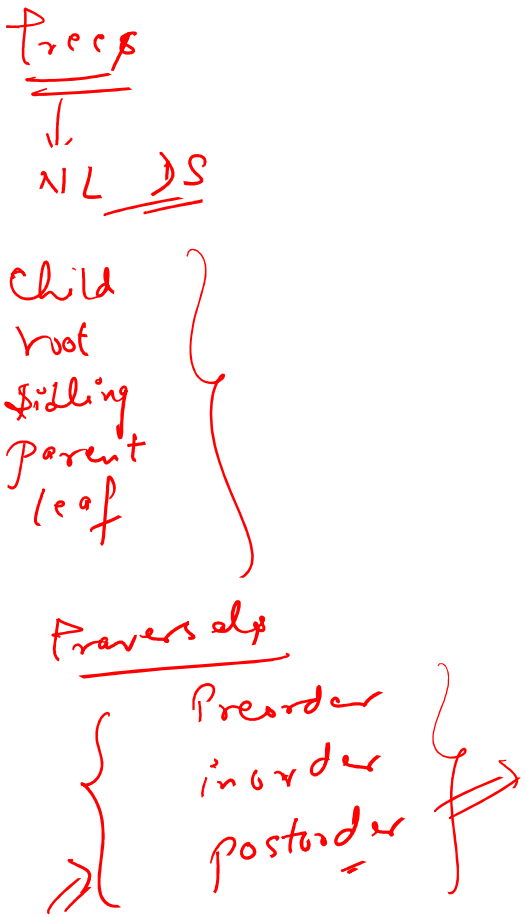
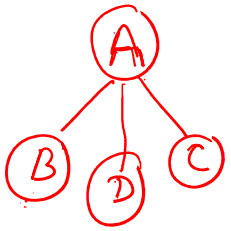
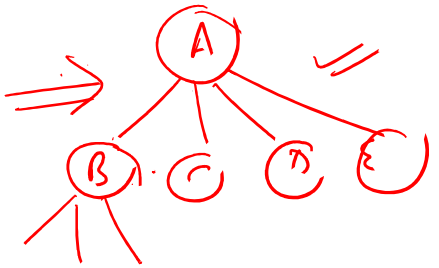

Trees
NIL DS
Child
root
sibling
parent
leaf
Traversals
Preorder
inorder
postorder

Trees

- Binary Tree
- BST
- AVL



Trees ⟹ 

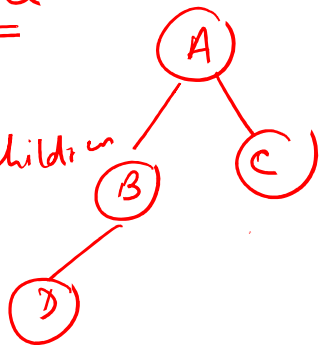
= General

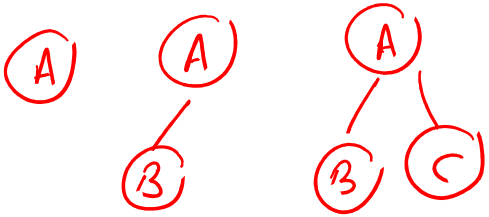
Binary Tree

2

at most 2 children

0 child
1 child
2 children

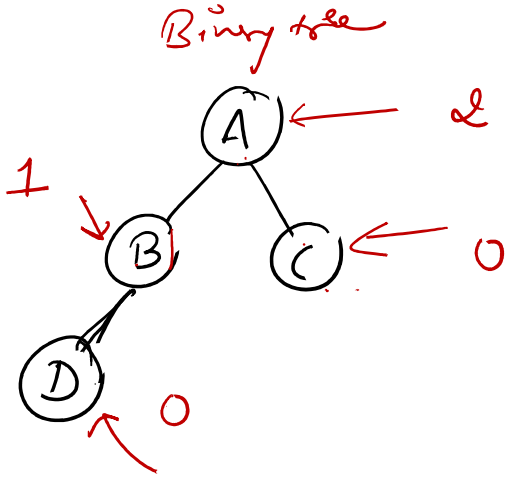




Binary Tree

max 2 children

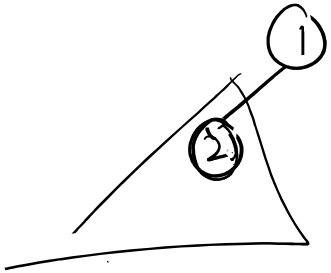
at most 0, 1, 2

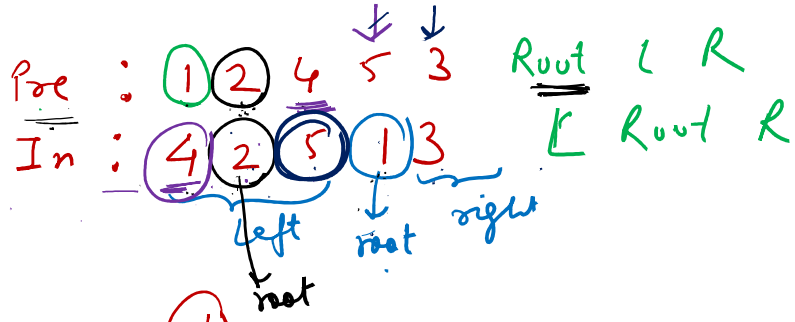


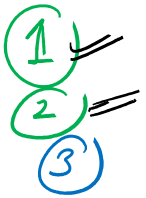
Pre order : Root L R

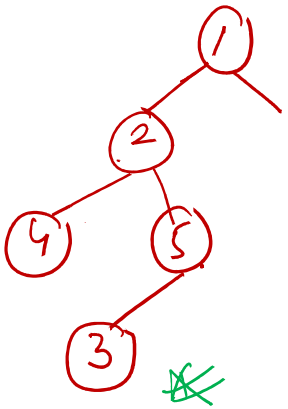
In : L Root R

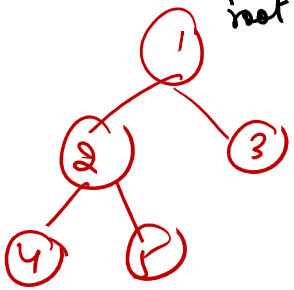
Post : L R Root

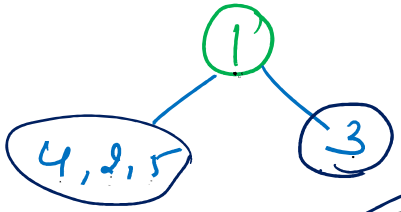




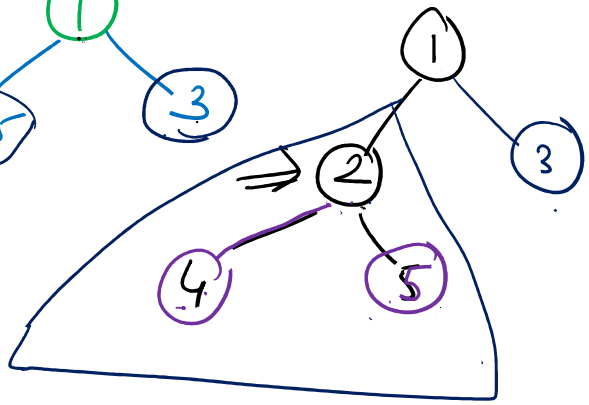








Pre : 2 4 5



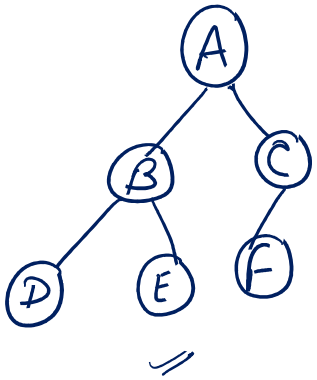
(1) find the root in preorder,
(1st element)

(2) find the position of root in inorder.

(3) values which are on the left side are elements of left subtree
right side right

Pre : A B D E C F

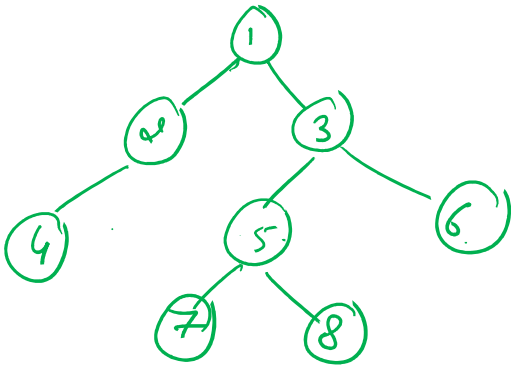
In : D B E A F C

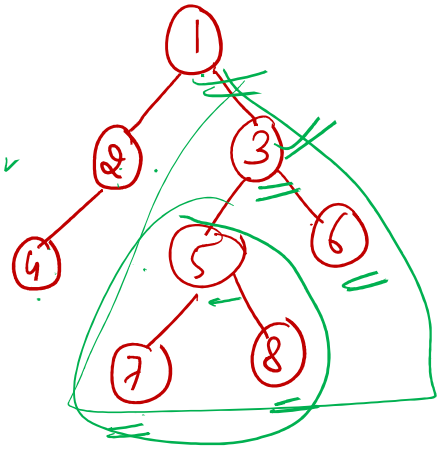


Pre : 1, 2, 4, 3, 5, 7, 8, 6

In : 4, 2, 1, 7, 5, 8, 3, 6

r root





Post : 4 2 7 8 5 6 3 1

Pre
In

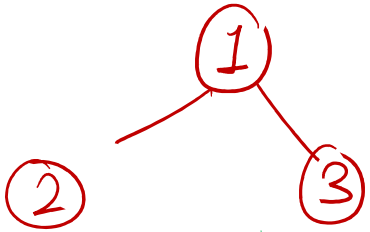
Post : 2, 3, 1 — L R Root

In : 2, 1, 3 — root

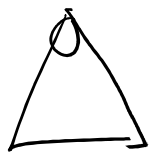
① Last will be the root node ; postorder

② root node position in inorder

③



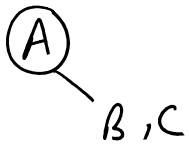
Post : L R Root

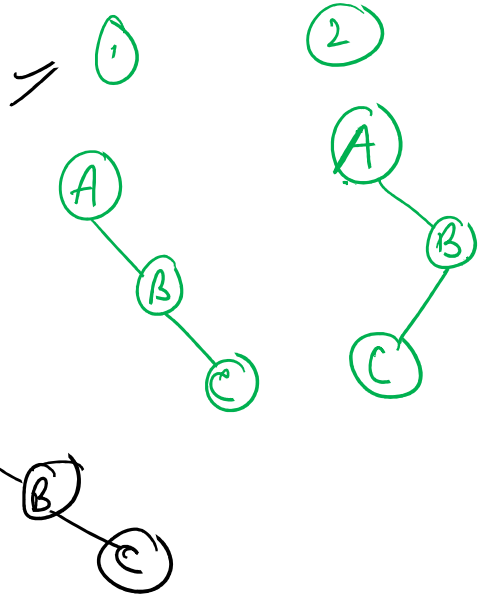


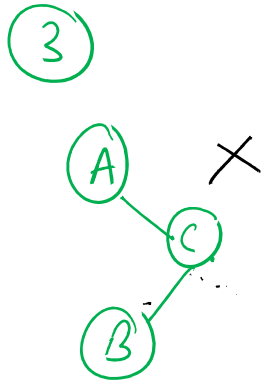
Post : C B A

In : A B C

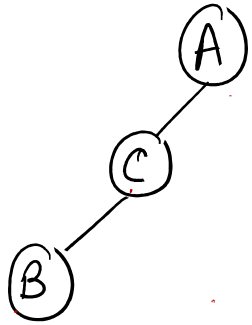
root



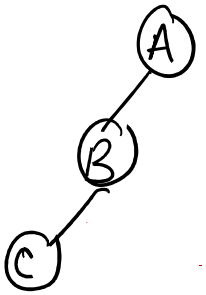




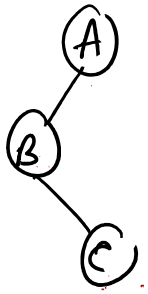
Post : C B A
In : B C A



In : B C A
Post : B C A

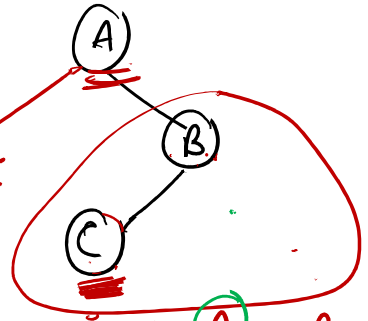


In : C B A



In : B C A
Post : C B A

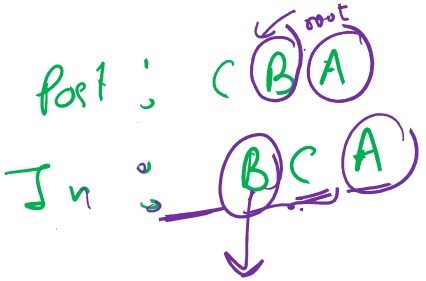
L Root R

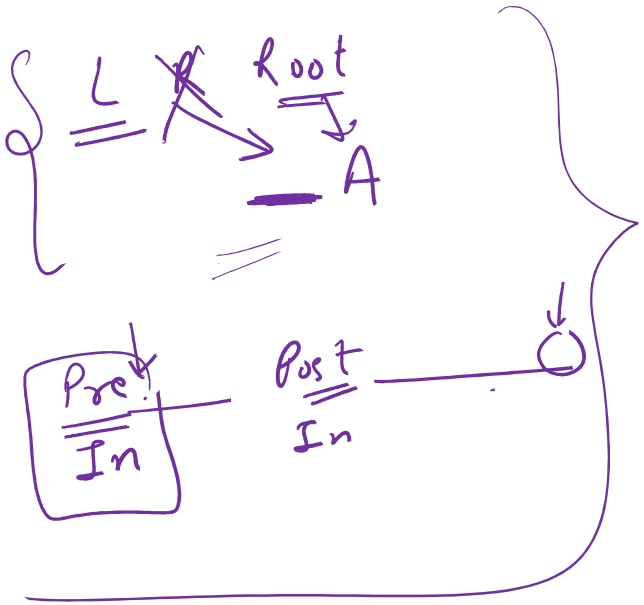


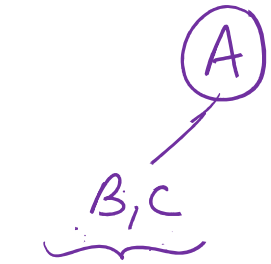
In : A C B
Post : C B A

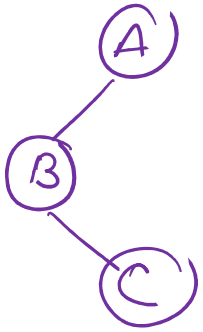
L R Root

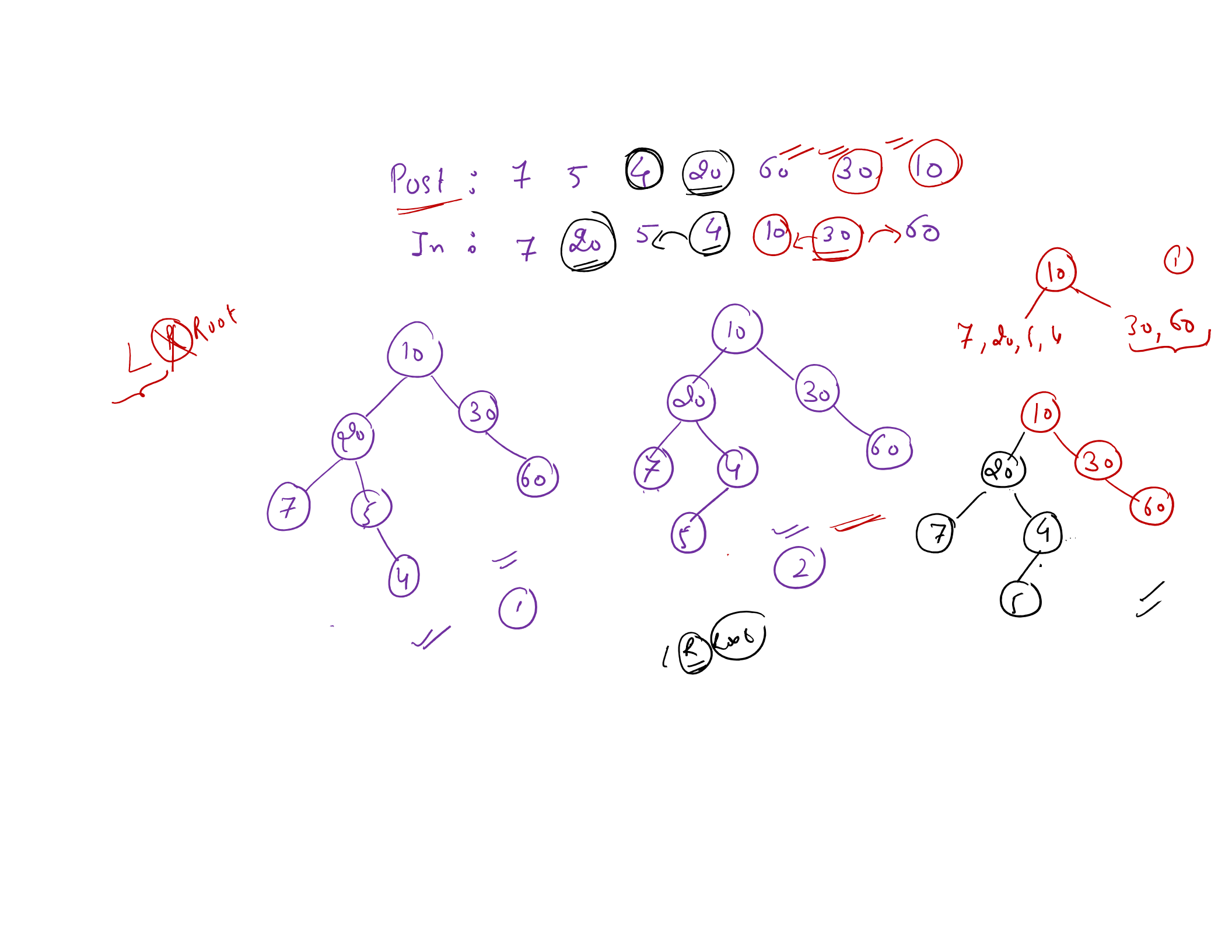
(1)


Post : C B A
root
In : B C A


L
Root
A
Pre
In
Post
In

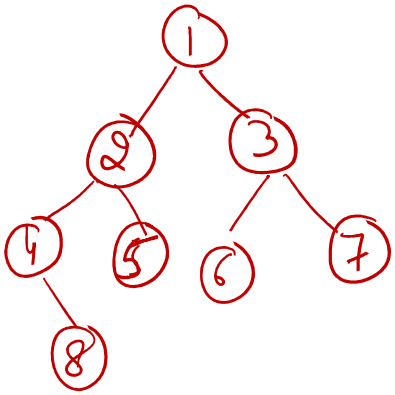

A
B, C

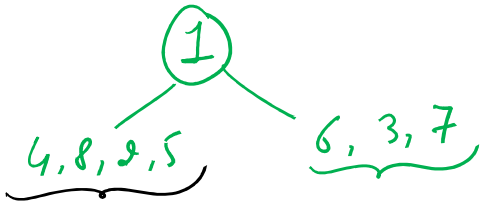

A
B
C

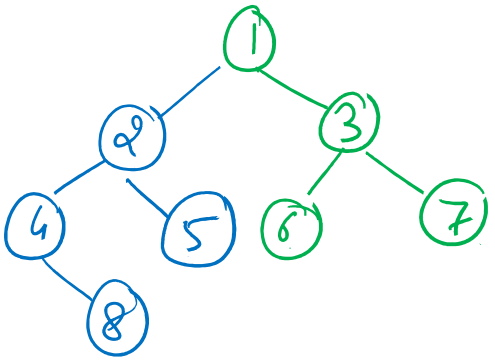

Post : 7 5 4 20 60 30 10
In : 7 20 5 4 10 30 60
Root
10
20 30
7 5 60
4
1
10
20 30
7 4 60
5
2
Root
10
7, 20, 5, 4
30, 60
1
10
20 30
7 4 60
5

