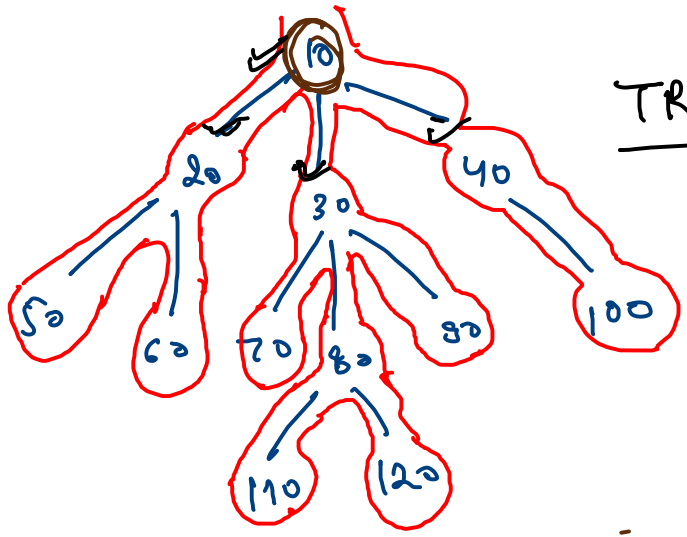
Similar in shape -

Expectation → isSimilar(10, a) → True / False

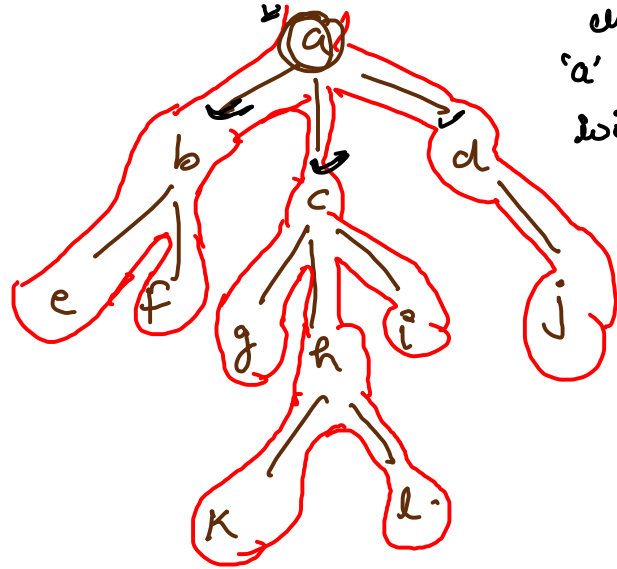
faith → isSimilar(10 i, a i) → loop of child of 'a' & '10' with index i → True / False

TREE1.



TRUE

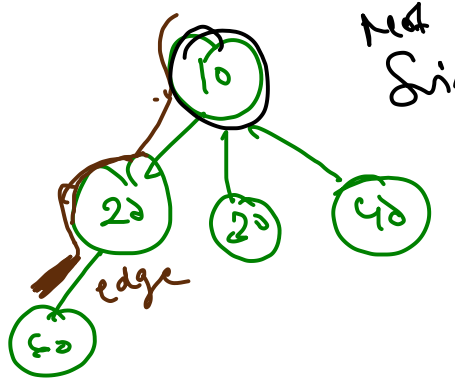
TREE2



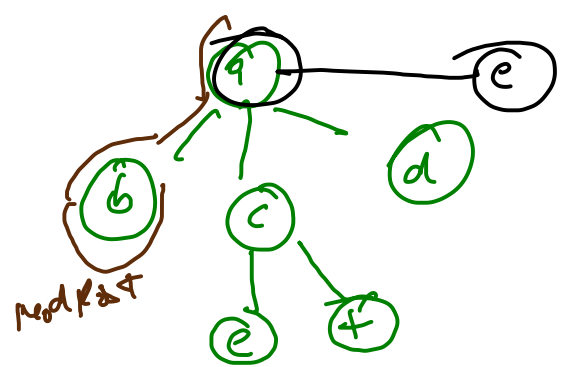
My code → Analysis using Recursion for one level

