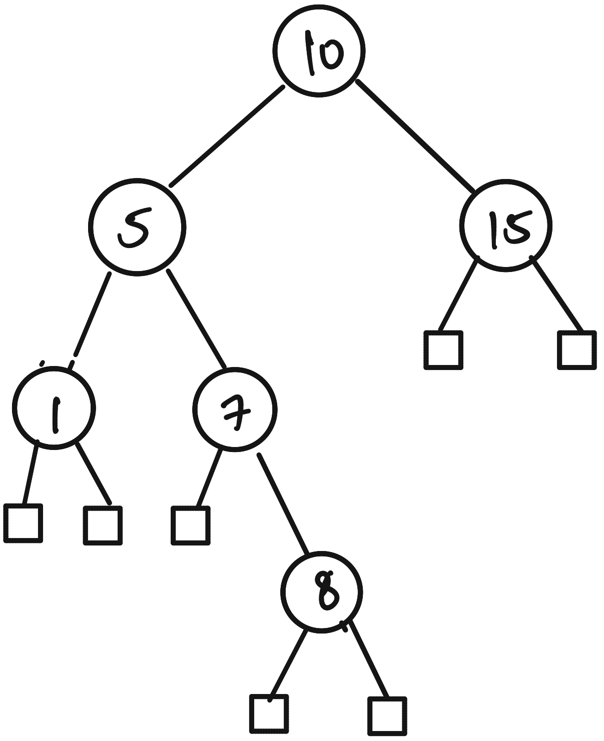


```
PREORDER(v)
{   If(v IS NULL)
        RETURN
    PRINT (v.value)
    PRE-ORDER (v.left)
    PRE-ORDER (v.right)
}
```

10 5 1 7 8 15



```
POSTORDER(v)
{   If(v IS NULL)
        RETURN
    POST ORDER (v.left)
    POST ORDER (v.right)
    PRINT (v.value)
}
```



```
POSTORDER(v)
{   If(v IS NULL)
        RETURN
    POST ORDER (v.left)
    POST ORDER (v.right)
    PRINT (v.value)
}
```

1 8 7 5 15 10

PROPERTIES OF TRAVERSALS

INORDER :

PREORDER : WHERE IS THE ROOT ?

POSTORDER:

PROPERTIES OF TRAVERSALS

INORDER :

PREORDER : FIRST

POSTORDER:

PROPERTIES OF TRAVERSALS

INORDER :

PREORDER : FIRST

POSTORDER: LAST

PROPERTIES OF TRAVERSALS

INORDER : AFTER ALL THE ELEMENTS OF LEFT SUBTREE OF ROOT.

PREORDER : FIRST

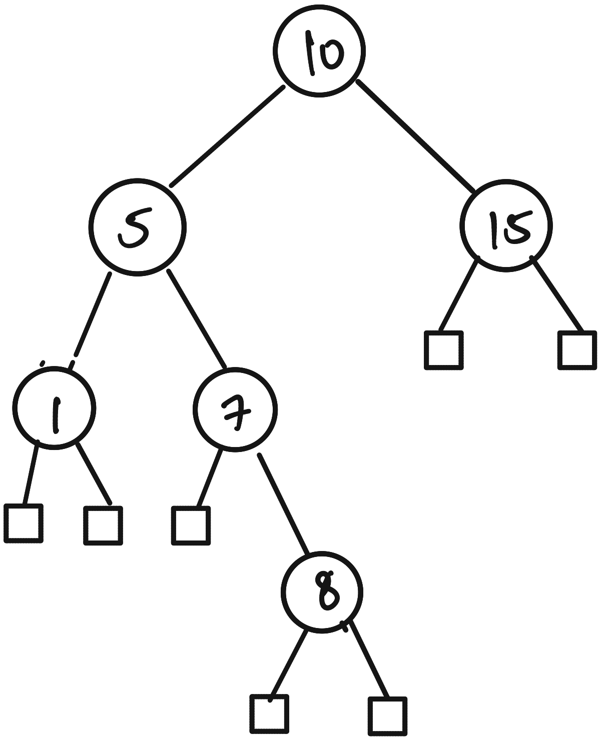
POSTORDER: LAST

PROPERTIES OF TRAVERSALS

INORDER : AFTER ALL THE ELEMENTS OF LEFT SUBTREE OF ROOT.