Post : 8, 4, 5, 2, 6, 7, 3, 1

In : 4, 8, 2, 5, 1, 6, 3, 7







Pre
Post
In

Pre
In

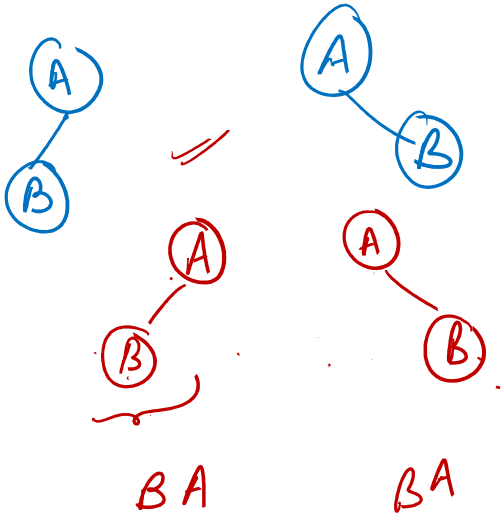
Post
In

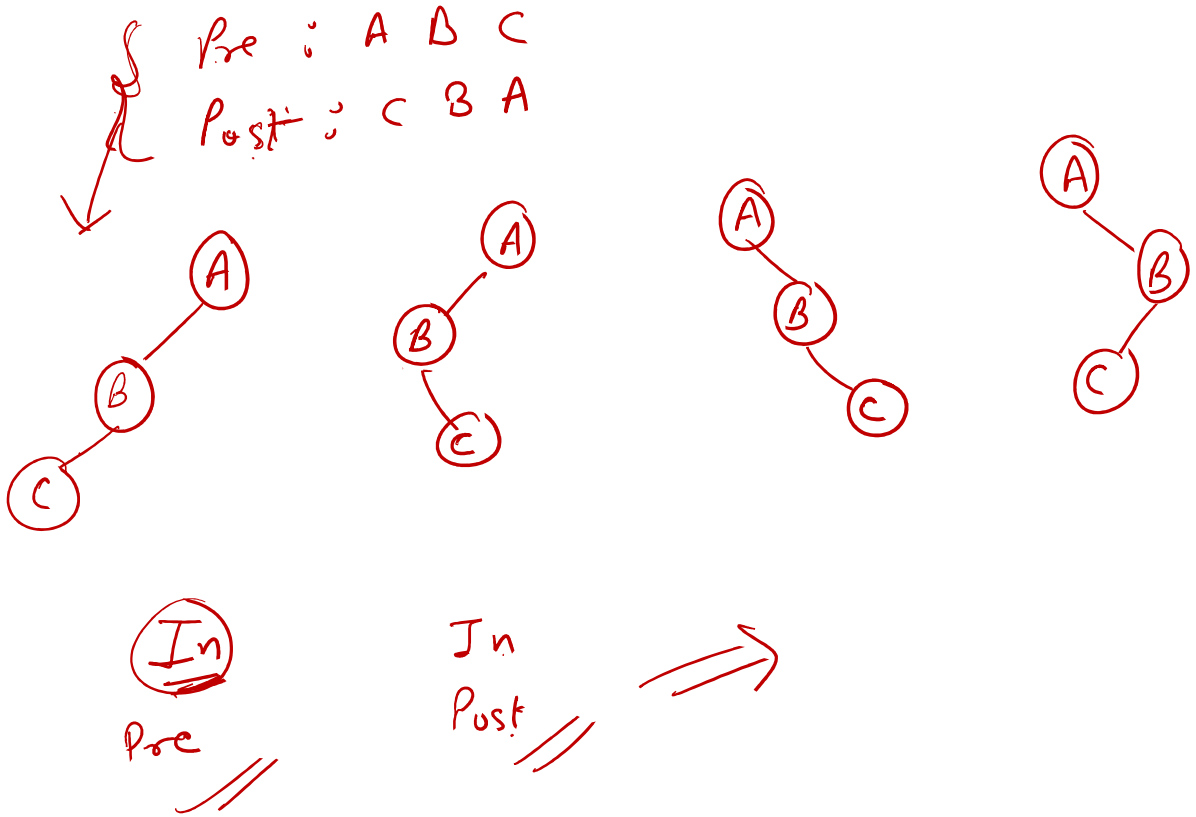
Pre
Post

Pre : A B

Post : B A

Root L R




Pre : A B C
Post : C B A
A
B
C
A
B
C
A
B
C
A
B
C
In
Pre
In
Post

Binary Tree

at most

0, 1, 2

- Preorder
- In
- Post

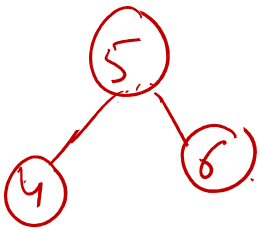
- Preorder
- In

- Post
- In
  - Pre
  - Post

Binary Tree

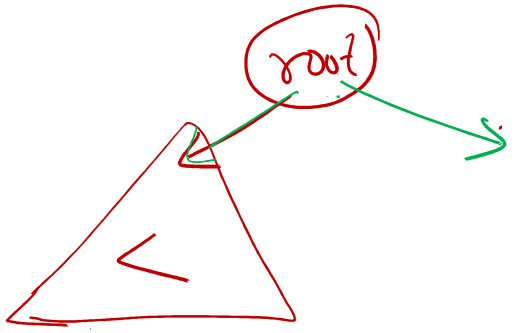
Binary Search Tree

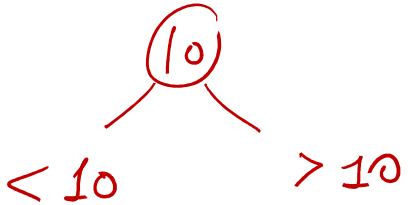
0, 1, 2



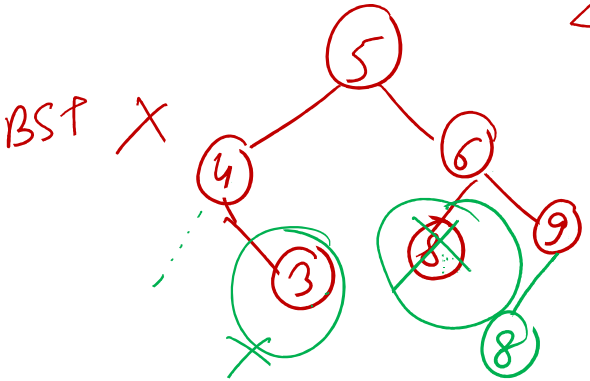
BT ~

BST ~

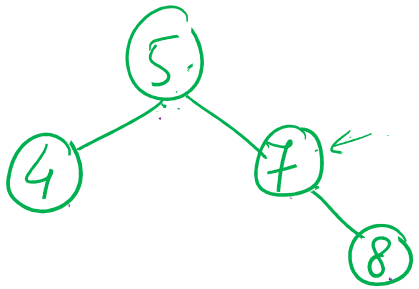




BST X



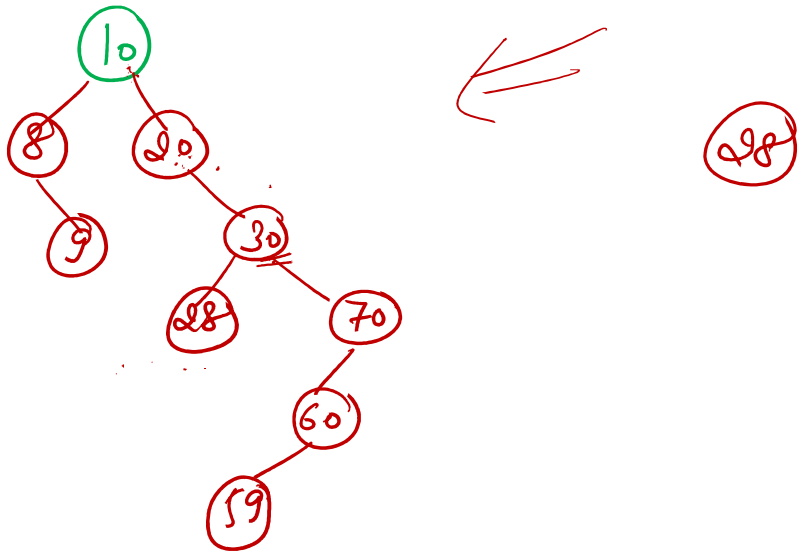
5, 4, 7, 8



in : 4 5 7 8

pre : 5, 4, 7, 8 Root L R

10, 20, 30, 70, 60, 8, 9, 59, 28



in : 8, 9, 10, 20, 28, 30, 59, 60, 70

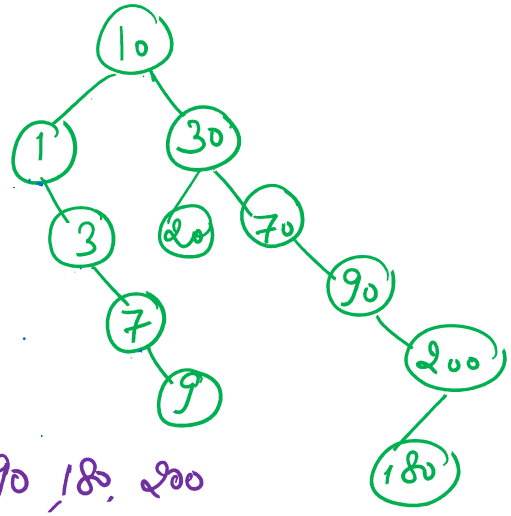
10, 30, 70, 90, 200, 180, 1, 3, 7, 9, 20

L Root R

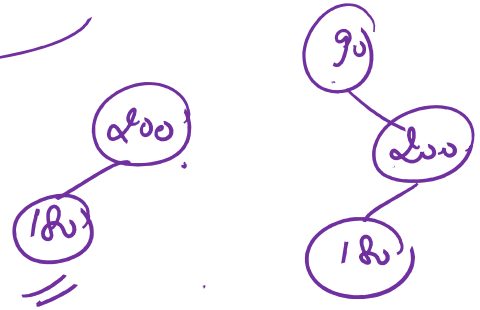
In: 1, 3, 7, 9, 10, 20,

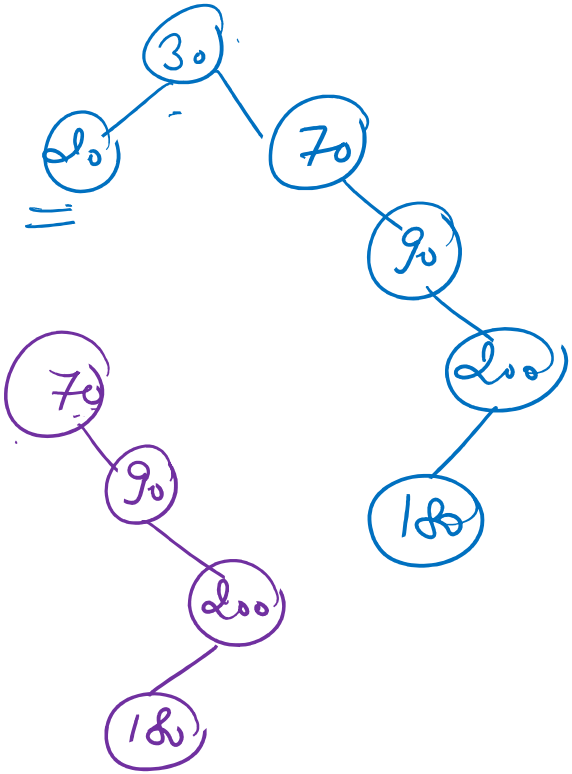
~~30, 180, 200, 90, 70~~

70, 90, 180, 200



1 3 7 9 10 20 30 70 90 180 200





BST ⇒ inorder : ascending order

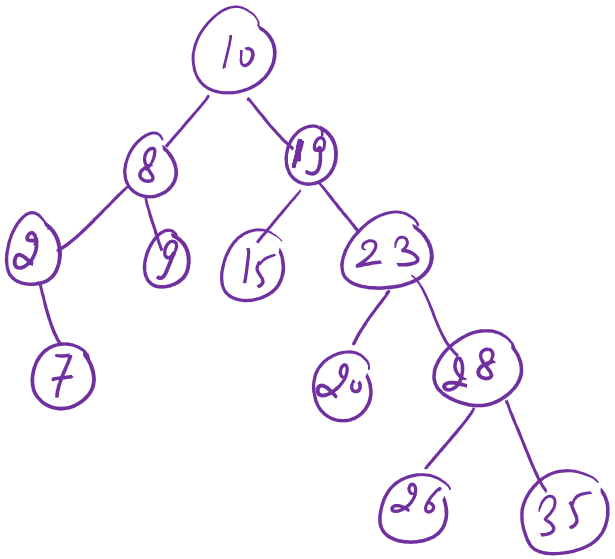
{ traversals

Pre
In
=
⟹

In:
Pre
Post

Pre : 5 4 7 8

In : 4 5 7 8



In : 2 7 8 9 10 15 19
20 23 26 28 35

Pre : 10, 8, 2, 7, 9, 19,
15, 23, 20, 28, 26, 35

Post : 7, 2, 9, 8, 15, 20, 26, 35,
28, 23, 19, 10

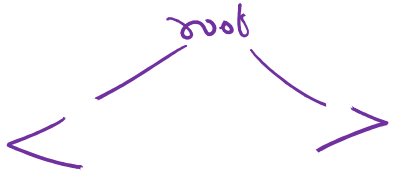
Binary Tree

Pre
In

Post
In

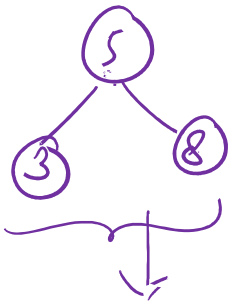
Pre
Post

Binary Search Tree



5, 8, 3

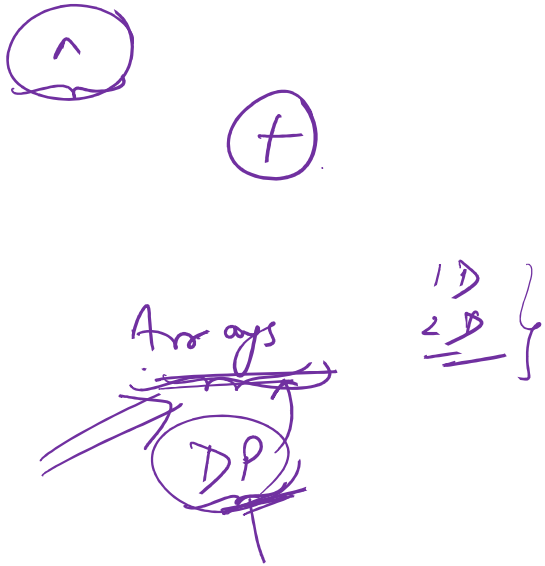
Construction
Insertion



Pre
In → ascending order
Post

Pre
In

Post
In

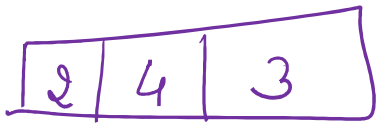

^
+
Arrays
DP
1D
2D

Array a : size $n-1$ (3) $n = 4$

$[1 - n]$

$[1 - 4]$

1, 2, 3, 4



(1)

Array a $n-1$

integ values $[1 - n]$

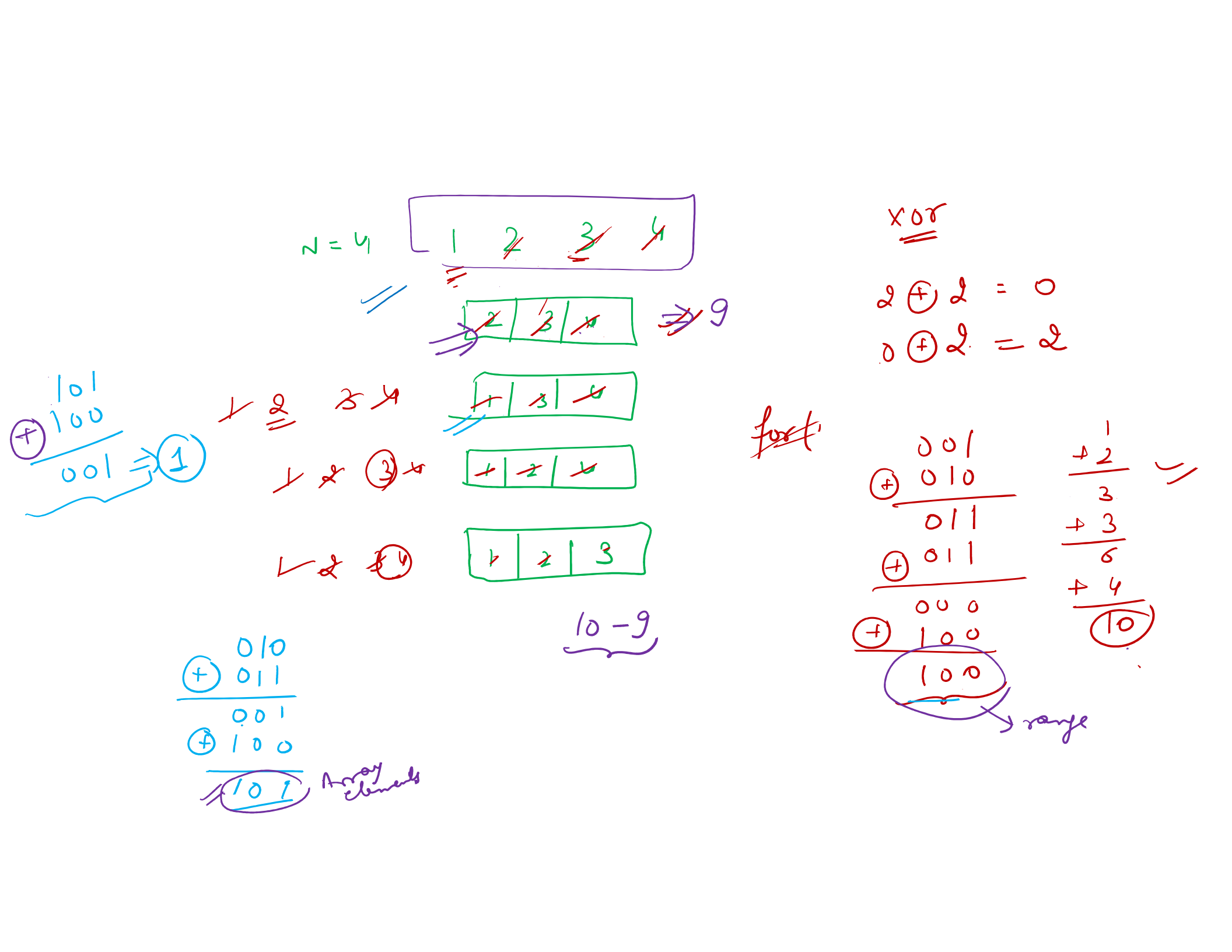
$1 - n$

```
for (i=1 ; i<=n ; i++)
    sum = sum + i ;
```

$1+2+3+4+5$

$n = 5$

```
for (i = 1 ; i < n ; i++)        (4)
    sum = sum - a[i];
```


N = 4
1 2 3 4
9
101
⊕ 100
001 ⇒ 1
1 ⊕ 2 ⊕ 3 ⊕ 4
1 ⊕ 3 ⊕ 4
1 ⊕ 2 ⊕ 4
1 ⊕ 2 ⊕ 3
10 - 9
010
⊕ 011
001
⊕ 100
101
Array Elements
XOR
2 ⊕ 2 = 0
0 ⊕ 2 = 2
001
⊕ 010
011
⊕ 011
000
⊕ 100
100
range
1
+ 2
3
+ 3
6
+ 4
10

```
// for ( int i = 1 ; i <= n ; i++)
    Sum = Sum + i ;

for ( int j = 1 ; j <= n ; j++)
    Sum1 = Sum1 + a[j];

Sum = Sum - Sum1;
```

O(n) + O(n) + O(1) → O(n)

XOR

```
for ( int i = 1 ; i <= n ; i++)
    Sum = Sum ^ i ;

for ( int j = 1 ; j <= n ; j++)
    Sum1 = Sum1 ^ a[j];

Sum = Sum ^ Sum1 ;
```

O(n) + O(n) + O(1) → O(n)

Trees

↓

NLDS

Root
Leaf
Parent
Child
Siblings

Traversal = Insertion Deletion → Binary Tree

at most 2

0, 1, 2

Preorder ⇒ Root L R

Inorder ⇒ L Root R

Postorder ⇒ L R Root

Pre
In

Post
In

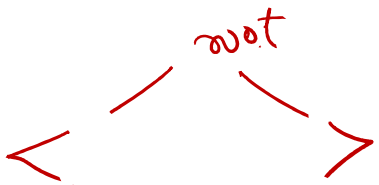
Pre
Post

Binary Search Tree (BST)