return

Traversal



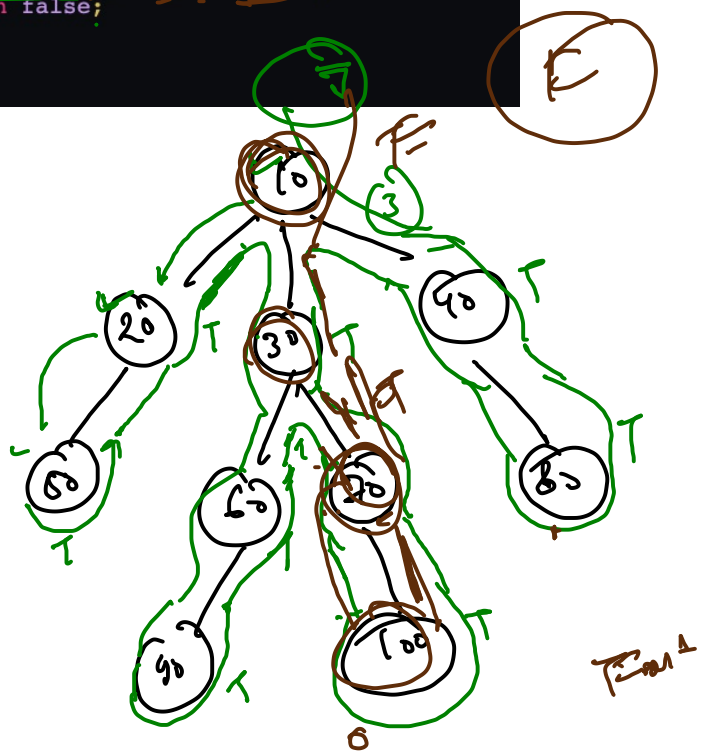
Not Sim

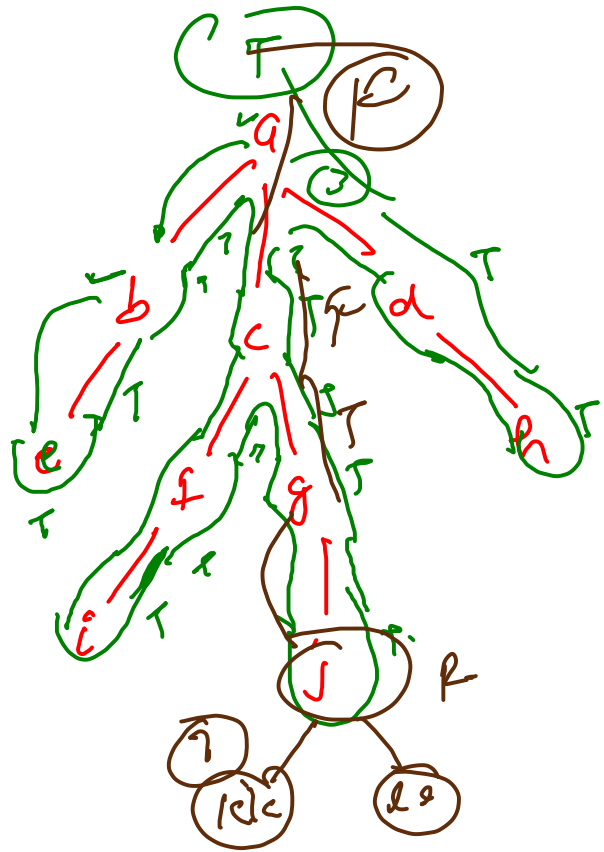


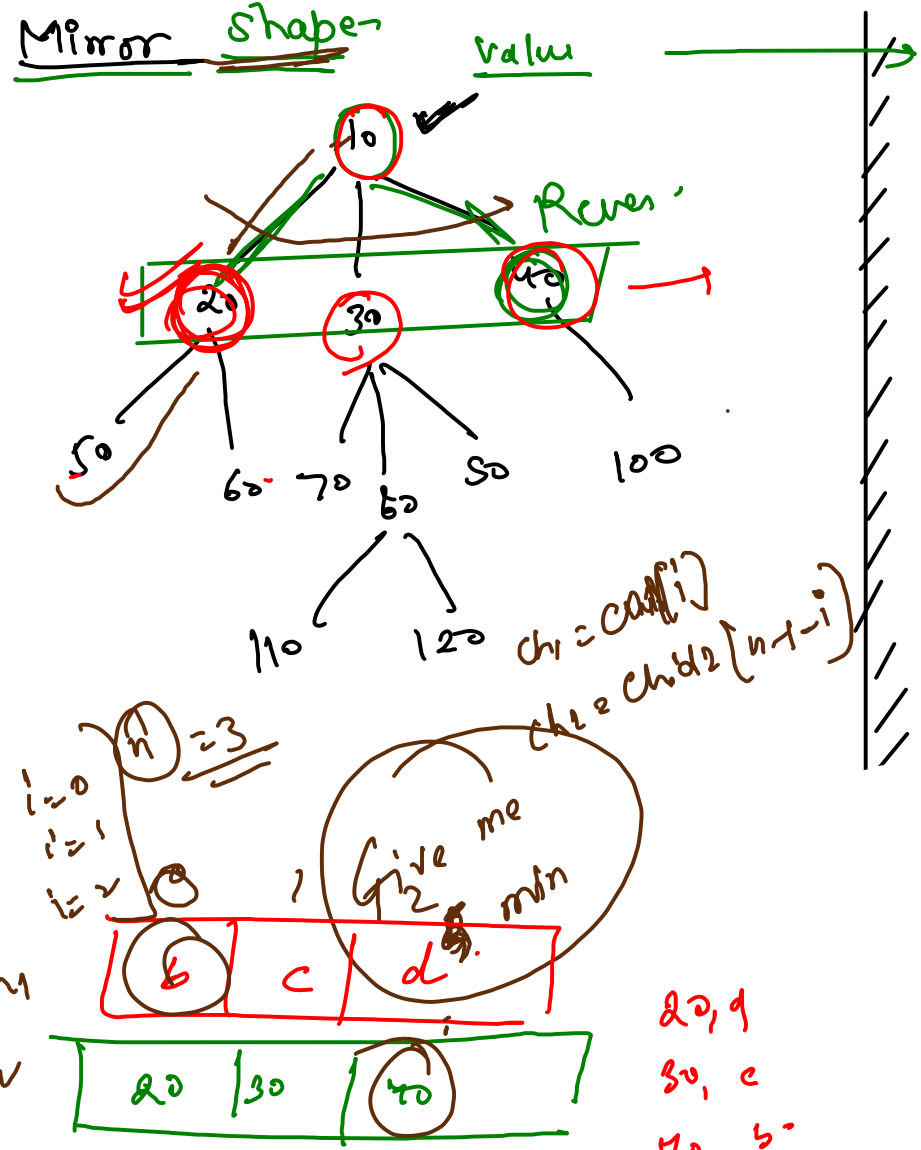
No Similar.

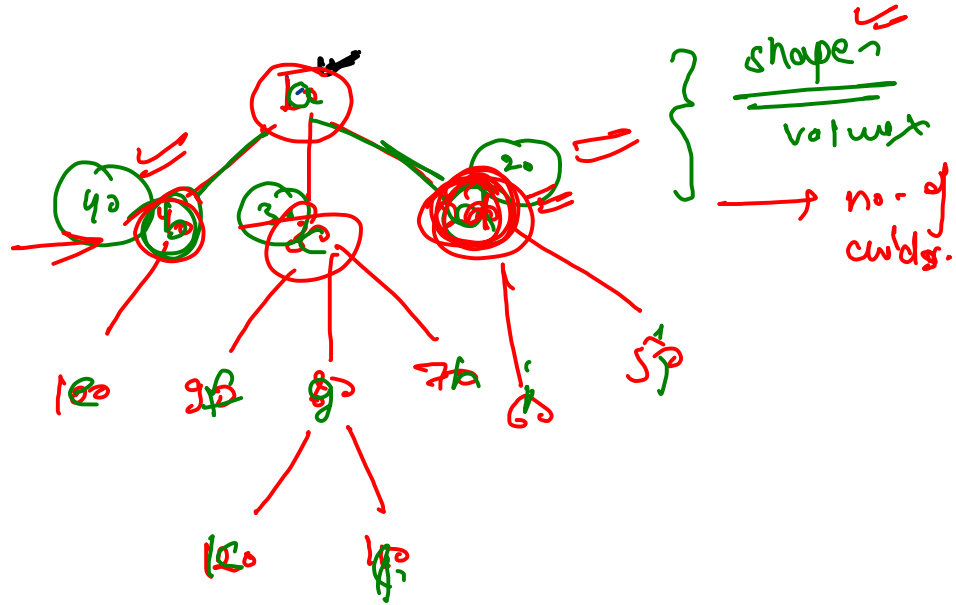
FALSE

```java
public static boolean areSimilar(Node n1, Node n2) {
  // write your code here
  if(n1.children.size() != n2.children.size())
      return false;

  for(int i = 0; i < n1.children.size(); i++) {
      Node ch1 = n1.children.get(i);
      Node ch2 = n2.children.get(i);


      if(areSimilar(ch1, ch2) == false)
          return false;
  }
  return true;
}
```