PREORDER : FIRST

POSTORDER : LAST



```
PREORDER(v)
{   If(v IS NULL)
        RETURN
    PRINT (v.value)
    PRE-ORDER (v.left)
    PRE-ORDER (v.right)
}
```

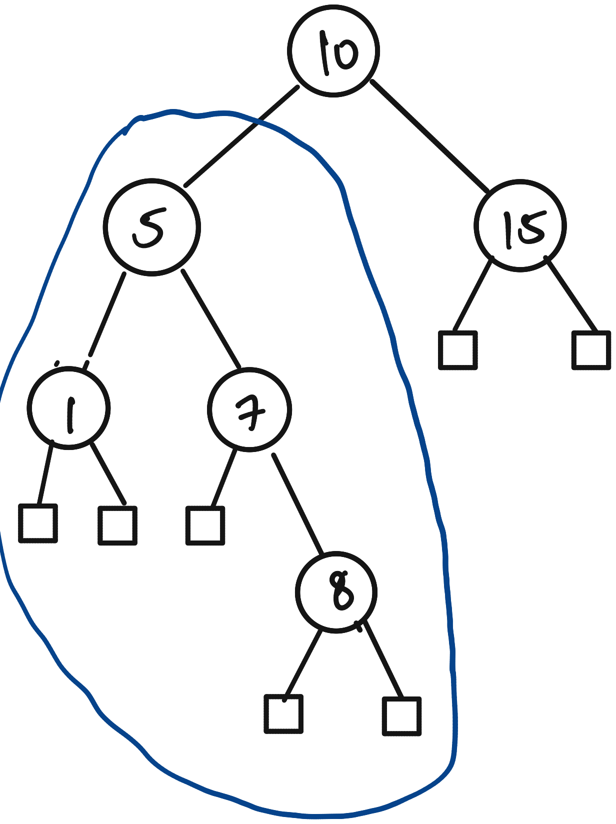
10 5 1 7 8 15

PROPERTIES OF TRAVERSALS

INORDER : AFTER ALL THE ELEMENTS OF LEFT SUBTREE OF ROOT.

PREORDER : FIRST

POSTORDER : LAST



```
PREORDER(v)
{   If (v IS NULL)
        RETURN
    PRINT (v.value)
    PRE-ORDER (v.left)
    PRE-ORDER (v.right)
}
```

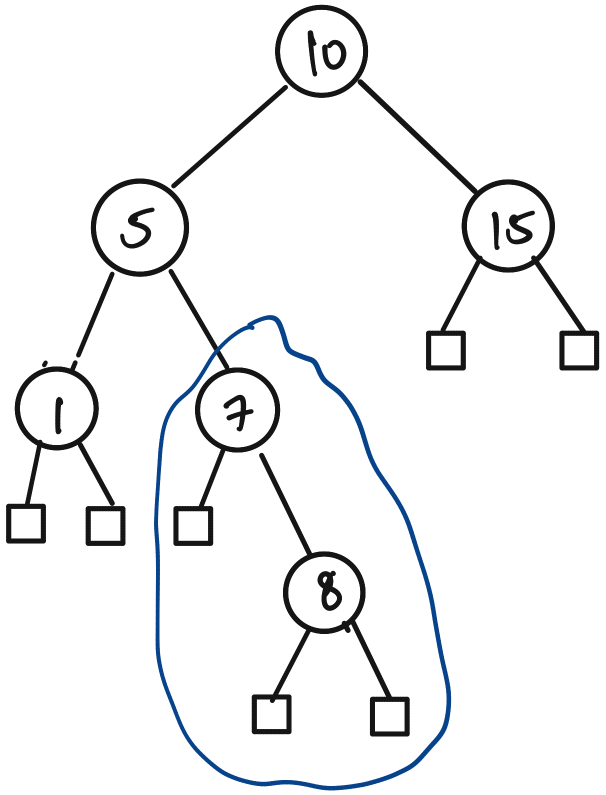
10 5 1 7 8 15

PROPERTIES OF TRAVERSALS

INORDER : AFTER ALL THE ELEMENTS OF LEFT SUBTREE OF ROOT.

PREORDER : FIRST

POSTORDER : LAST



```
POSTORDER(v)
{  If (v IS NULL)
        RETURN
      POST ORDER (v.left)
      POST ORDER (v.right)
      PRINT (v.value)
}
```

1 8 7 5 15 10

PREORDER : 5 4 1 12 6 13

INORDER : 1 4 5 6 12 13

PREORDER : 5 4 1 12 6 13

INORDER : 1 4 5 6 12 13

ROOT OF TREE

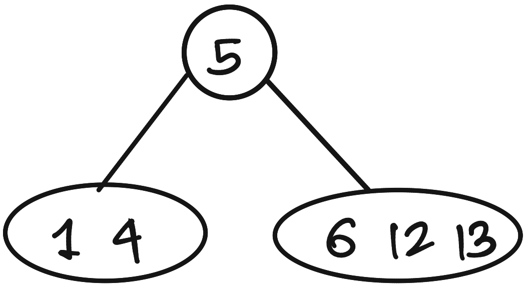
PRE ORDER : 5 4 1 12 6 13

INORDER : $\underbrace{1 \quad 4}_{\text{Left Subtree}} \quad 5 \quad \underbrace{6 \quad 12 \quad 13}_{\text{RIGHT SUBTREE}}$