↓

BT at most 2 children

0, 1, 2



Insertion

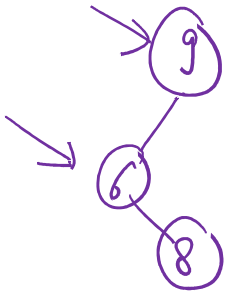
Deletion

Traversal

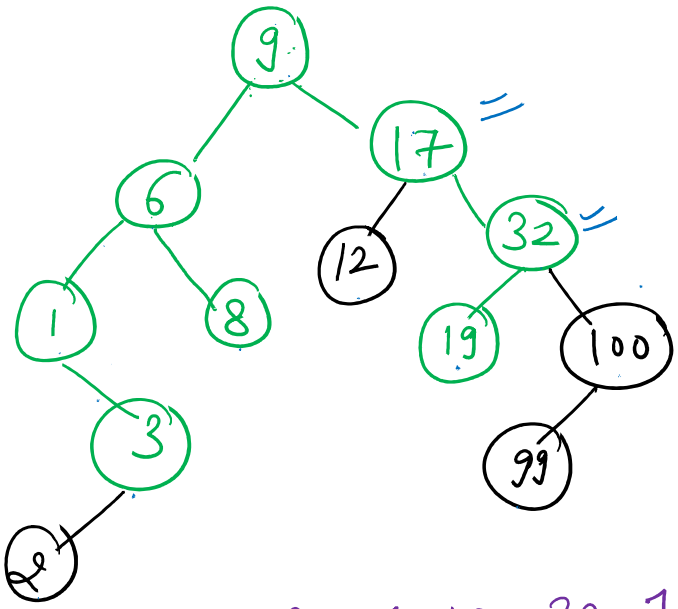
Construction

root

9, 6, 8, 17, 32, 19, 1, 3



Traversal



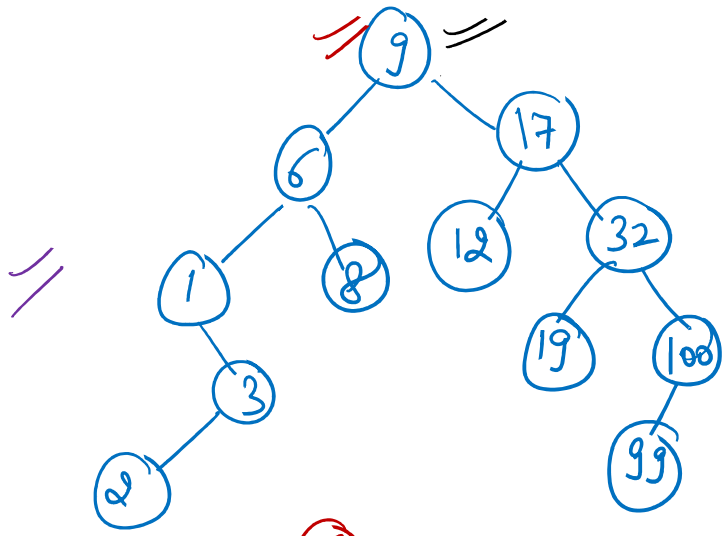
Pre : 9 6 1 3 2 8 17 12 32 19 100 99

Insert 100, 99, 2, 12

Root L R

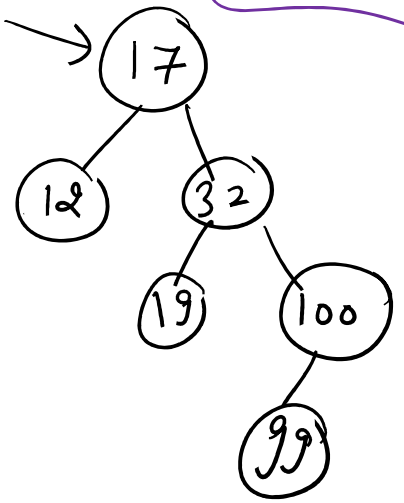
Pre : 3 2 1 6 8 9 12 17 19 32 99 100

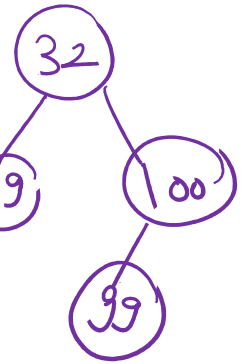
In : L Root R

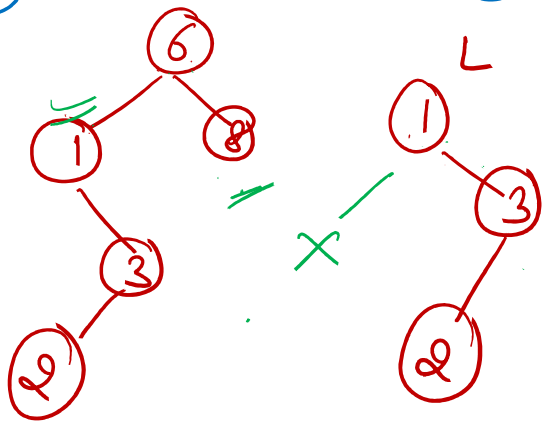


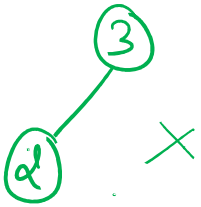
Pre : Root L R

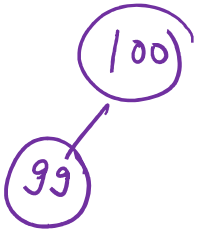
9 6 1 3 2 8 17 12 32 19 100 99

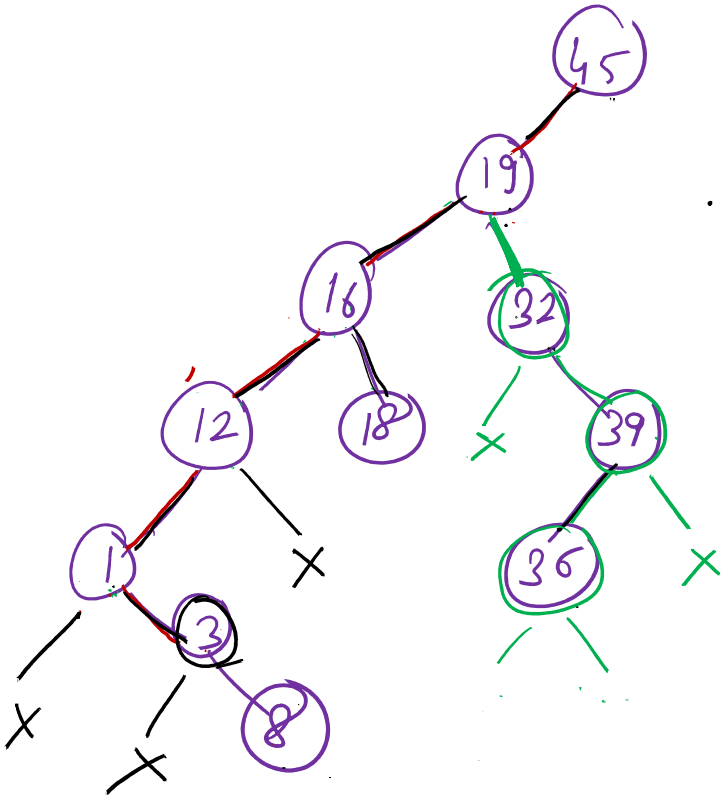












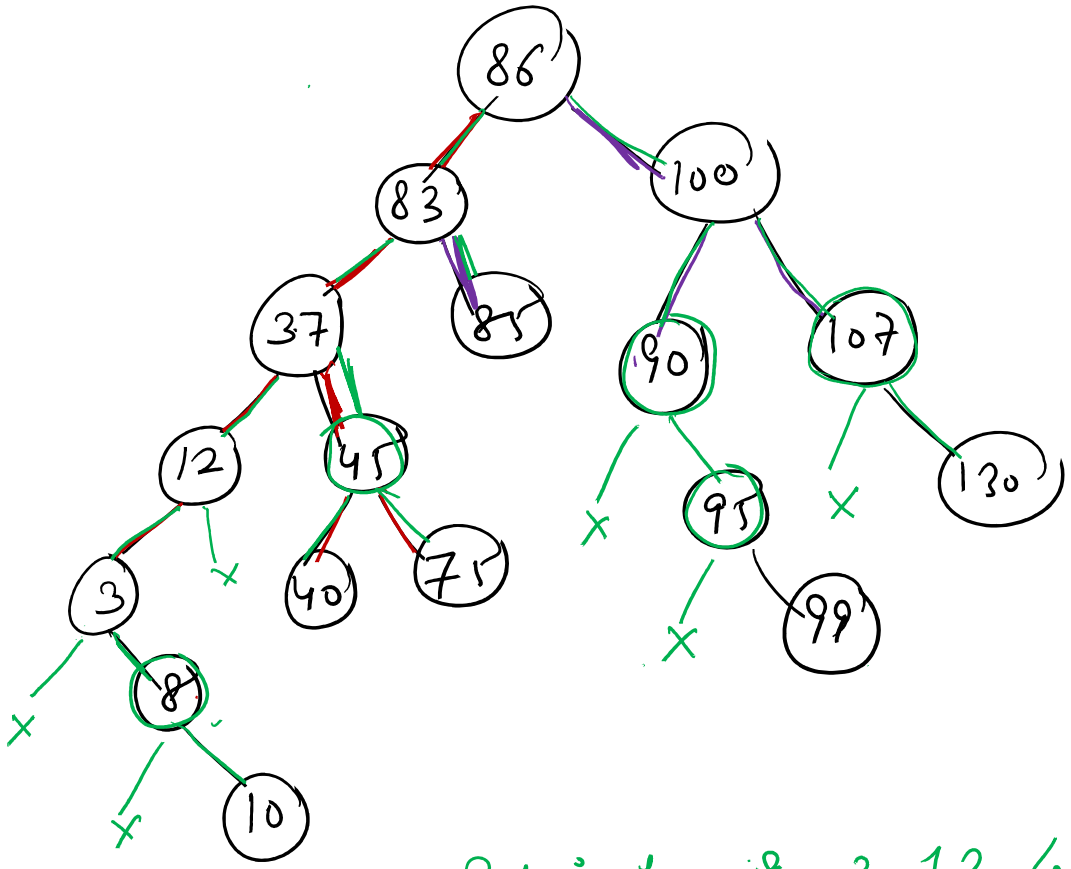
Pre : Root L R

45 19 16 12 1 3 8 18

32 39 36

Post : 8 3 1 12 18 16 36 39

32 19 45



Pre

86 83 37 12 3 8 10

45 40 ~~75~~ 85

100 90 95 99

107 130

in : 3 8 10 12 37 40 45

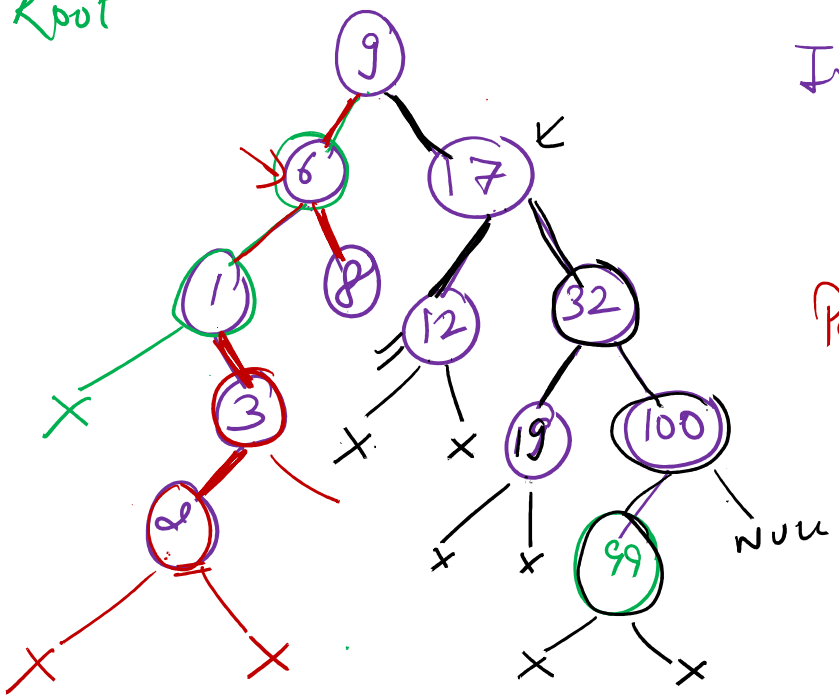
~~75~~ 83 85 ~~86~~ 90 95

99 100 107 130

Pst : 10 8 3 12 40 75 45 37 85 83 99 95

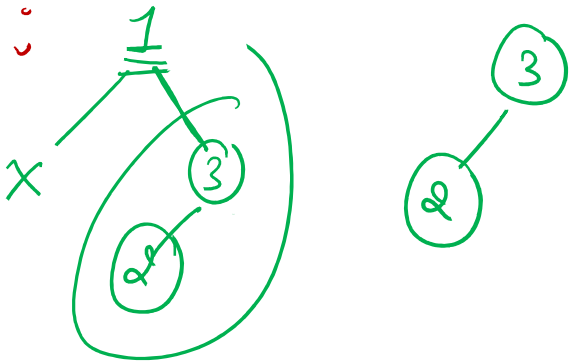
90 130 107 100 86

L R Root



In: 1 2 3 6 8 9 12 17 19 32 99 100

Post: 1 3 2



2 3 1 8 6 12 19 99 100 32 17 9

Insertion

Traversal — Pre, Post, In

BST

Root L R Pre : 40, 30, 25, 55, 50, 45, 60

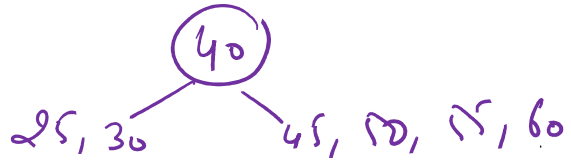
L Root R In : 25, 30, 40, 45, 50, 55, 60

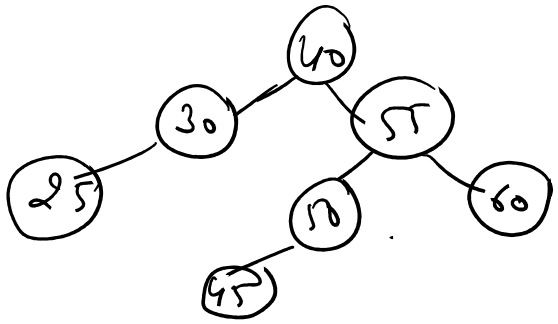
BT

preorder

Inorder

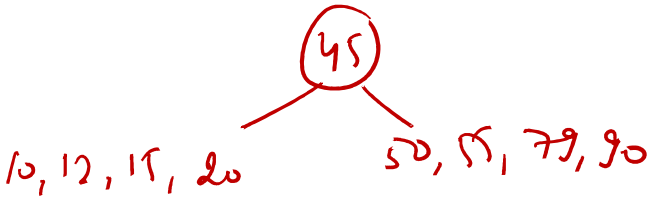
(1)

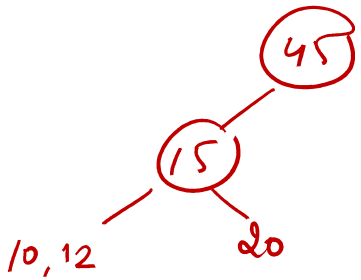


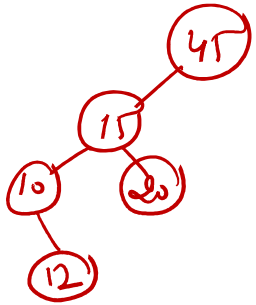


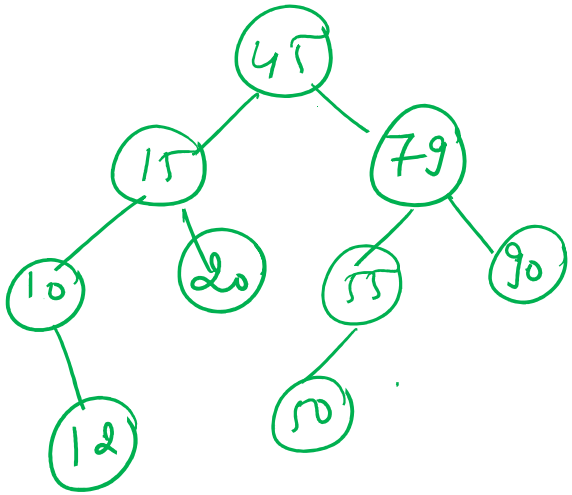
Pre : 45, 15, 10, 12, 20, 79, 55, 50, 90

In : 10, 12, 15, 20, 45, 50, 55, 79, 90





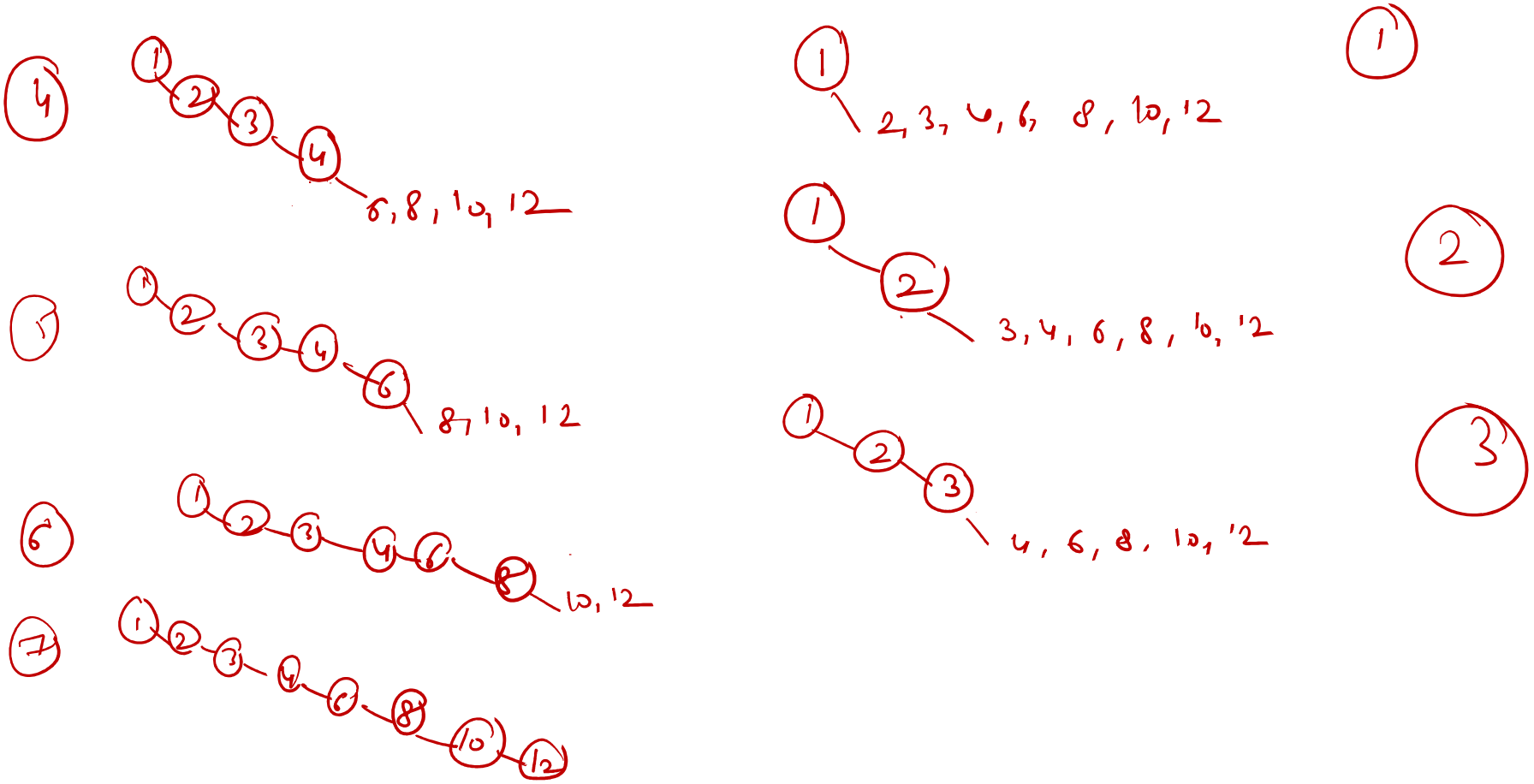


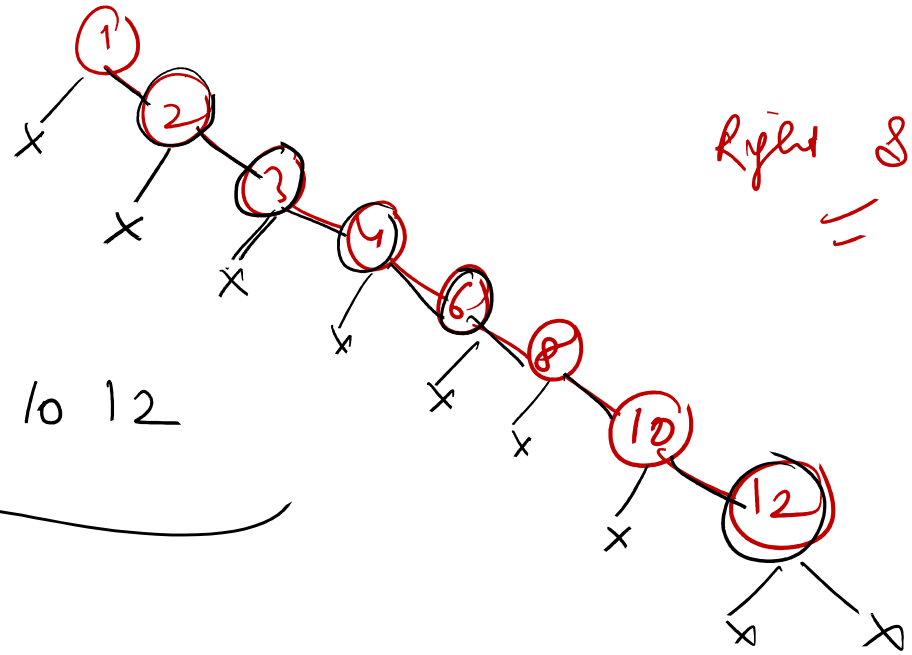


Pre

L R Root Post : 12 10 8 6 4 3 2 1

L Root R IN : 1 2 3 4 6 8 10 12





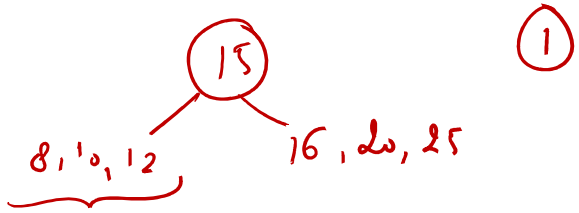
Right Skew free

Root L R

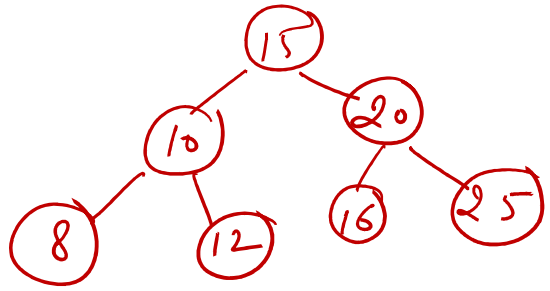
Pre : 1 2 3 4 6 8 10 12

Post : 8, 12, 10, 16, 25, 20, 15

In : 8, 10, 12, 15, 16, 20, 25



1



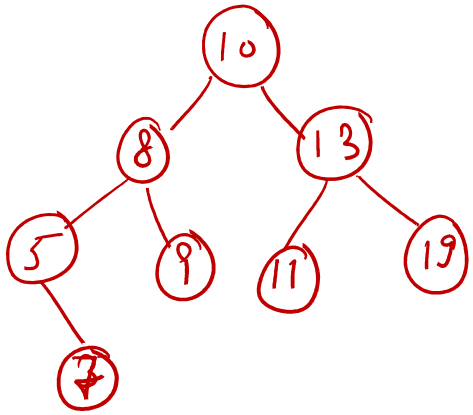
BST

- Insertion
- Traversal - Pre
  - Post
  - In

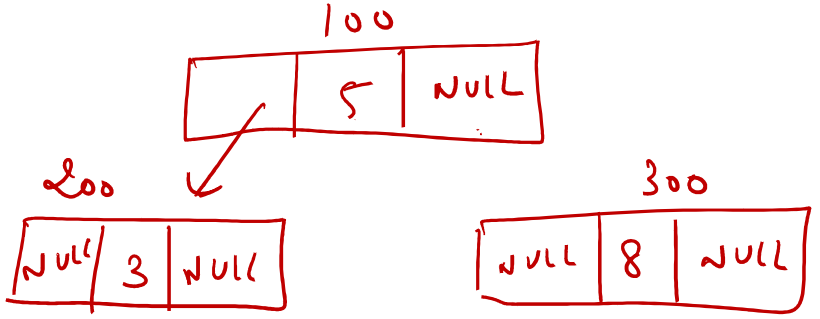
Pre →

Post →

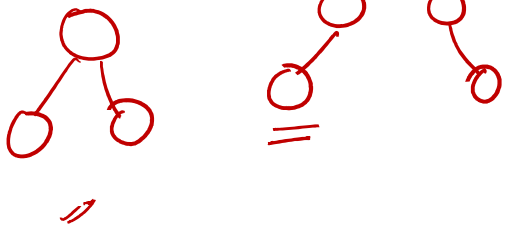
Deletion ??