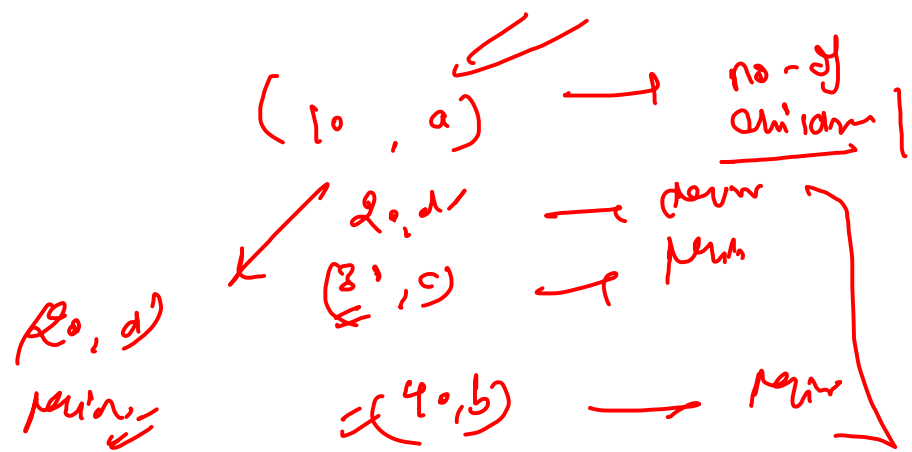





Mirror
Shape
Value
Revers
10
20
30
40
50
60
70
60
80
100
110
120
ch1 = ch1[i]
ch2 = ch2[n-1-i]


shape
value
no. of childs.