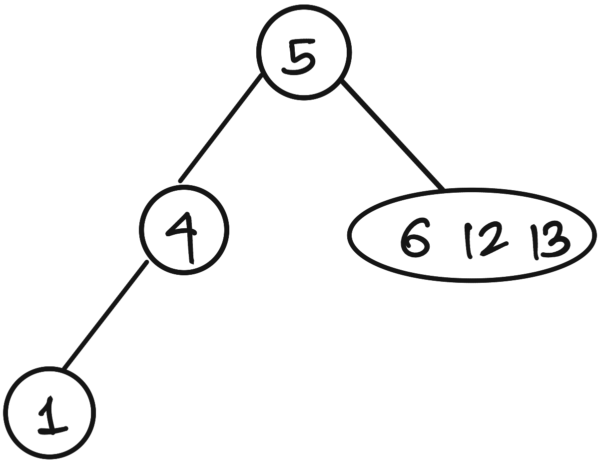
ROOT OF TREE

PRE ORDER : 5 4 1 12 6 13

INORDER : 1 4 5 6 12 13

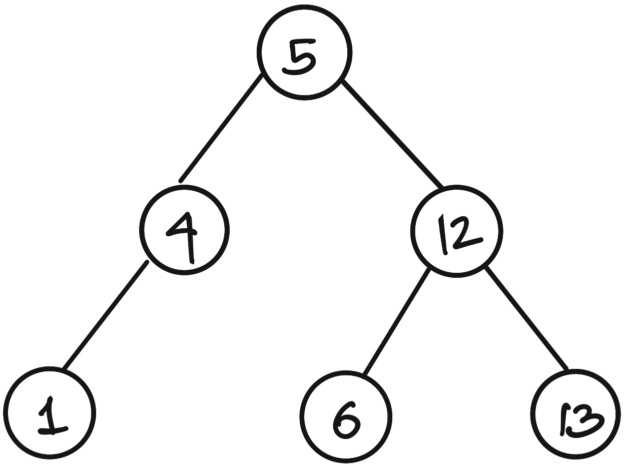


PRE ORDER :	5	4	1	12	6	13
INORDER :	1	4	5	6	12	13



PREORDER : 5 4 1 12 6 13

INORDER : 1 4 5 6 12 13

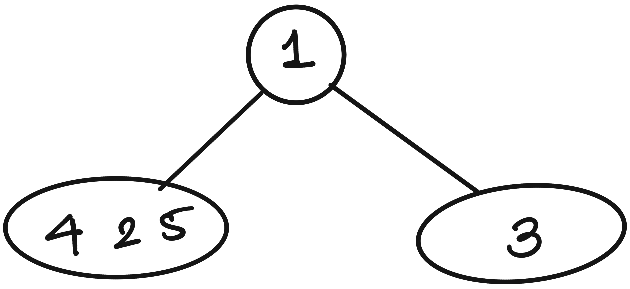


POST-ORDER : 4 5 2 3 1

INORDER : 4 2 5 1 3

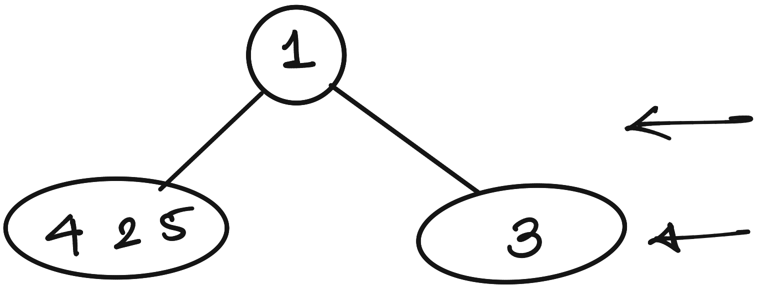
POST-ORDER : 4 5 2 3 1

INORDER : 4 2 5 1 3



POST-ORDER : 4 5 2 3 1

INORDER : 4 2 5 1 3

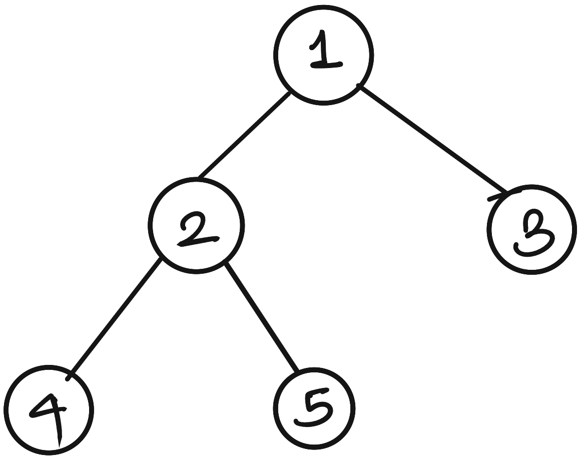


NOT A BST

TRAVERSAL WORKS FOR ANY BINARY TREE, NOT JUST FOR BST.