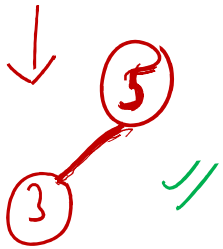


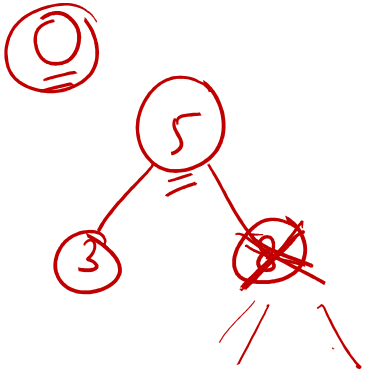
Deletion

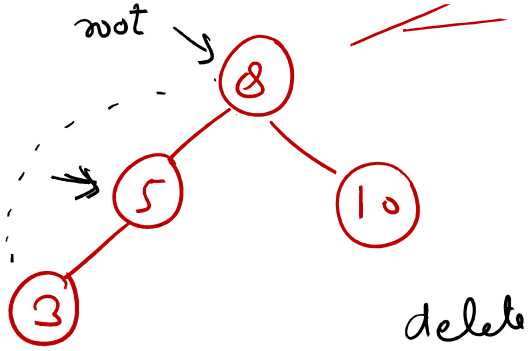


BST





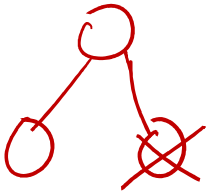


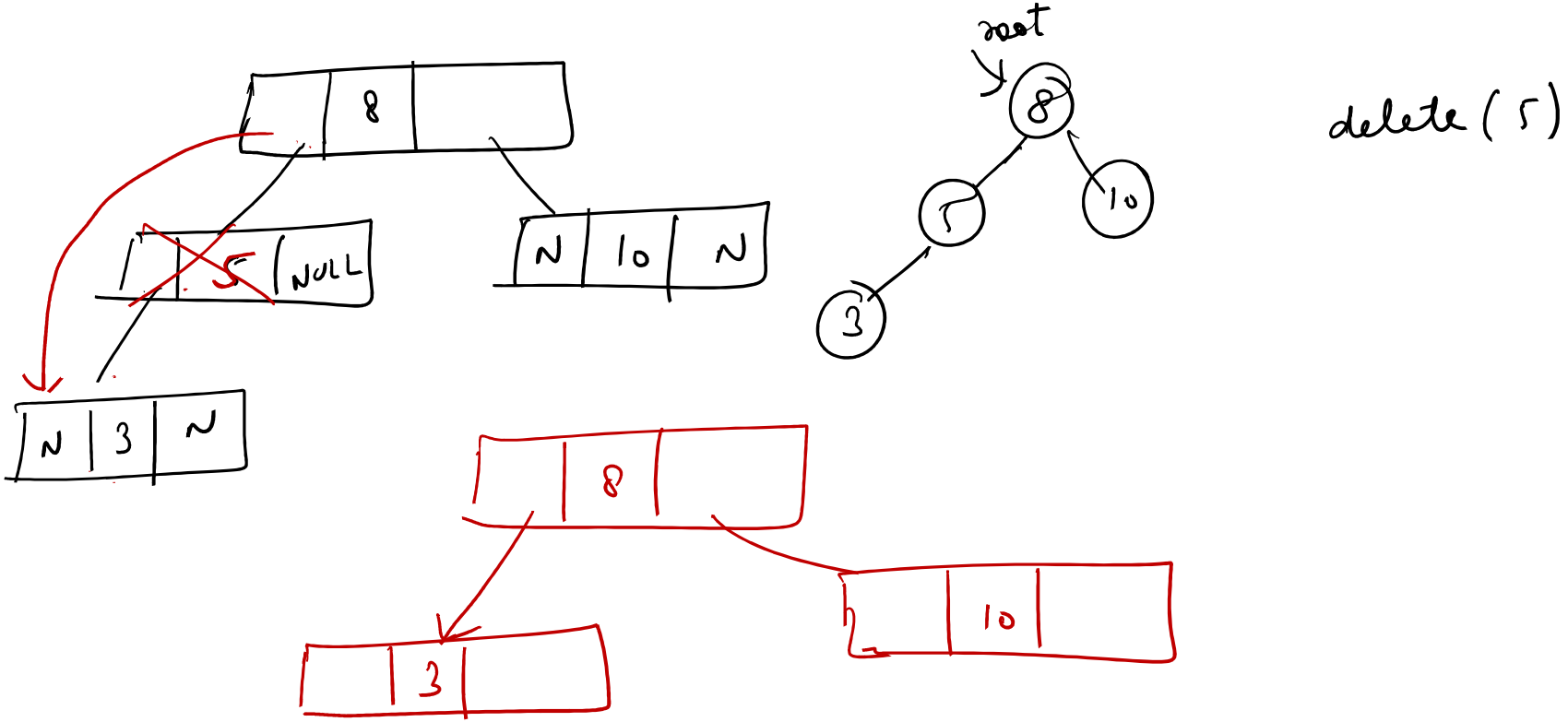


delete(5)

Deletion

1) when a node has 0 child :

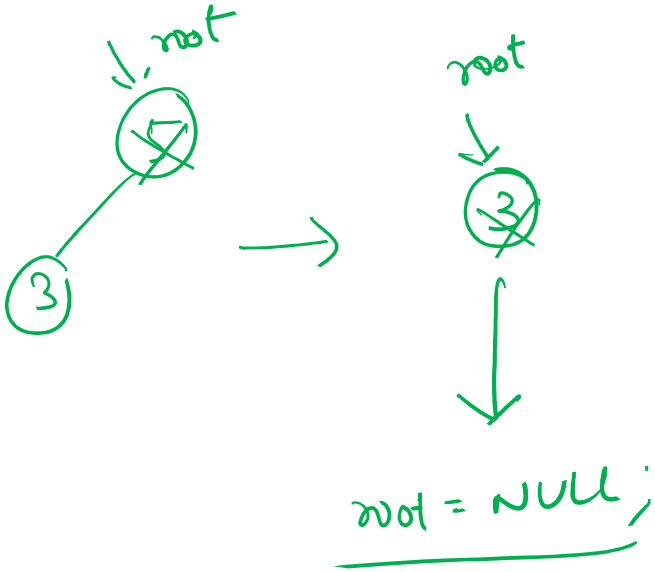


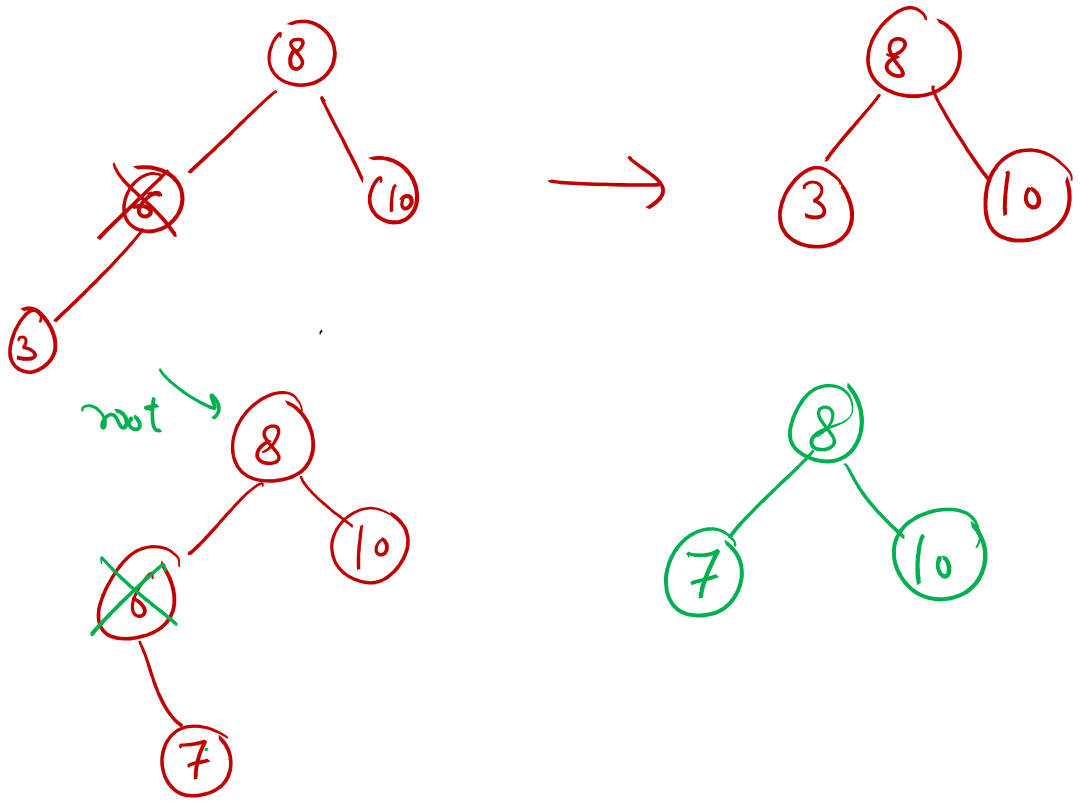


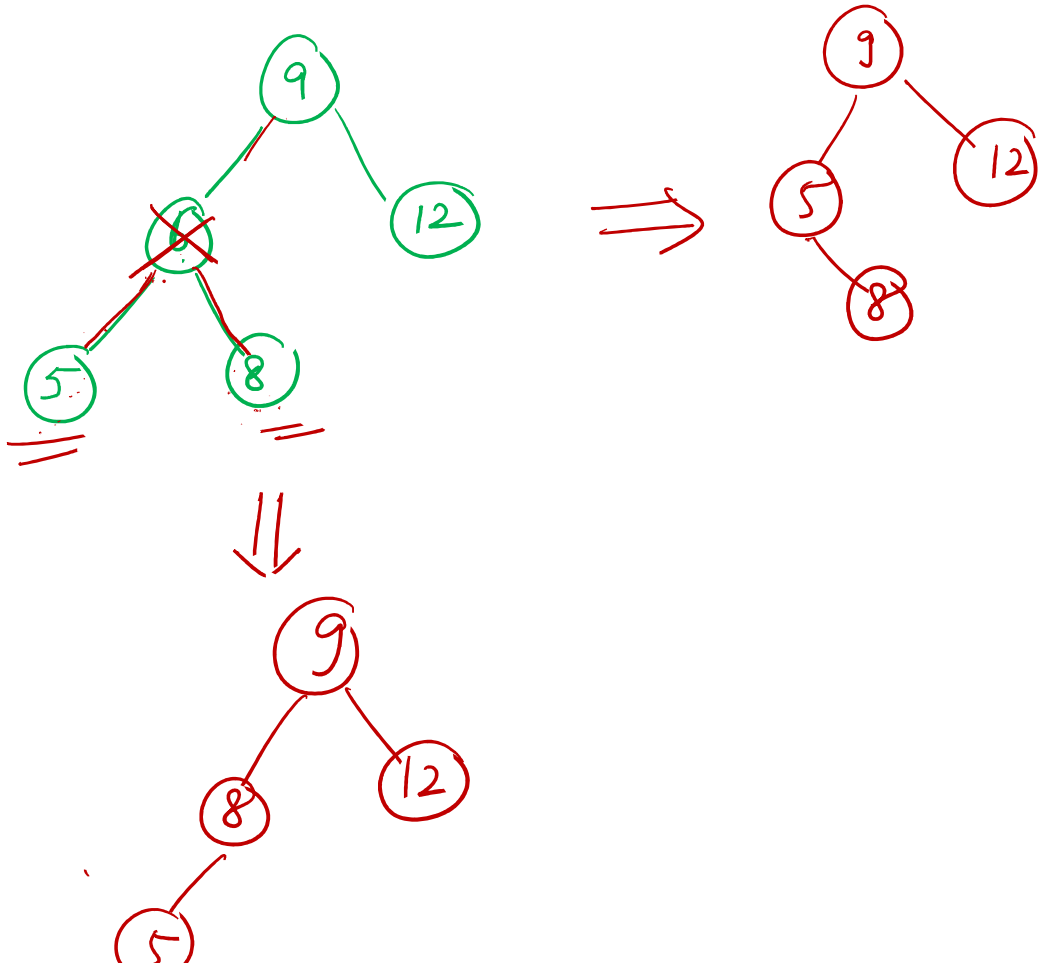
delete (5)

0, 1, 2

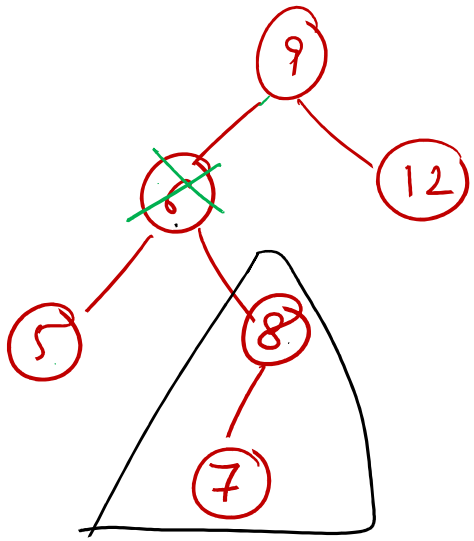
1 child

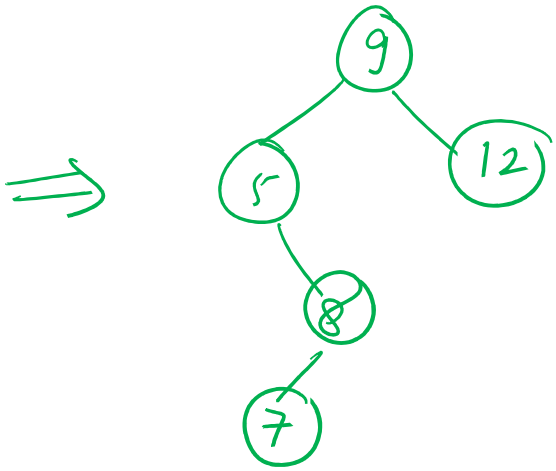





9
6
12
5
8
9
5
12
8
9
8
12
5

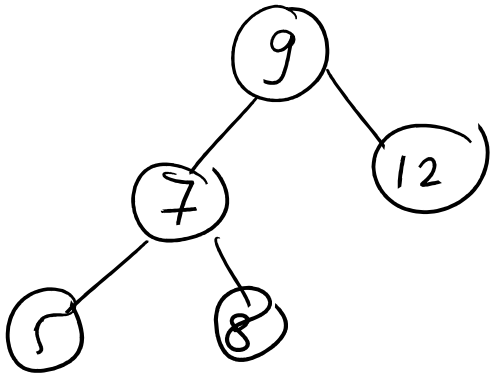
inorder successor

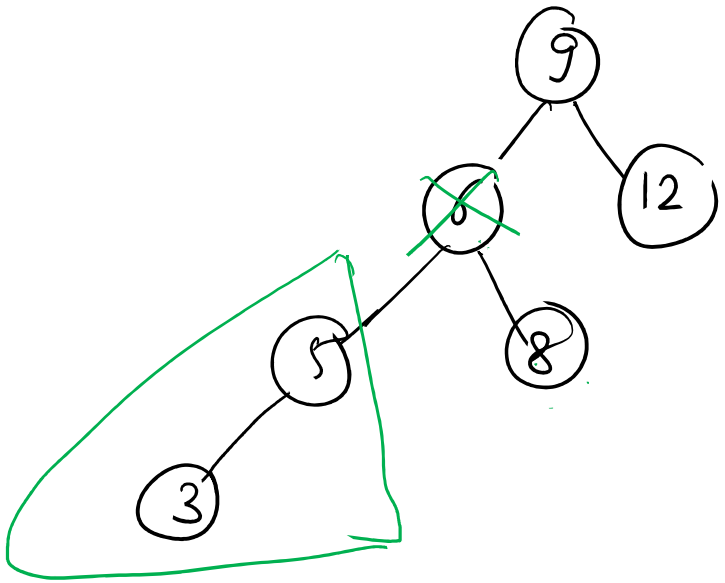


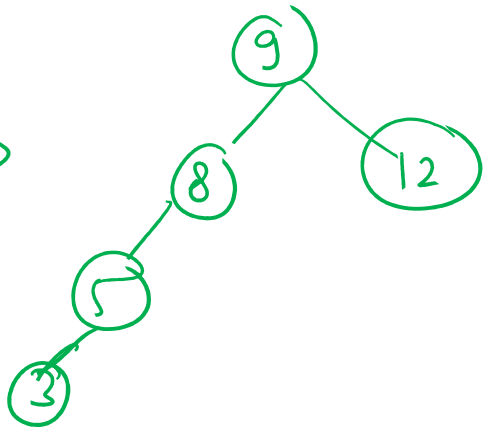


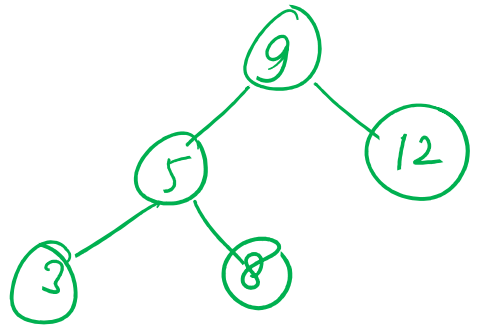
In: 5 6 7 8 9 12

Smallest value of right subtree







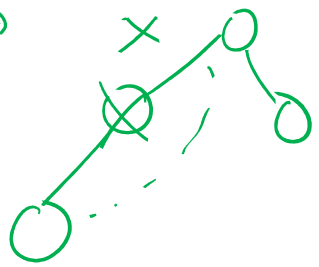


inorder predecessor

3 5 6 8 9 12

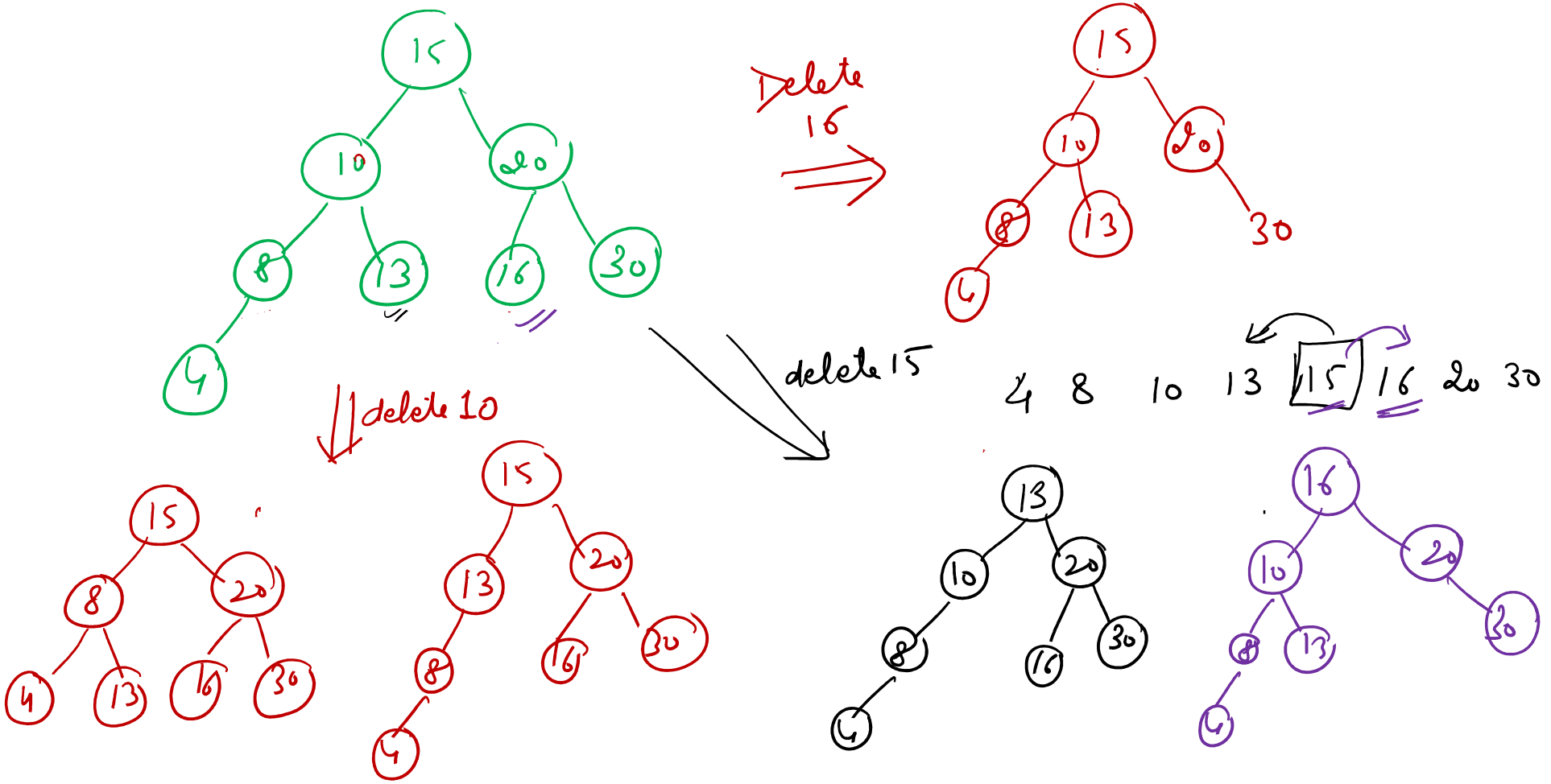
largest value of left subtree

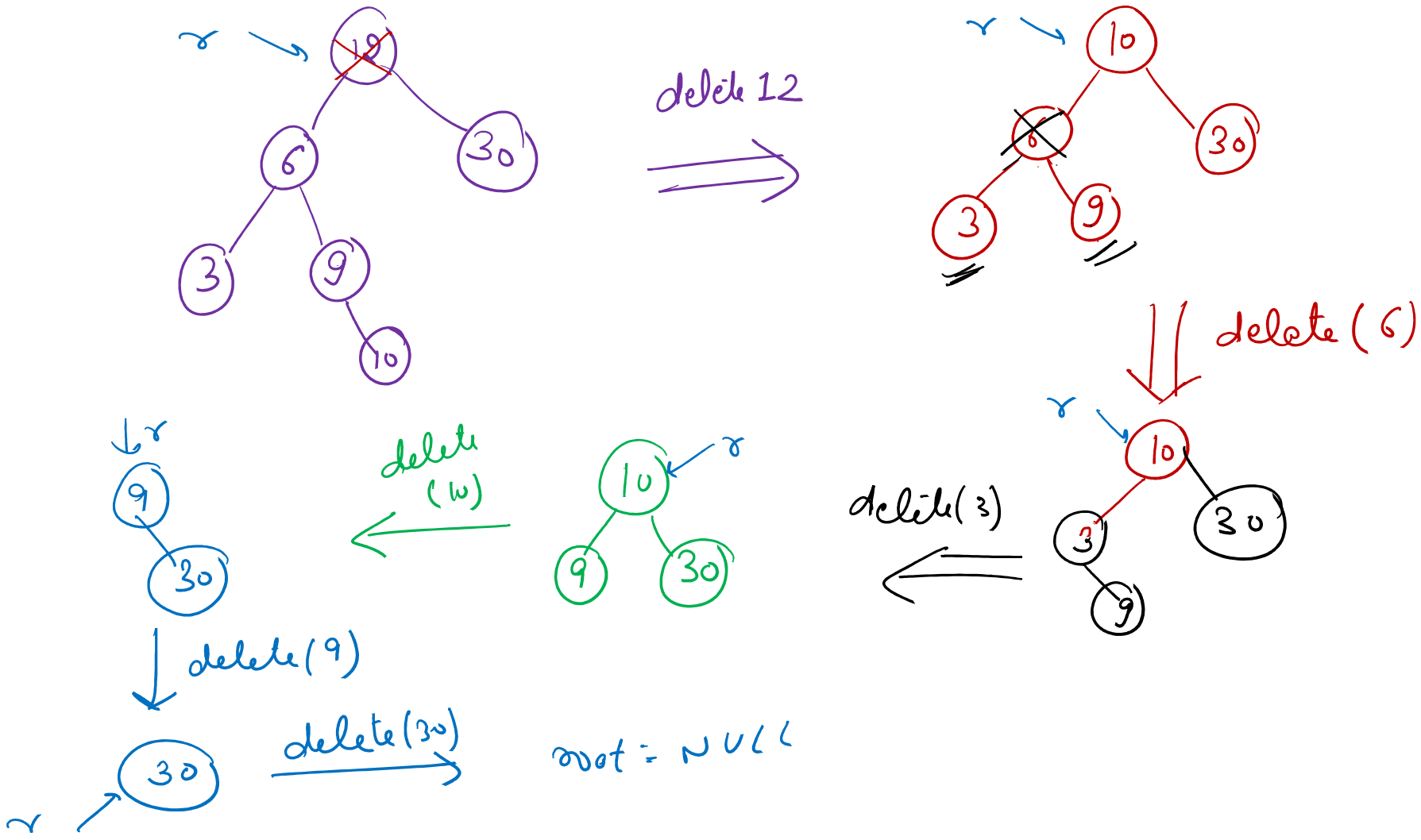
0 child ⇒

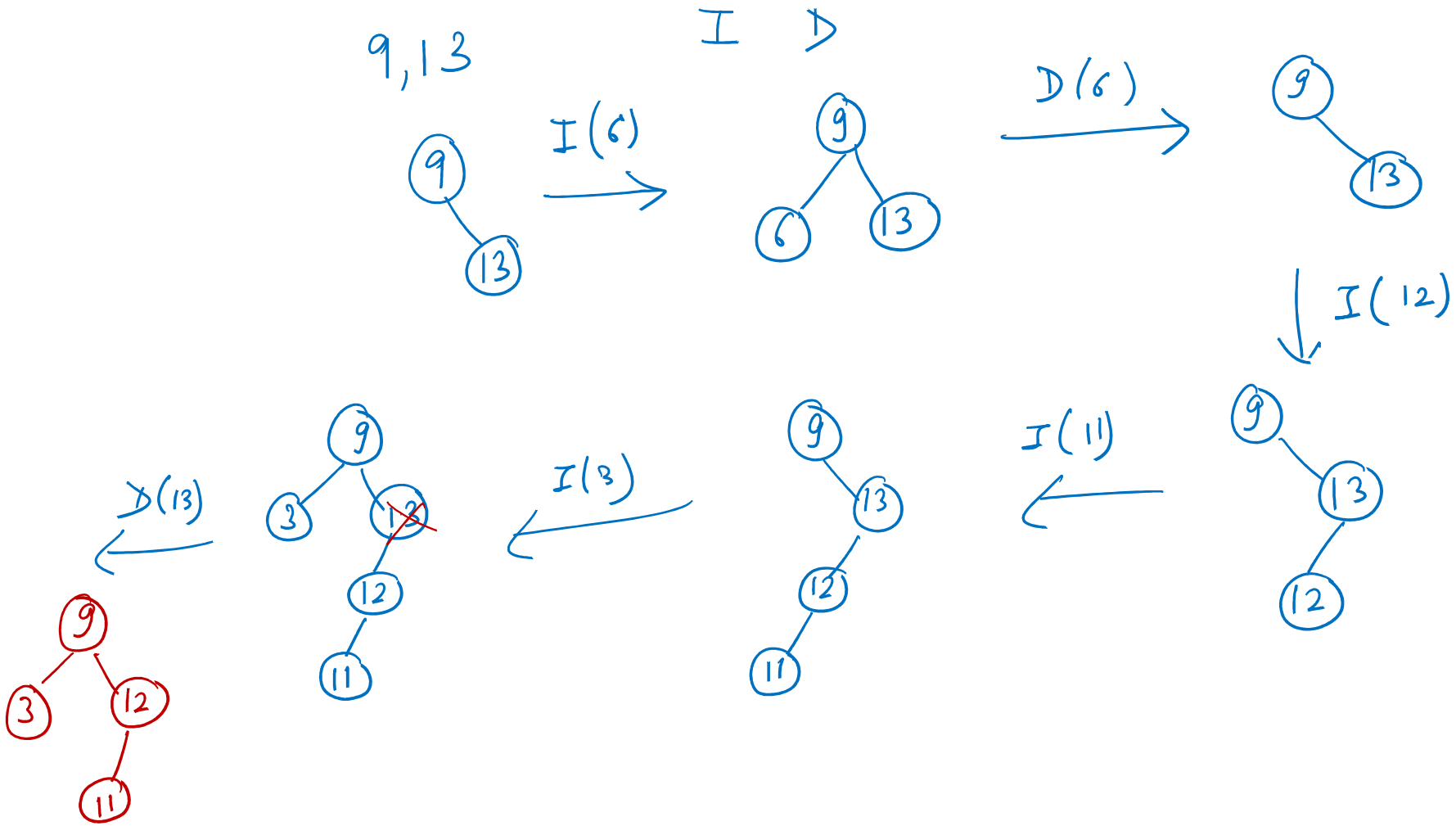


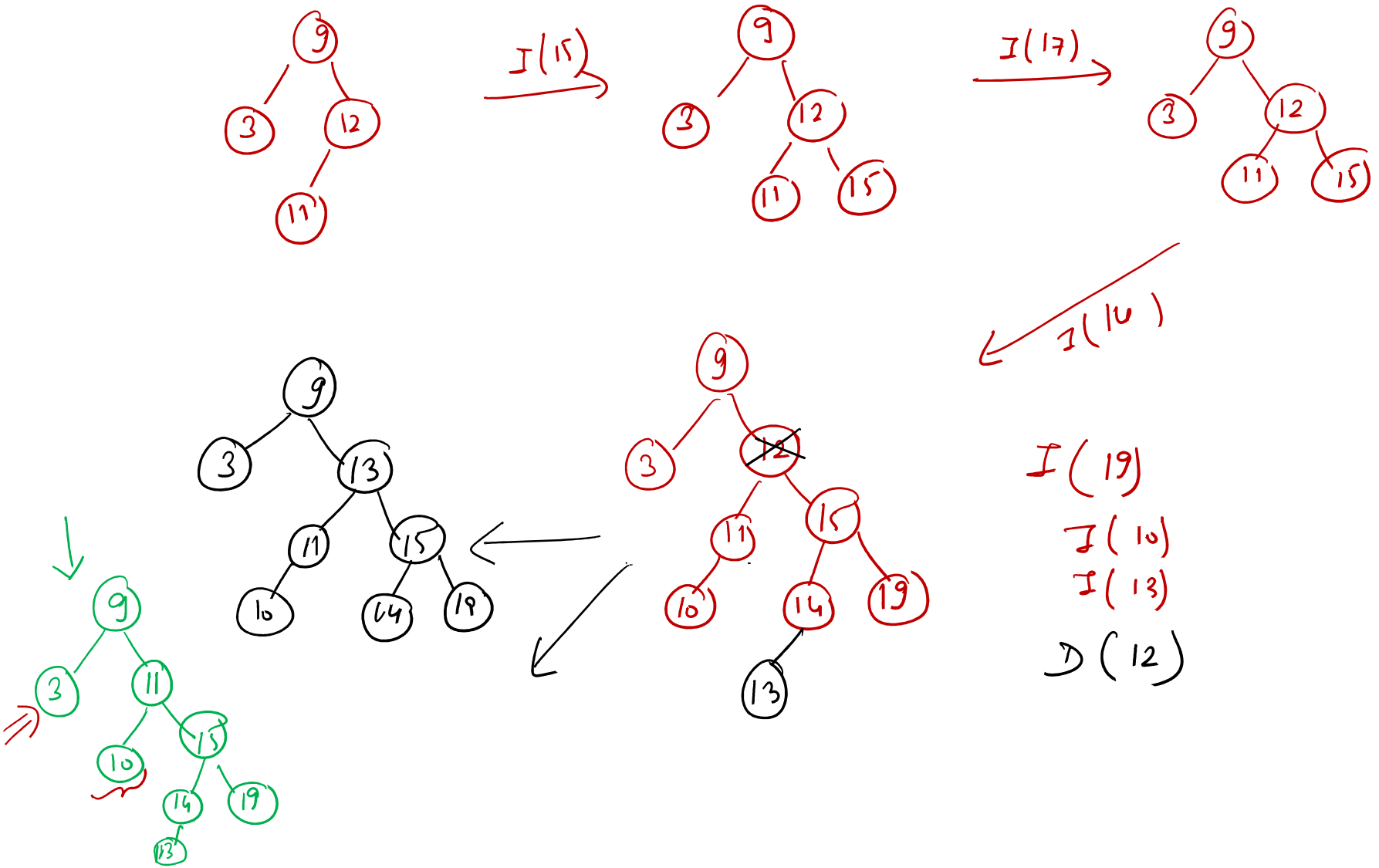
1

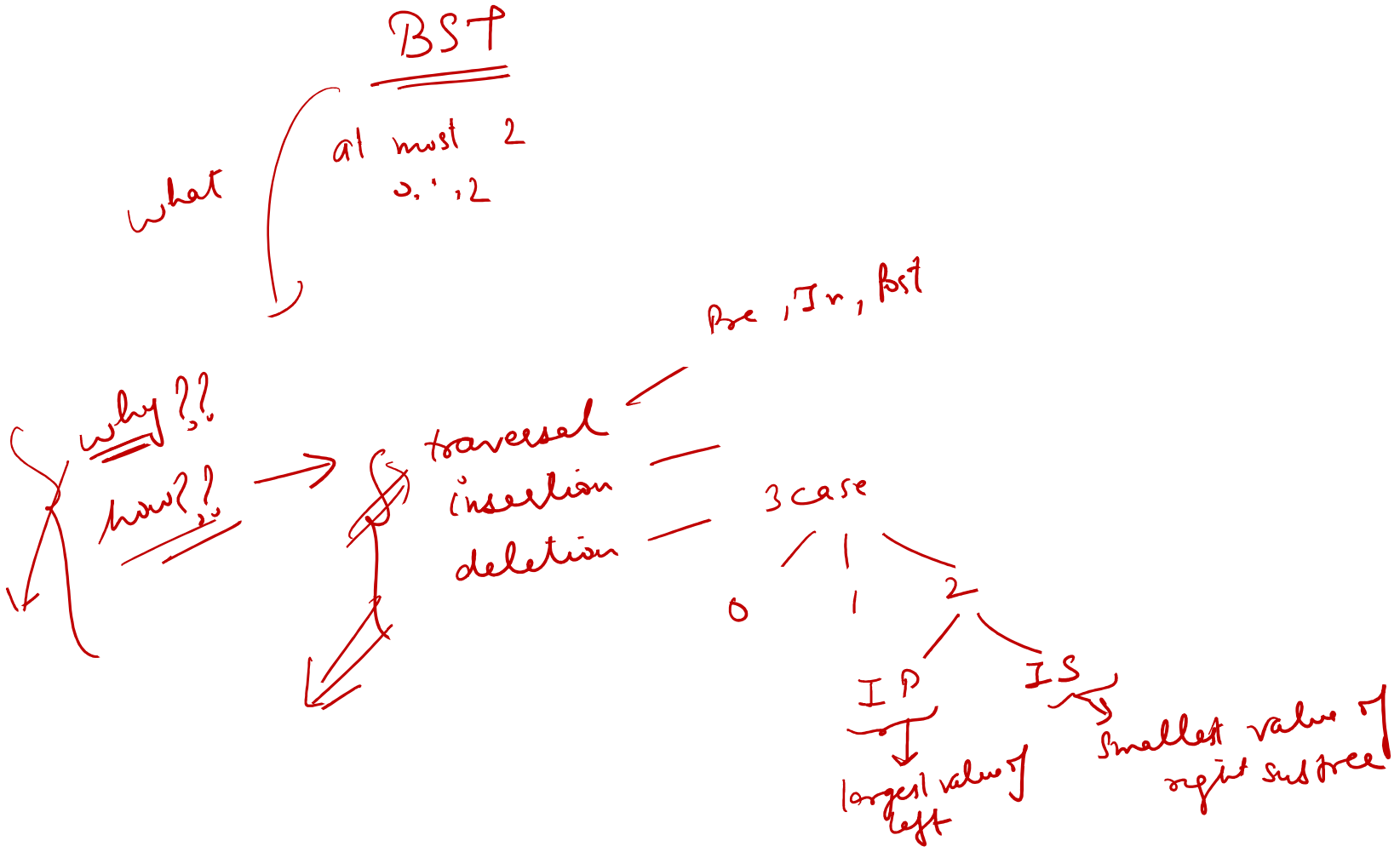
2 children .. inorder predecessor ; inorder successor


15
10
20
8
13
16
30
4
Delete 16
15
10
20
8
13
30
4
delete 15
4 8 10 13 15 16 20 30
delete 10
15
8
20
4
13
16
30
15
13
20
8
16
30
4
16
10
20
8
13
30
4

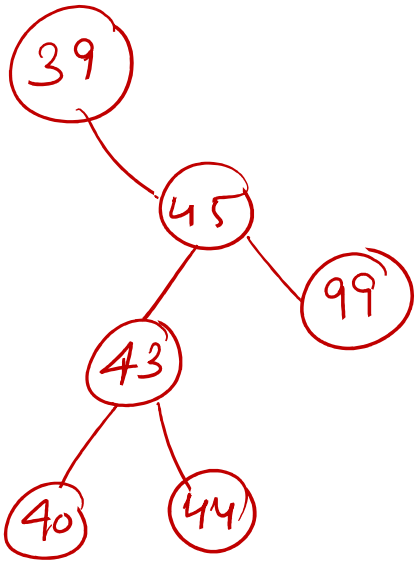

r
12
6
30
3
9
10
delete 12
r
10
6
30
3
9
delete (6)
r
10
3
30
9
delete(3)
r
10
9
30
delete (10)
r
9
30
delete(9)
30
r
delete(30)
root = NULL


9,13
I D
I(6)
D(6)
I(12)
I(11)
I(3)
D(13)


I(15)
I(17)
I(10)
I(19)
I(10)
I(13)
D(12)


BST
what
at most 2
0, 1, 2
why ??
how ??
traversal
insertion
deletion
Pre, In, Post
3 case
0
1
2
IP
IS
largest value
left
Smallest value
right subtree

1

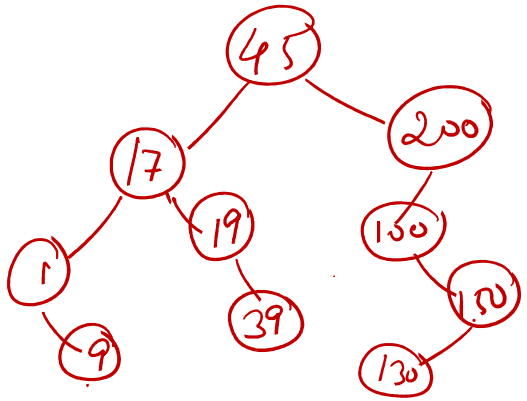


Pre

In

Post

2



Pre

In

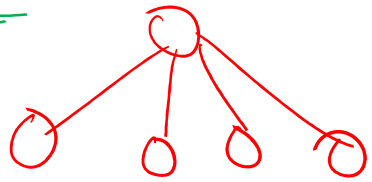
Post

Insert : 8, 119, 50, 85, 230

Deletion : 19, 45, 17, 100, 230

General Trees

non linear DS

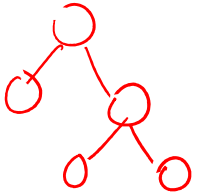


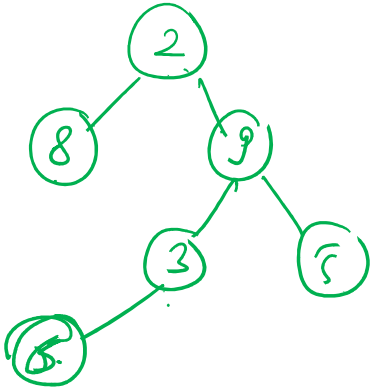
Binary Search Tree

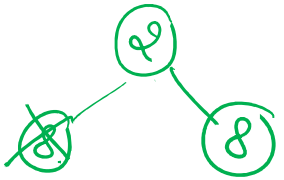
- Binary Tree → at most 2 children
- Binary Search Tree → 
  - AVL Tree
    - R B
    - B
    - B+

BT

0, 1, 2







BST

every BST can be considered as BT ✓
every BT BST ✗

what?! Tree
BT
BST ⇒ compression
sorted/ordered, organized data

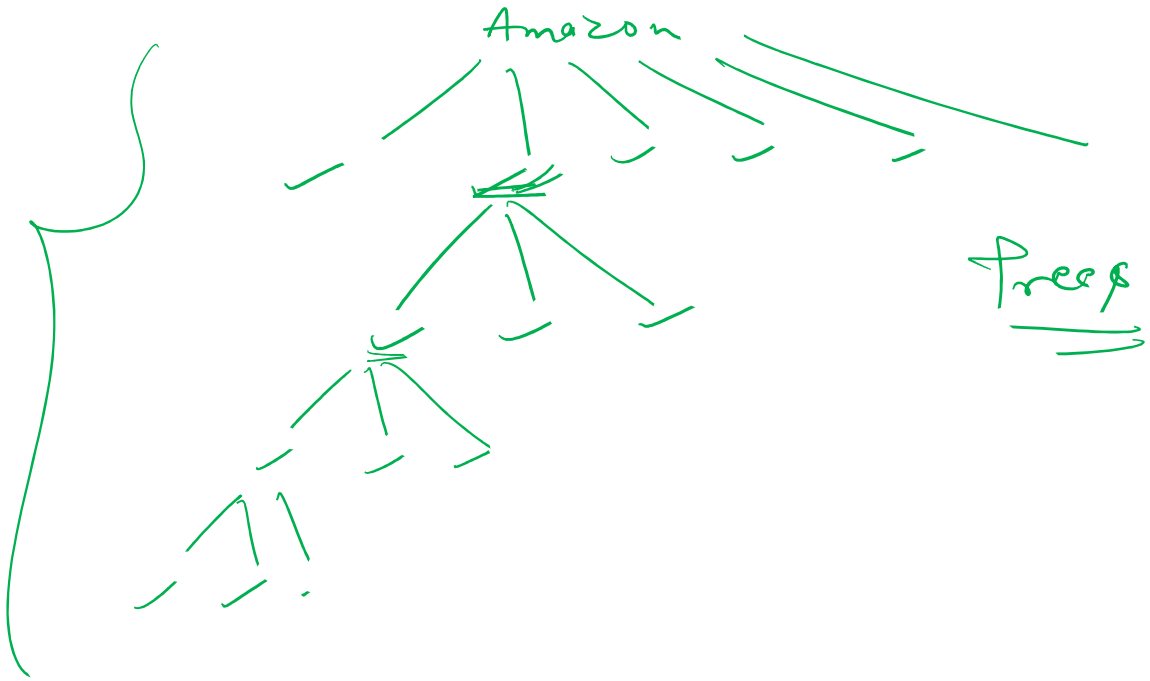
Tree

Linear structure
array
LL

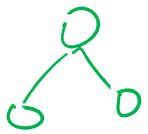
why?!
Level order

Select movie
Action Romatic Thriller

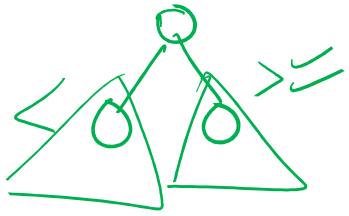
Non Linear
Tree


Amazon
Preps

Binary Tree



BST



0- 1-

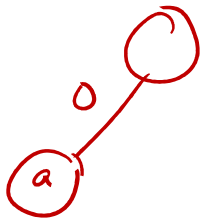
Compress

⇒ 1 byte
8 bits

a ⇒ 5 times
b ⇒ 3 times
c ⇒ 1 time

0
a


aaaaa 5 bytes
5x8 => 40 bits



abc
aaaaabbbc 9 bytes
8x9 = 72 bits

a b c

72 bits

a = 0
b = 1
c = 01

72

10 bits

aaaaa bbb c
00000 1 11 01 ⇒

Sandwich

Binary Tree

8 bits

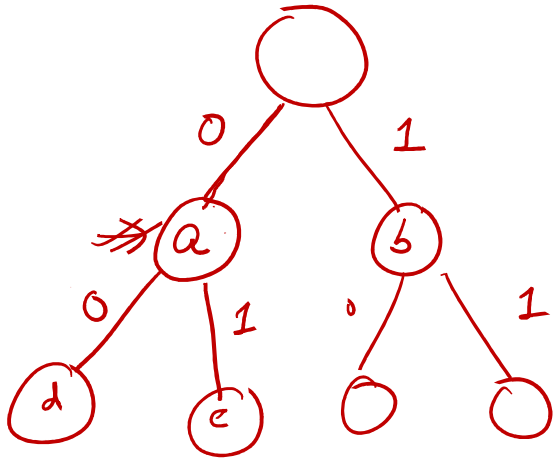
1

2

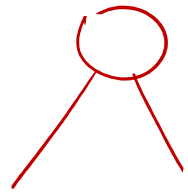
c = 01

d :

Huffman coding



BT



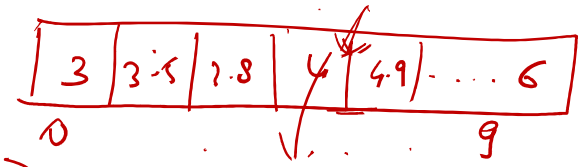
BST

ordered
=
sorted

10 students

holds 3 students

sorted array

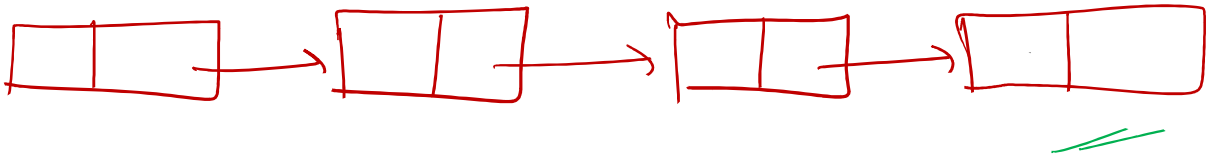


insertion is costly

deletion

4.5

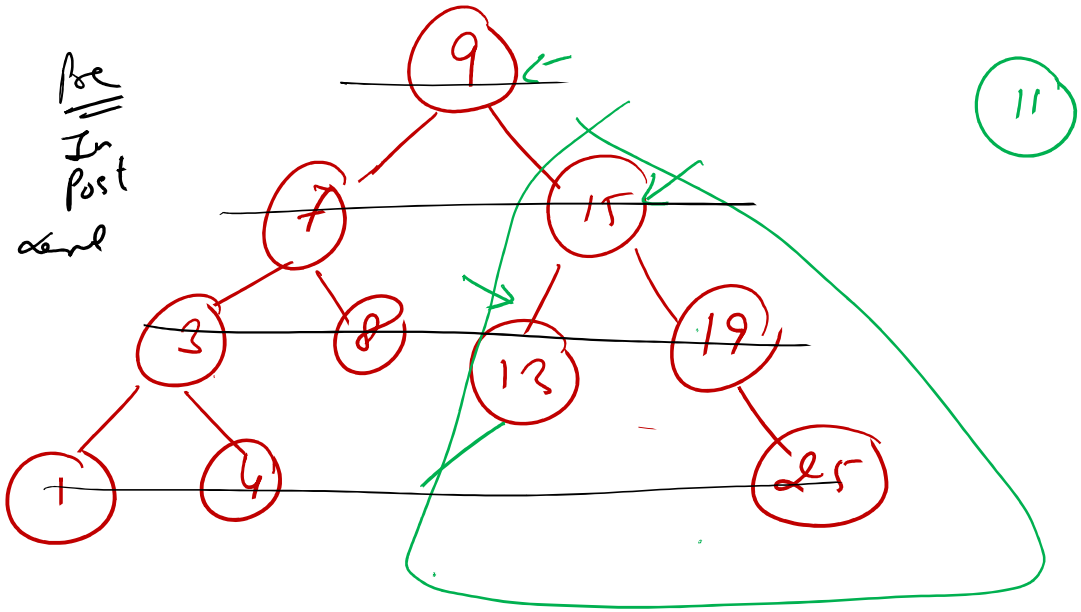
11



BST

ordered / sorted

Pre
In
Post
and



11

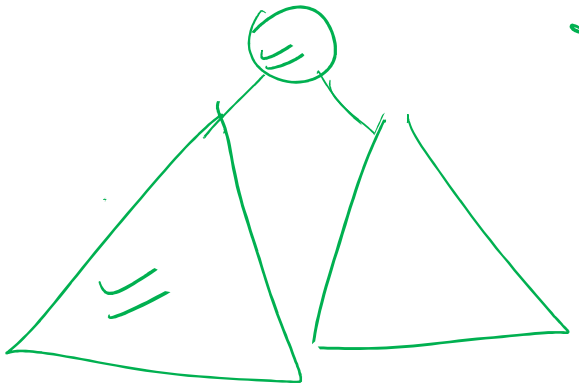
Employee

Price

Dictionary

Alphabetical

Price



Sorted

<

Height

Tc

BST

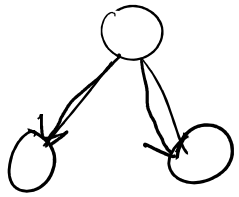
Trees

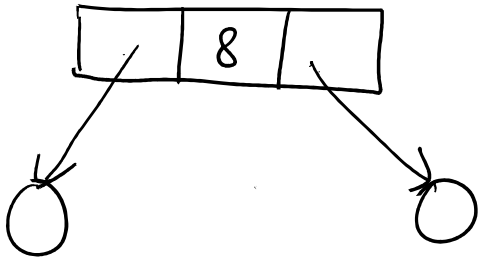
BT

BST

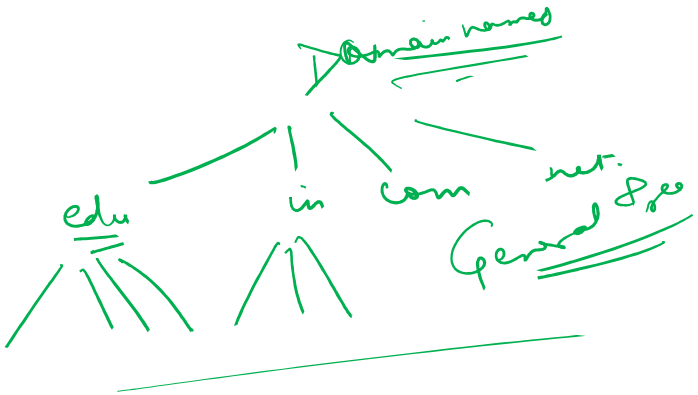
} what, why, how?