POST-ORDER : 4 5 2 3 1

INORDER : 4 2 5 1 3

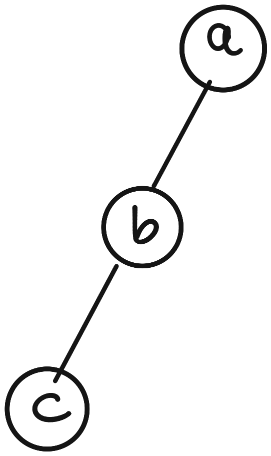


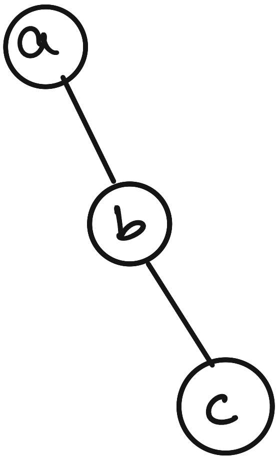
PREORDER : a b c

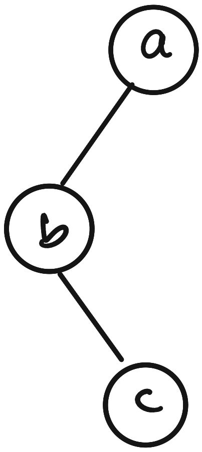
POSTORDER : c b a

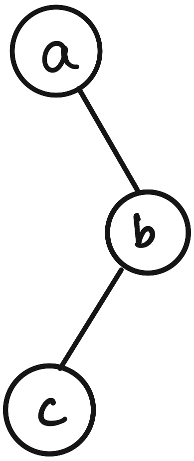
PREORDER : a b c

POSTORDER : c b a



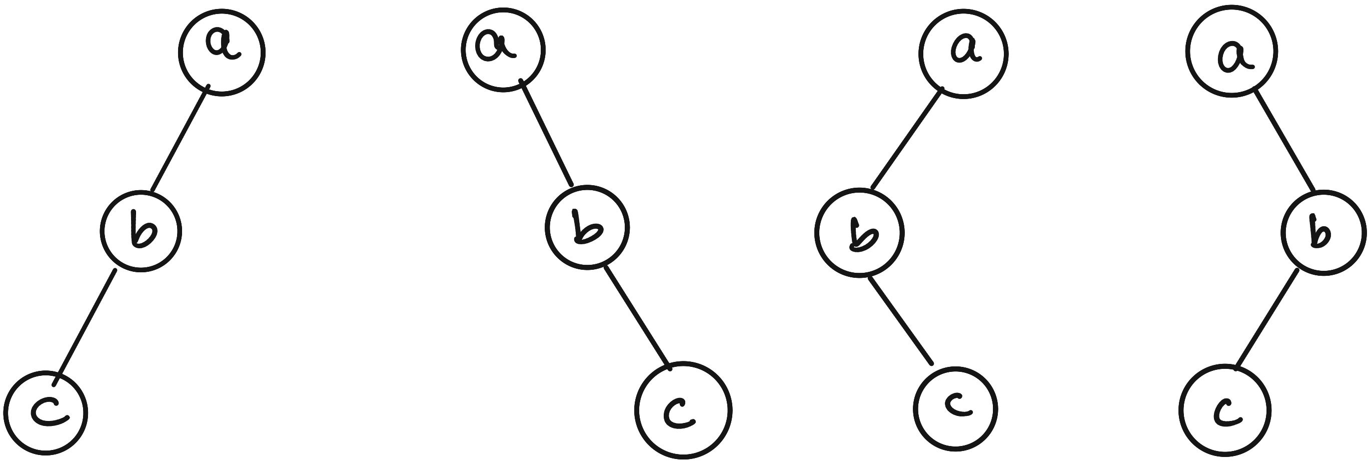






PREORDER : a b c

POSTORDER : c b a



NO UNIQUE BINARY TREE IF ONLY PRE-ORDER & POSTORDER TRAVERSAL IS GIVEN.