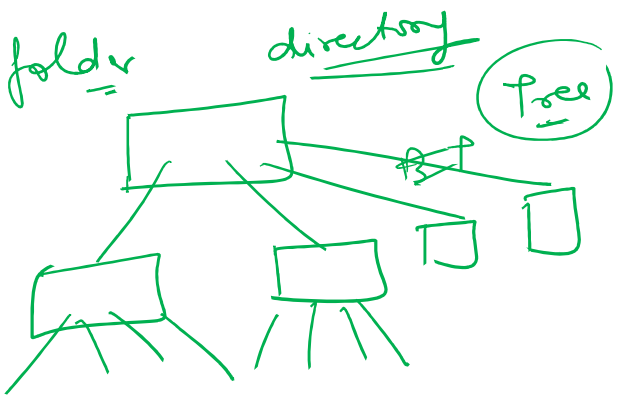
Declaration

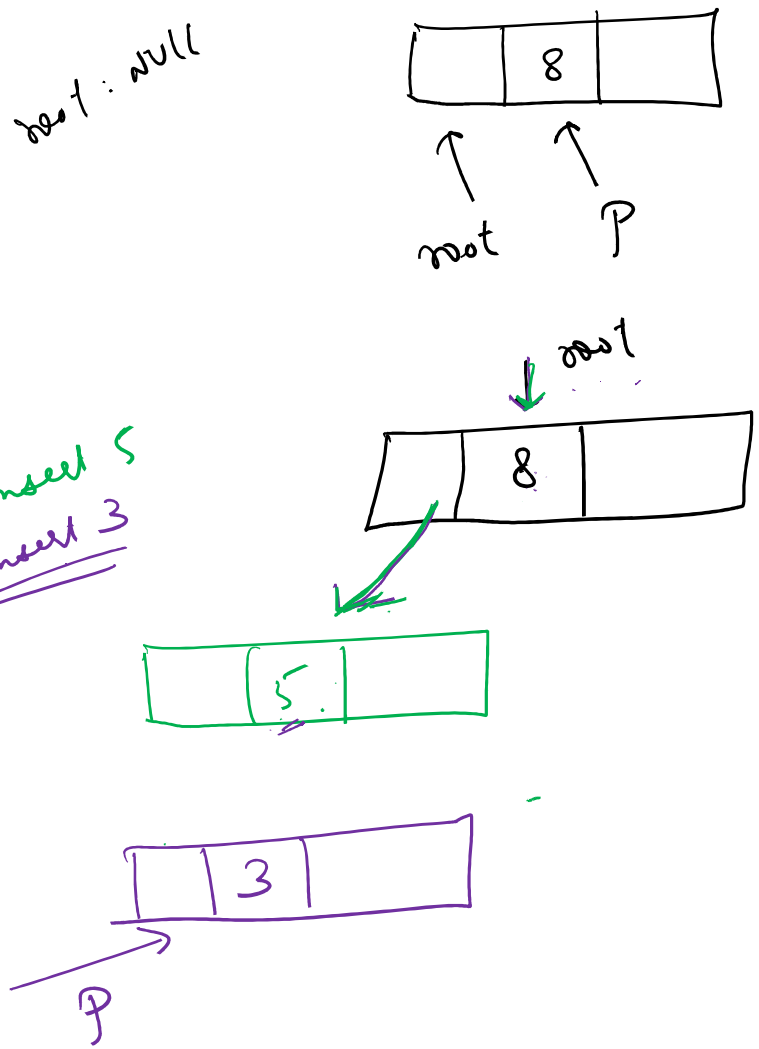





```
class node
{
    public:
     int data;
     node *left;
     node *right;
};
node *root = NULL;
```

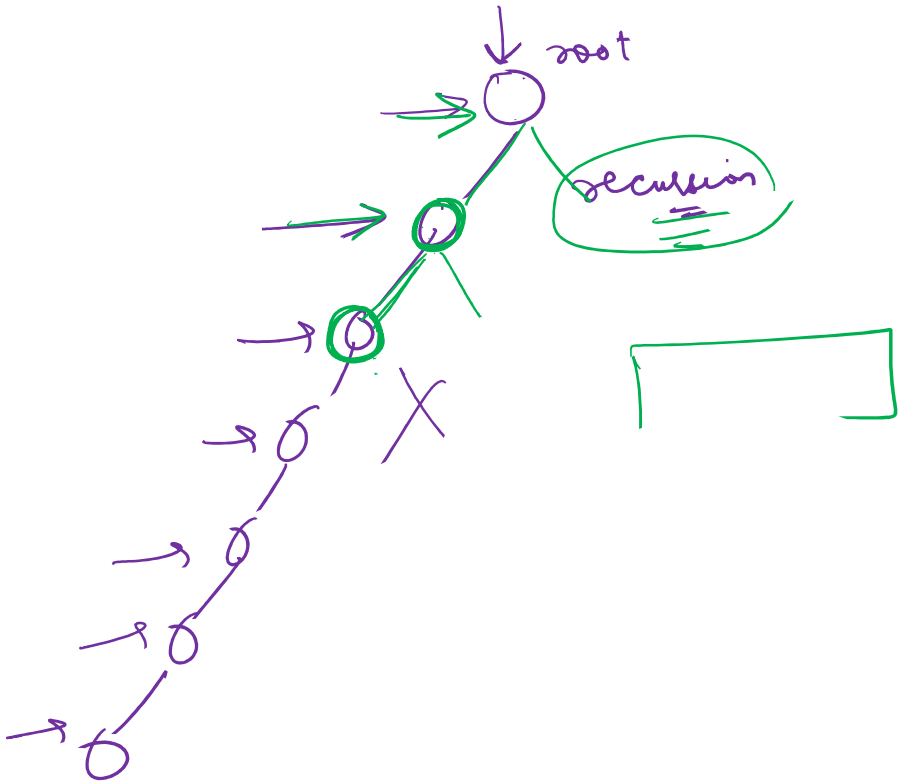


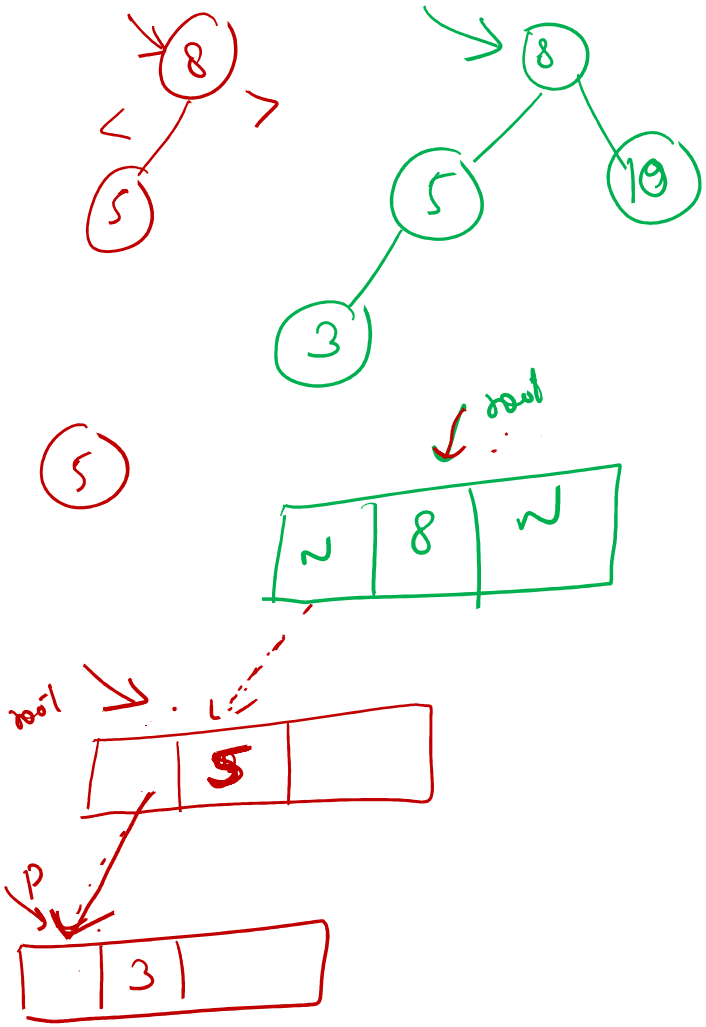




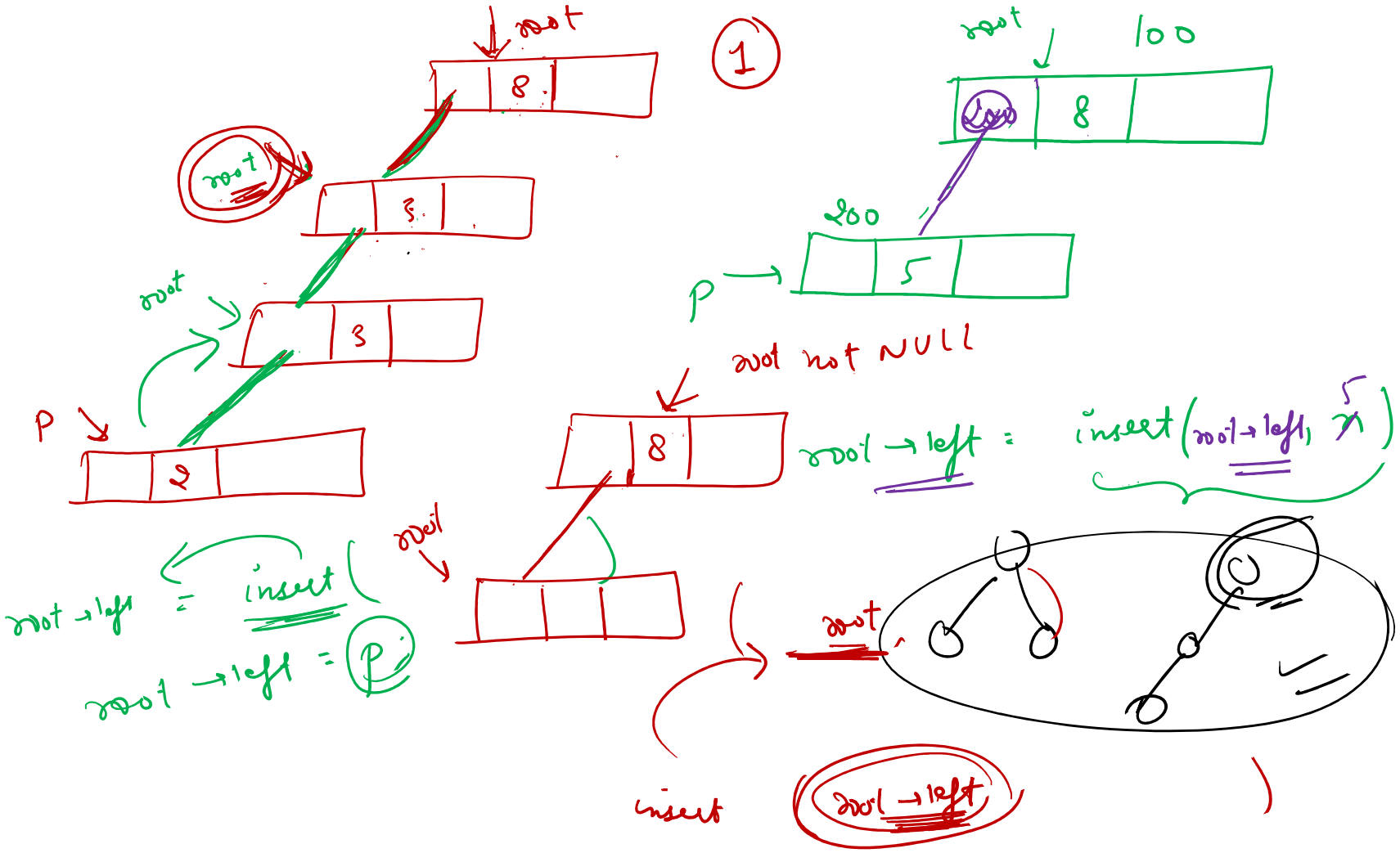

```
node * p = new node;
p -> data = 8;
p -> left = NULL;
p -> right = NULL;
root = p;
```

```
if (p -> data < root -> data)
    root -> left = insert(root -> left, value);
```


root
recursion



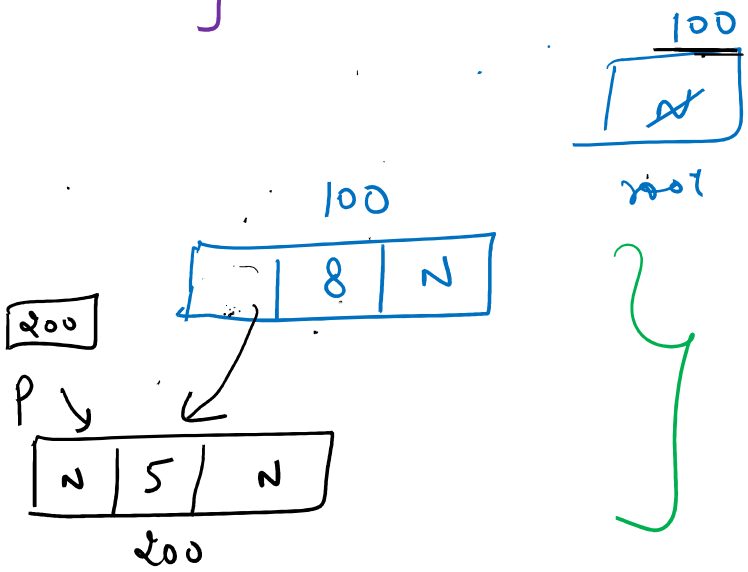

```
node* insert ( __ , int n )
{
  if ( root == NULL)
{
    node * p = new node;
      p→data = n;
      p→left = NULL;
      p→right = NULL;

else if ( n < root→data )
root→left = insert (      , n)
```

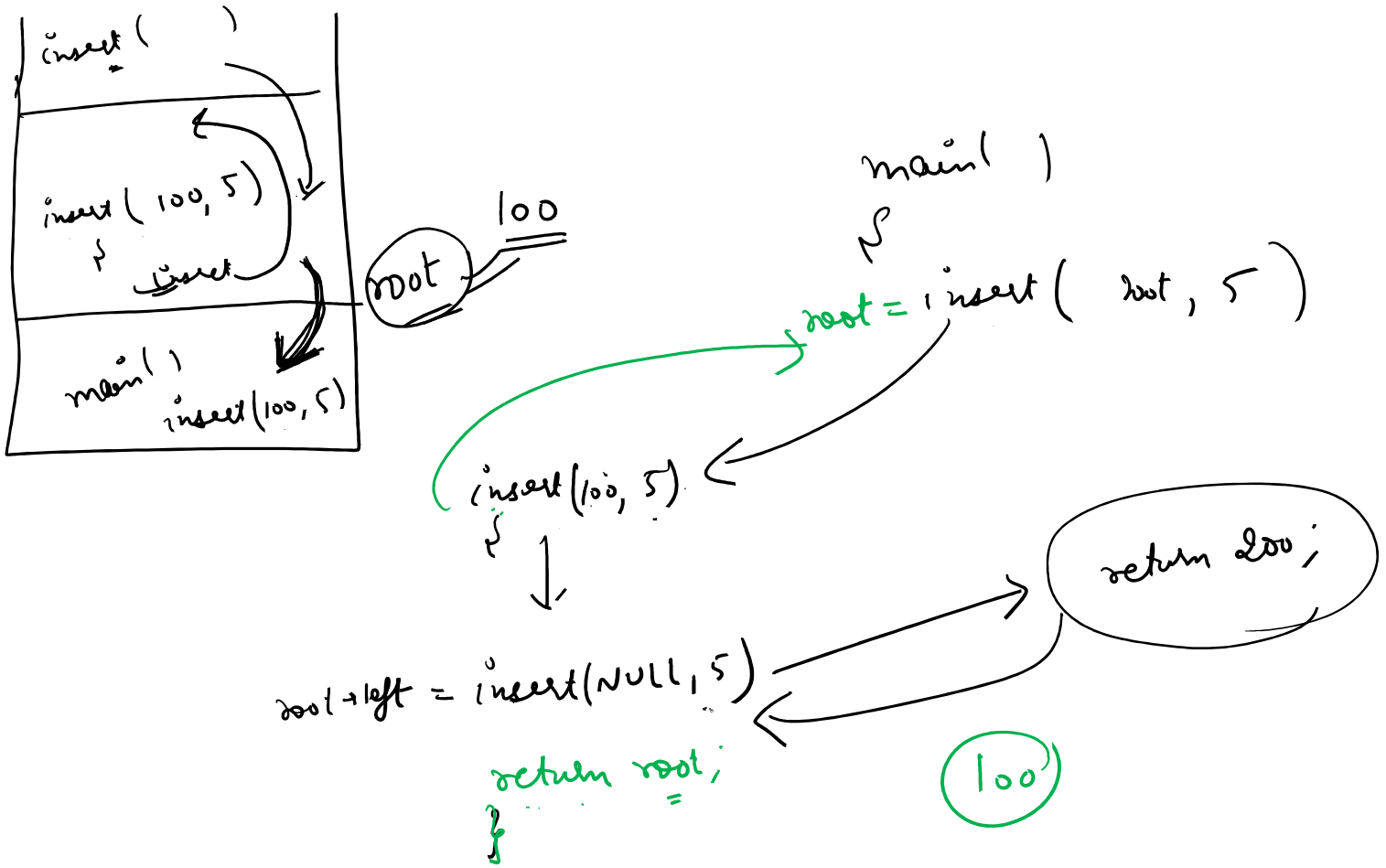

root
8
3
root
3
p
2
root
8
root not NULL
root → left = insert(root → left, 5)
root → left = insert(
root → left = p
insert
root
root → left
1
root
100
8
200
p
5

```
node * insert( node * root , int n )
{
    if ( root == NULL )
    {
        node *p = new node;
        p->data = n;
        p->left = NULL;
        p->right = NULL;
        return p;
    }
    else if ( n < root->data )
        root->left = insert( root->left, n );
    return root;
    else
```

5 < 8 True

```
// node *root = NULL;
int main( )
{
    // node *root = NULL;
    root = insert( root, 8 );
    root = insert( root, 5 );
}
```




insert( )
insert(100, 5)
insert
main( )
insert(100, 5)
root
100
main( )
root = insert(root, 5)
insert(100, 5)
root->left = insert(NULL, 5)
return root;
}
return 100;
100

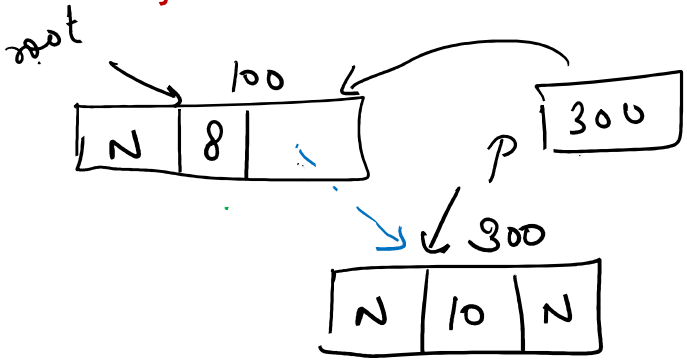
```
node *insert(node *root, int n)
{
  if (root == NULL)
  {
    node *p = new node;
    p->data = n;
    p->left = NULL;
    p->right = NULL;
    return p;
  }
else if (n < root->data)
  root->left = insert(root->left, n);
  return root;

else
root->right = insert(root->right, n);
  return root;
}
```

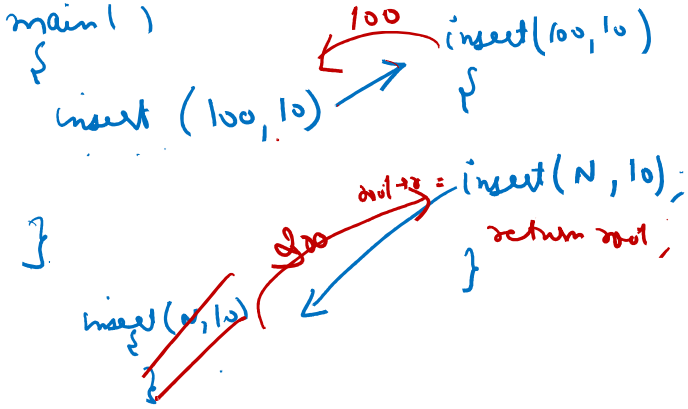
10 < 8
F
9 ≤ 10

T

(8)

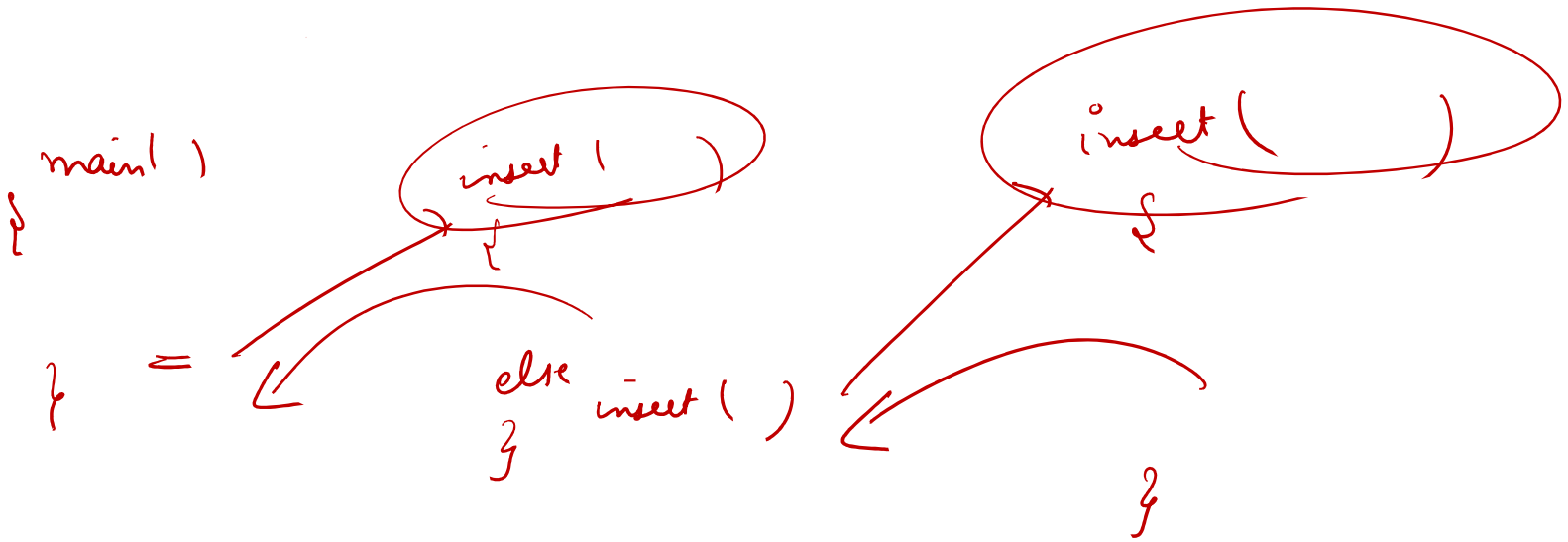
```
node *root = NULL;
main()
{
root = insert(root, 8);
root = insert(root, 10);
}
```

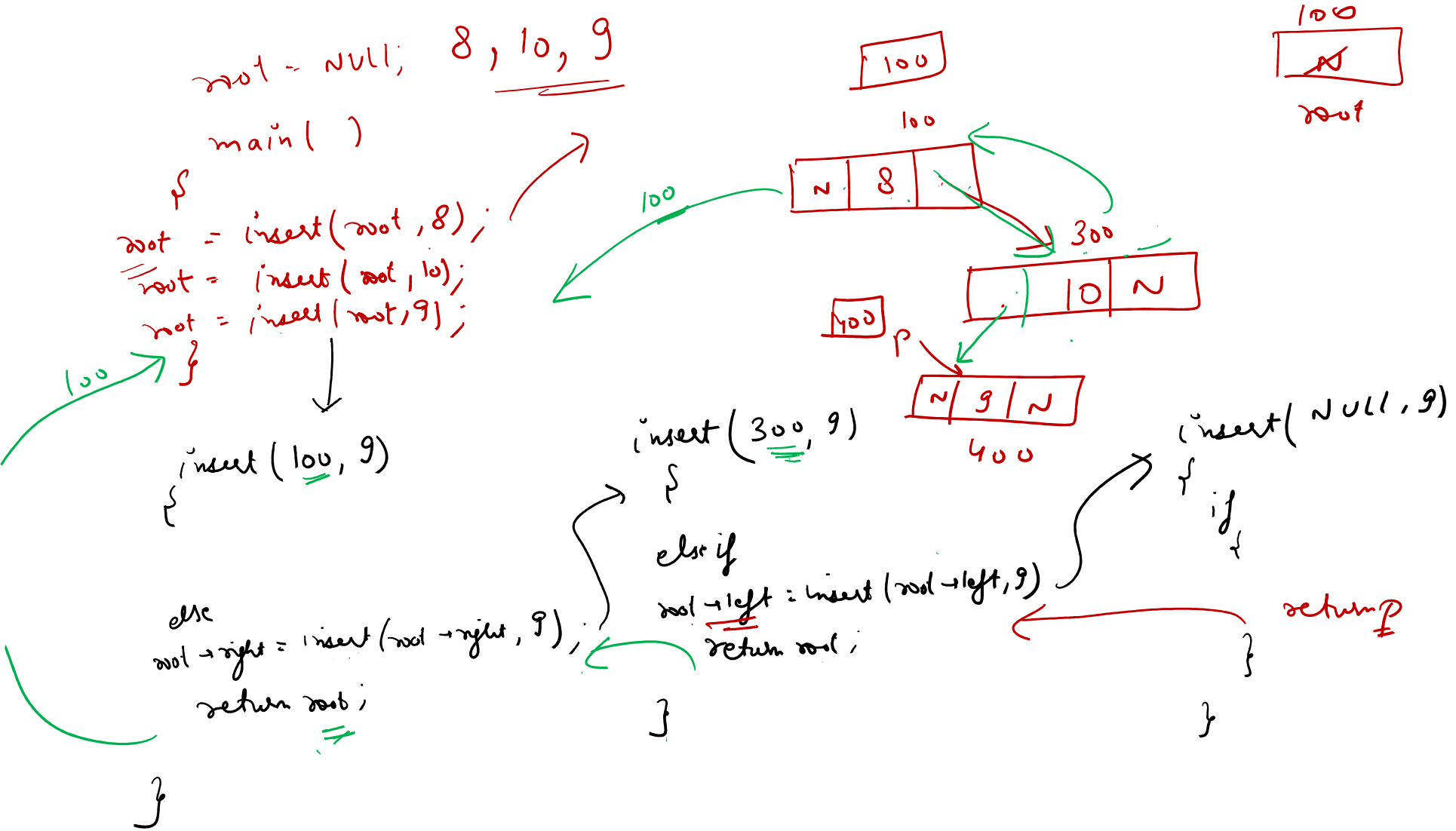




insert

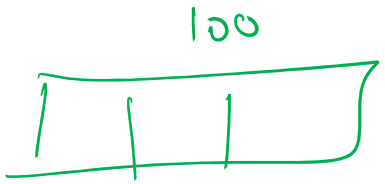
(10)




root = NULL;
8, 10, 9
main( )
{
root = insert(root, 8);
root = insert(root, 10);
root = insert(root, 9);
}
100
insert(100, 9)
{
else
root->right = insert(root->right, 9);
return root;
}
insert(300, 9)
{
else if
root->left = insert(root->left, 9)
return root;
}
insert(NULL, 9)
{
if
return p
}
100
N 8
300
10 N
400
p
N 9 N
400
100
N
root

insert(, int)

{



insert(100, 8)

;

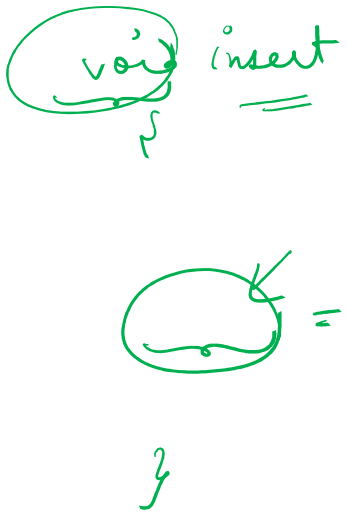
insert(300

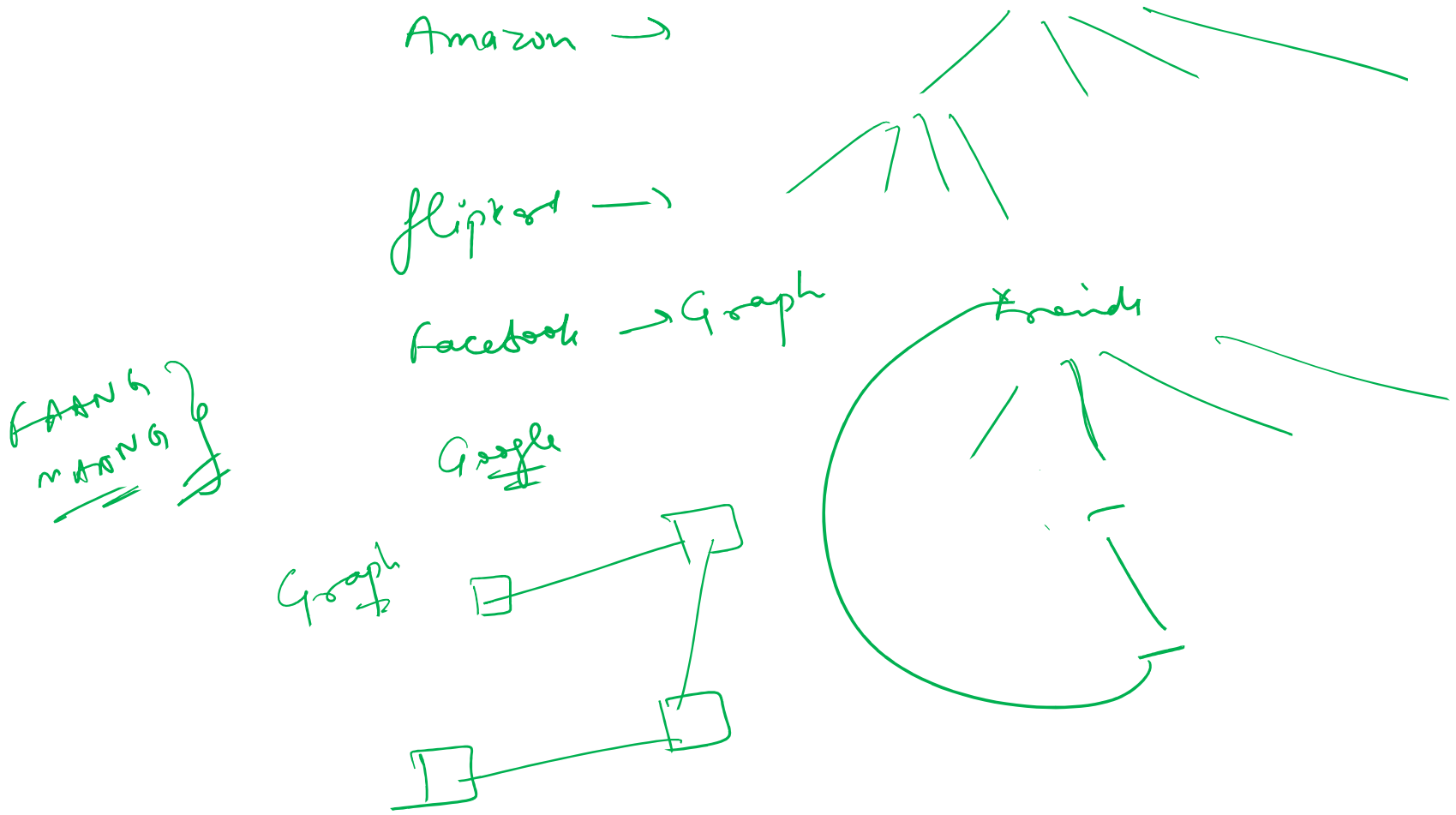
}

8, 10, 5, 13, 9, 3, 1

Assignment

Pdf

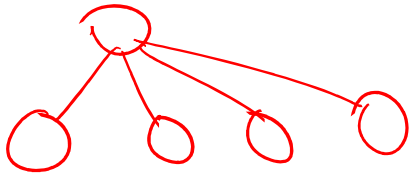



Amazon →
flipkart →
facebook → Graph
Google
FAANG
MAANG
Graph
Friends

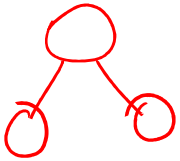
Trees

↓

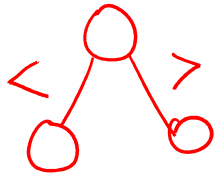
NL DS



General Tree



Binary Tree



Binary Search Tree

//

Binary Search Tree

↓

Insertion

what

why

how →

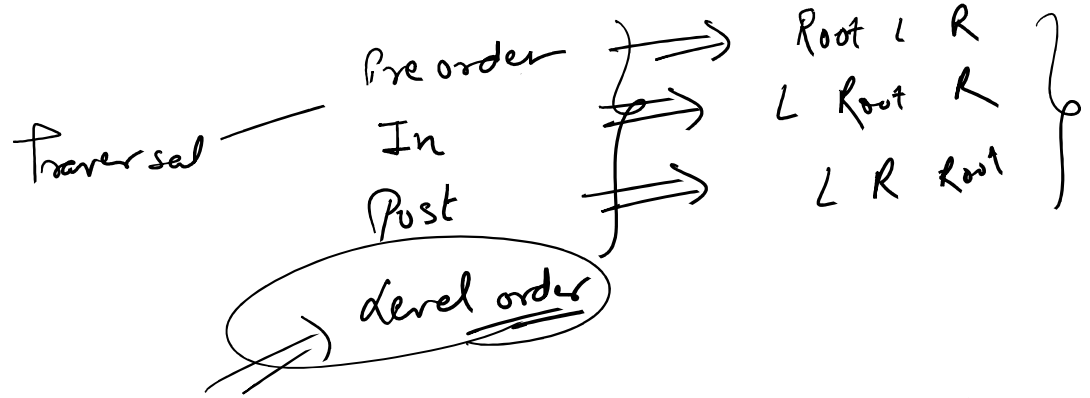
operations

- Insertion
- deletion
- Traversal

class node

↓

```
int data;
node * left;
node * right;
};
```

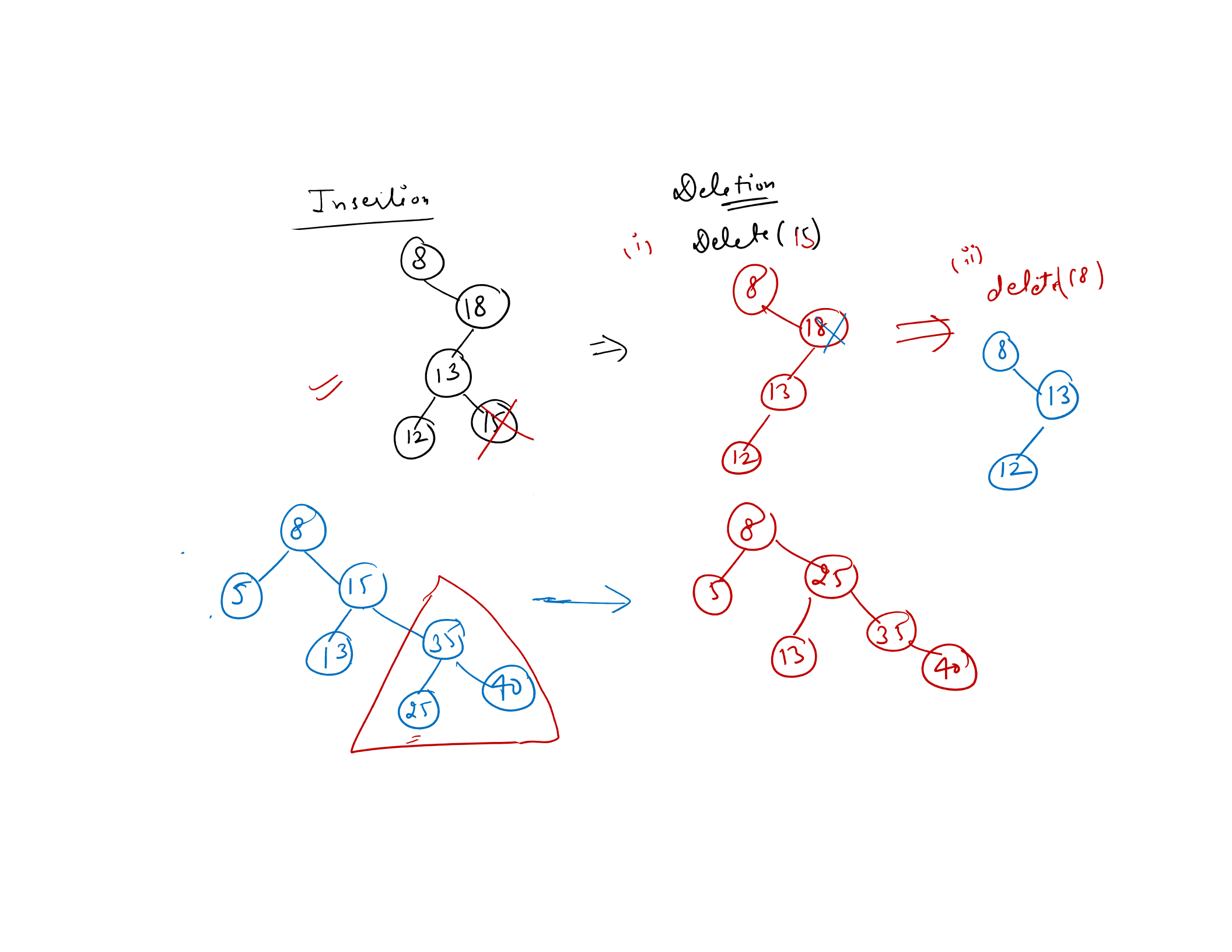


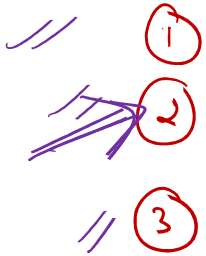
Pre
Post
Pre
In
Post
In

BST

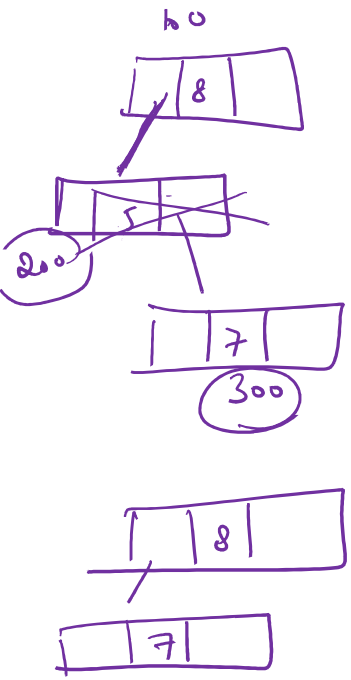
Pre
In

Post
In

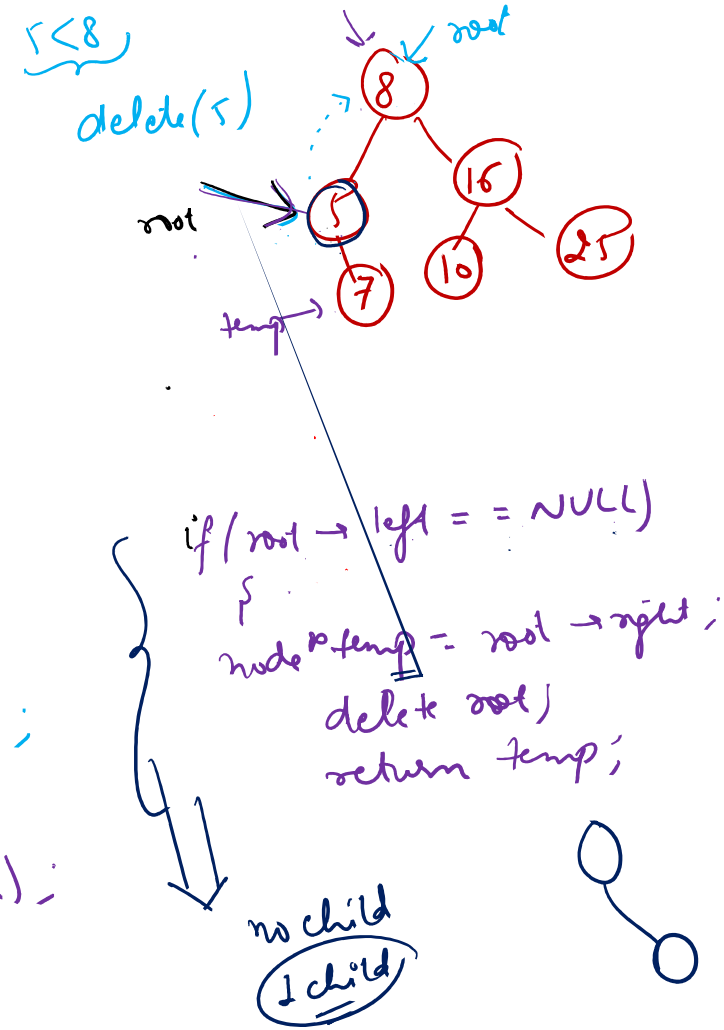

Insertion
8
18
13
12
15
Deletion
Delete (15)
(i)
8
18
13
12
(ii)
delete(18)
8
13
12
8
5
15
13
35
25
40
8
5
25
13
35
40

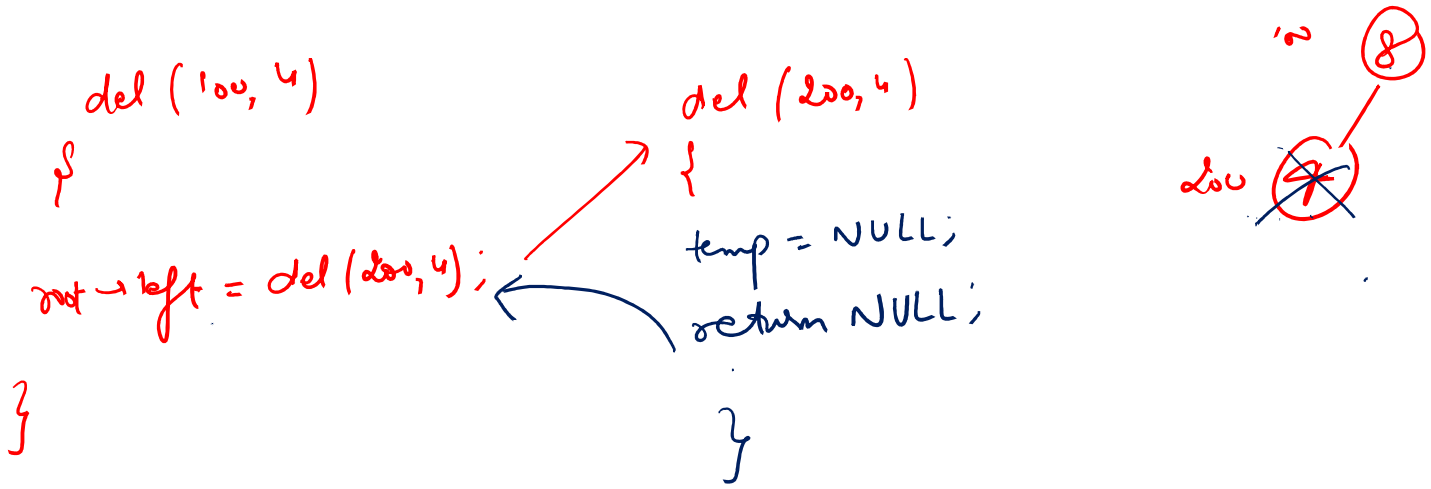


no child
1 child → left child
" → right "
2 children → Inorder predecessor
" → successor

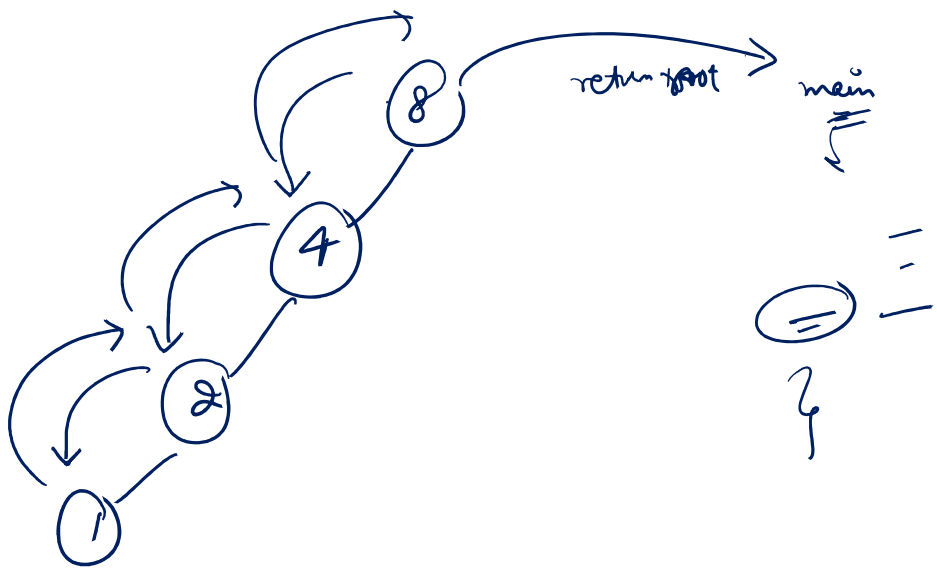



```
node * deletenode (node *root, int k )
{
    if (root == NULL)
        return root;
    if ( k < root -> data )
root->left = deletenode ( root -> left , k);
    else if ( k > root ->data)
root->right = deletenode ( root -> right , k);
```




del (100, 4)
{
root → left = del (200, 4);
}
del (200, 4)
{
temp = NULL;
return NULL;
}

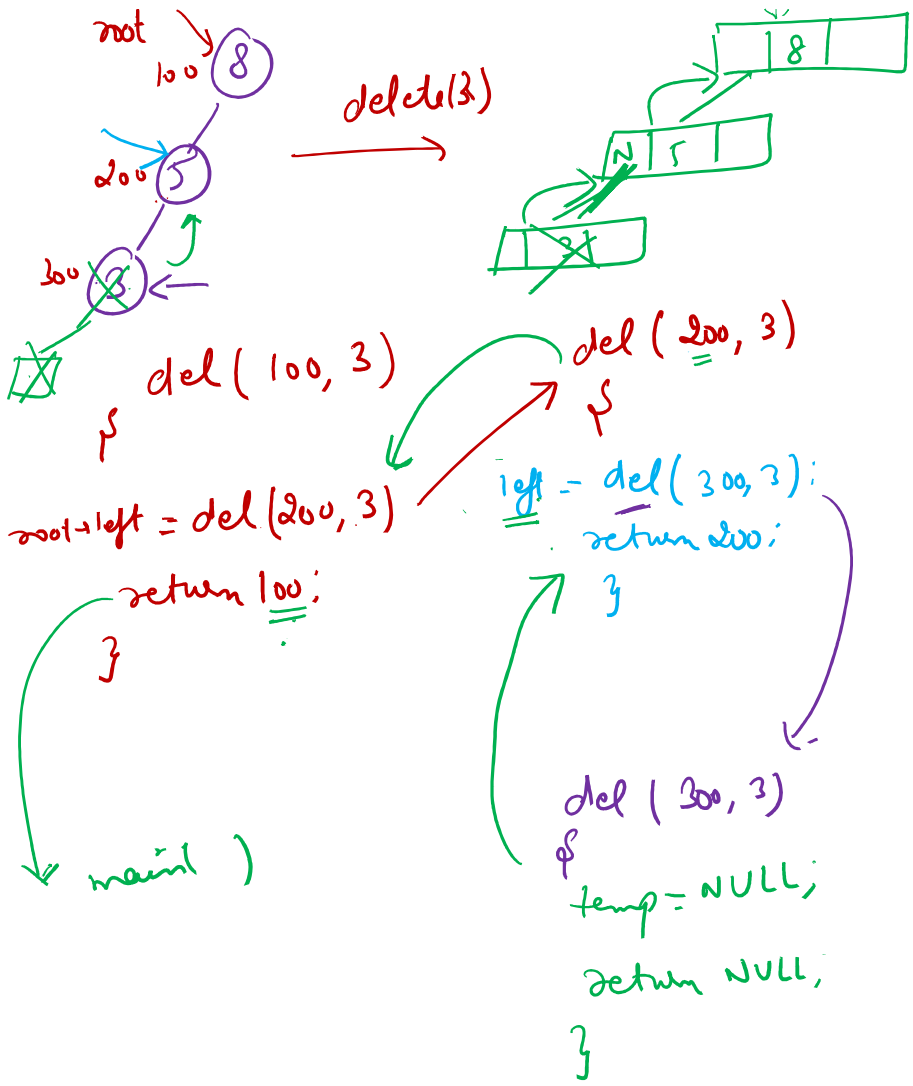
8
200

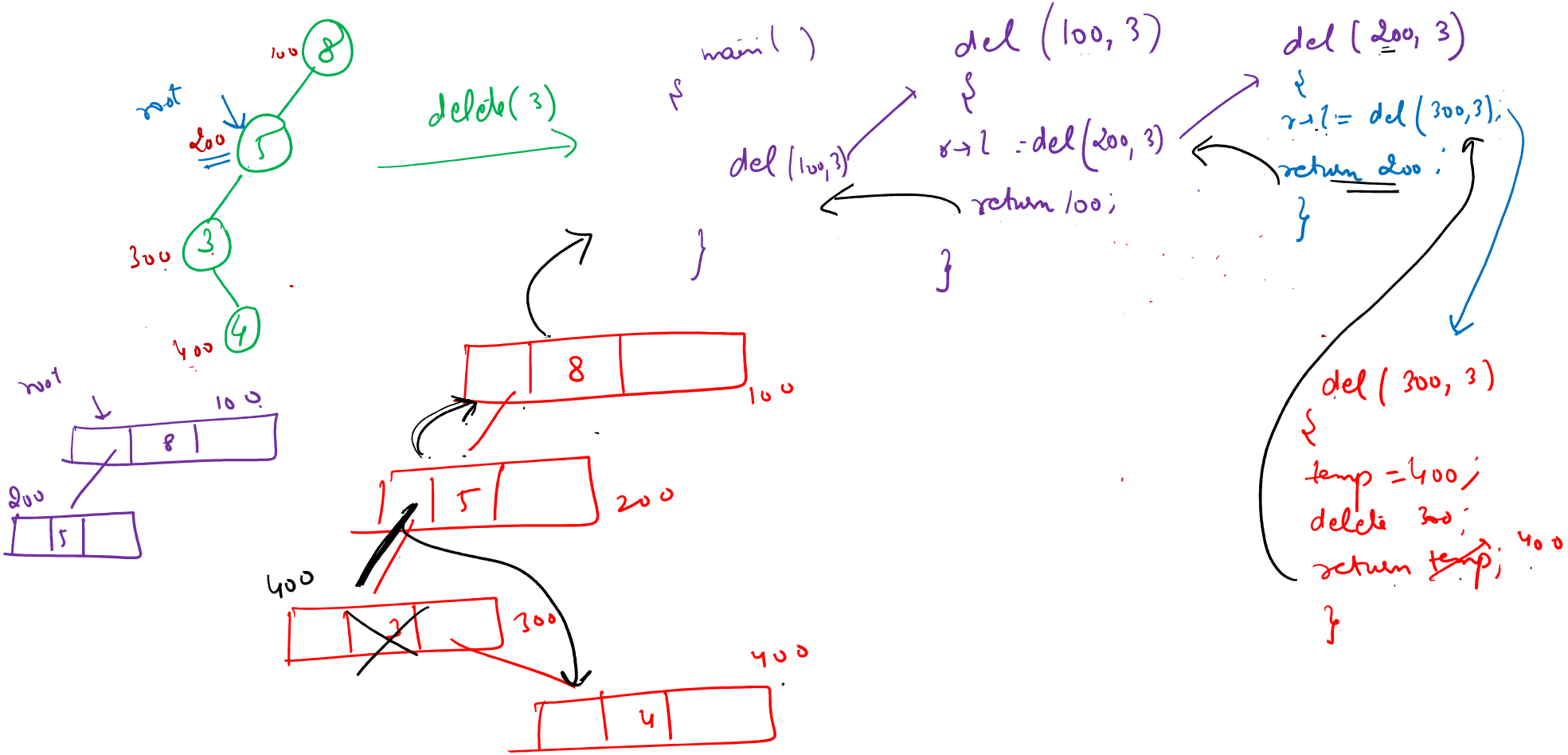

8
4
2
1
return
main

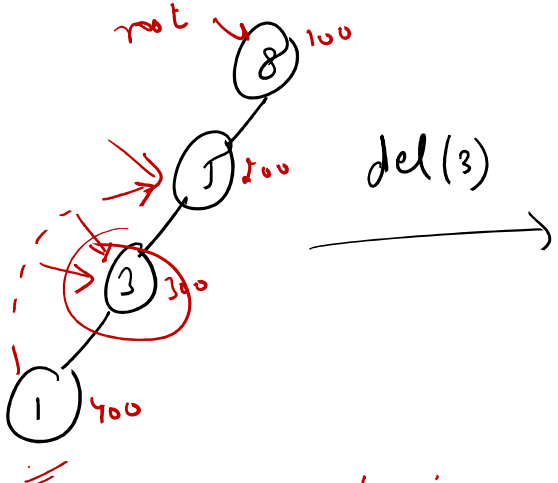
```
node * deletenode(node* root , int k)
{
  if(root == NULL)
     return root;
  if( k < root->data)
     root->left = deletenode(root->left, k);
     return root;
  else if( k > root->data)
     root->right = deletenode(root->right, k);
     return root;
  if( root->left == NULL)
  {
     node *temp = root->right;
     delete root;
     return temp;
  }
  else if( root->right == NULL)
  {
     node *temp = root->left;
     delete root;
     return temp;
  }
}
```

5 < 5
5 > 5

3 < 8 t
3 < 5 t
3 < 3 f
3 > 3 f




100
8
root
200
5
delete(3)
300
3
400
4
root
100
8
200
5
main()
del(100,3)
del(100,3)
r→l = del(200,3)
return 100;
del(200,3)
r→l = del(300,3);
return 200;
del(300,3)
temp = 400;
delete 300;
return temp;
400
100
8
200
5
400
300
400
4



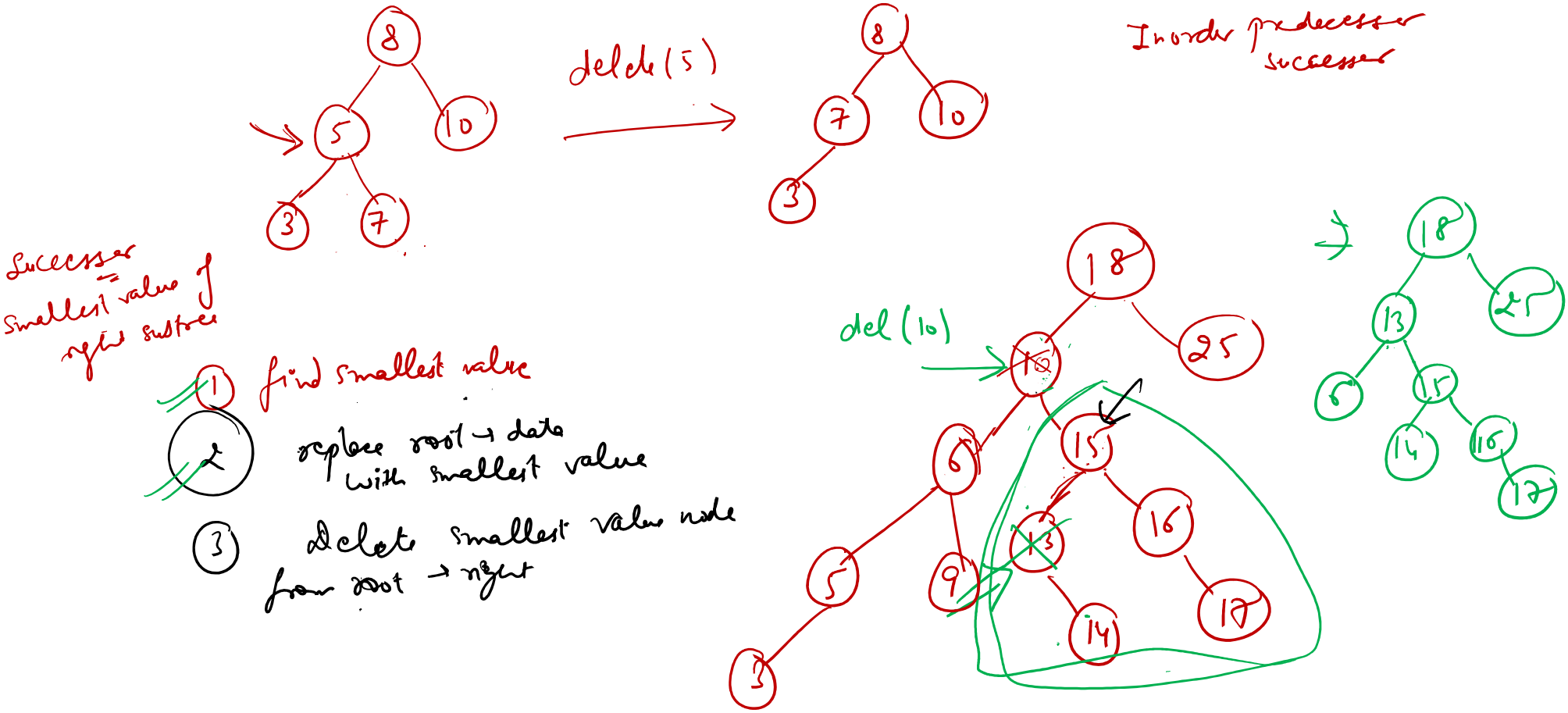

```
main()
{
  del(100, 3);
}
```

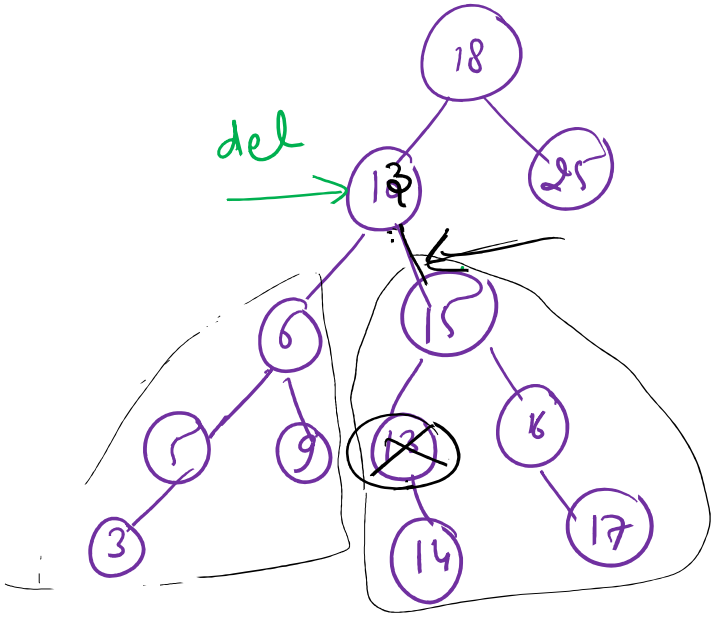
```
del(100, 3)
{
```

1) no child
2) 1 child
3) 2 children

temp = 400;

return 400;


8
5
10
3
7
delete(5)
8
7
10
3
Inorder predecessor
successor
Successor
=
Smallest value of
right subtree
1
find smallest value
2
replace root → data
with smallest value
3
delete smallest value node
from root → right
del(10)
18
10
25
6
15
5
9
13
16
3
14
17
18
13
25
6
15
14
16
17



Inorder predecessor → left subtree ⇒ max/biggest value

successor → right subtree → smallest value

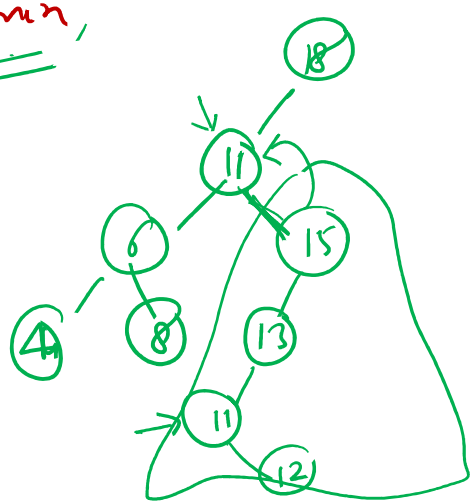
pre suc

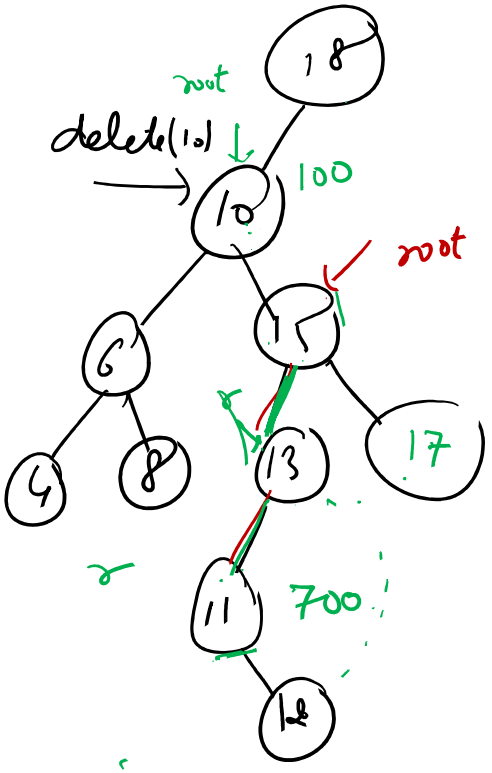
⇒ 3 5 6 9 10 13 14 15 16 17

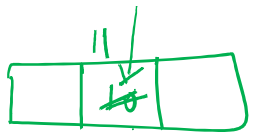
18 25

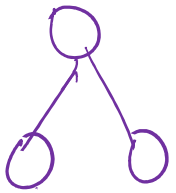

```
root->data = minVal(root->right);
root->right = del(root->right, 11);
```

```
int minVal(node *root)
{
    int min = root->data;
    while(root->left != NULL)
    {
        min = root->left->data;
        root = root->left;
    }
    return min;
}
```









Deletion

- 0 child — left child / right child
- 1 child
- 2 — inorder predecessor → largest value of left subtree
  succ → smallest value of right subtree

dele

① if(root == NULL)

② if (k < root → data)
left

③ elseif (k > root → data)
right

④ if (root → left == NULL)
{
temp = root → right
return temp;

no child / right child

⑤ else if (root → right == NULL)
{
temp = root → left;

left child

}

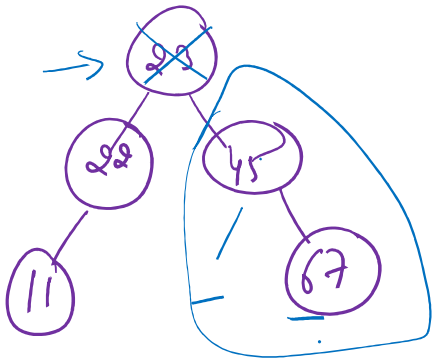
else
{
root → data = minVal (root → right) ;
root → right = del (root → right, min) ;

2 children

}

Root L R

L R Root



11 22 23 45 67 In

23 22 11 45 67 Pre

~~11 22 45 67 23~~ Post

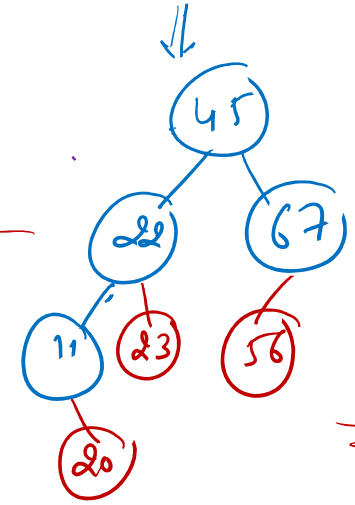
11 22 67 45 23 Post

Preorder

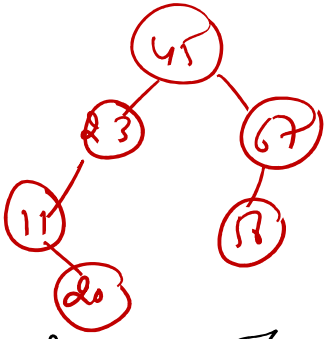
45 22 11 20 23 67 56

Post

20 11 23 22 56 67 45



22



45 23 11 20 67 56