n = 3
i = 0
i = 1
i = 2
Give me
min
b
c
d
20
30
40
20, d
30, c
40, b

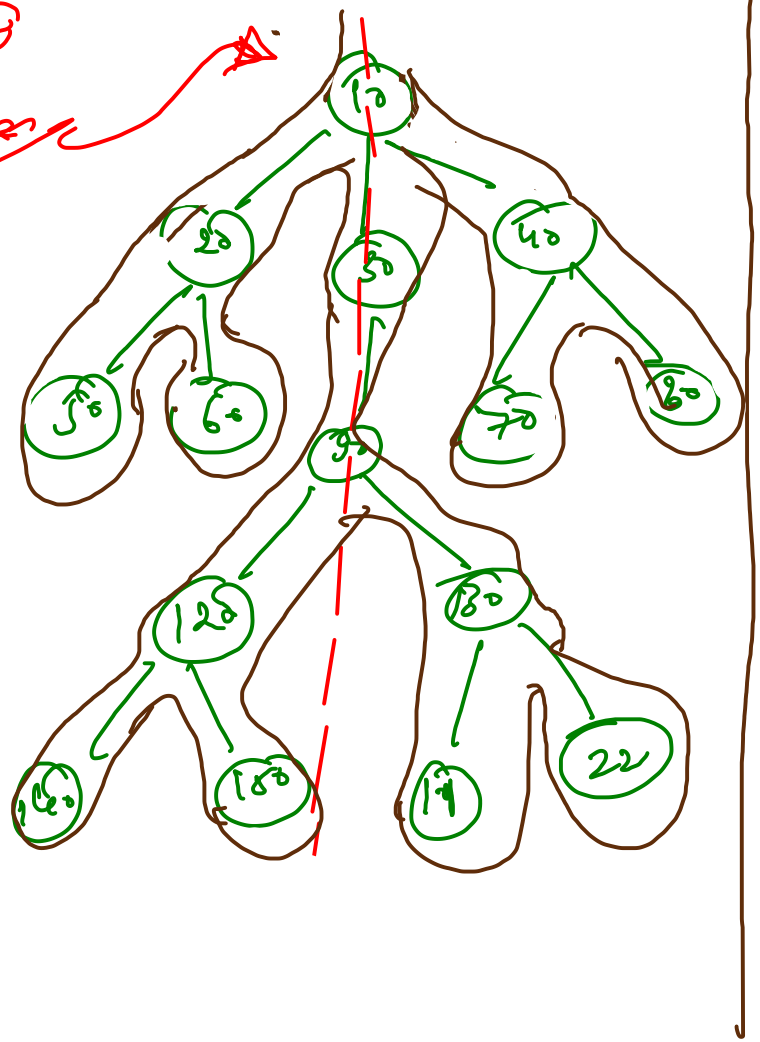

(10, a)
no. of children
20, d
(30, c)
(40, b)
min
min
min

Symmetric Shape.

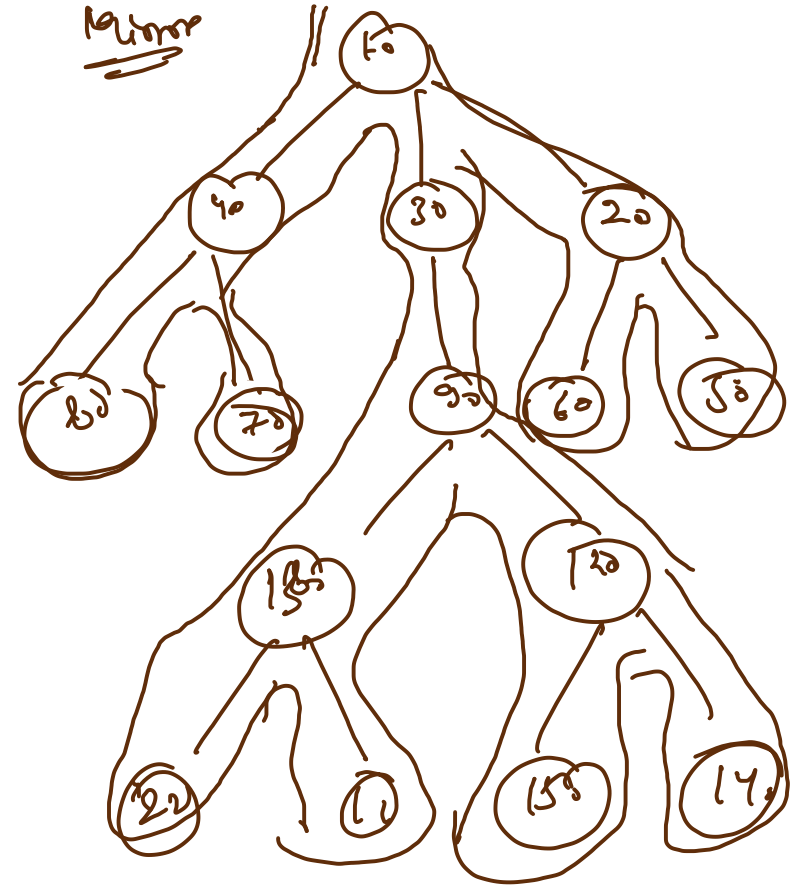
TREE Node

Mirror

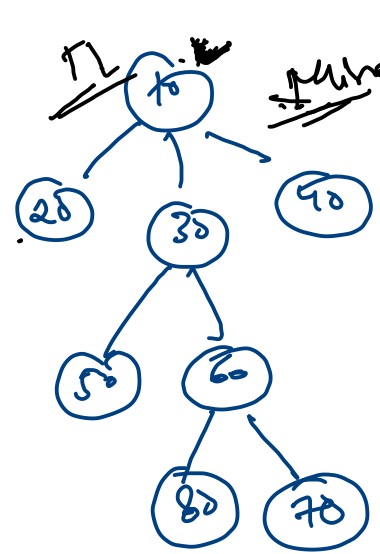
Preorder traversal

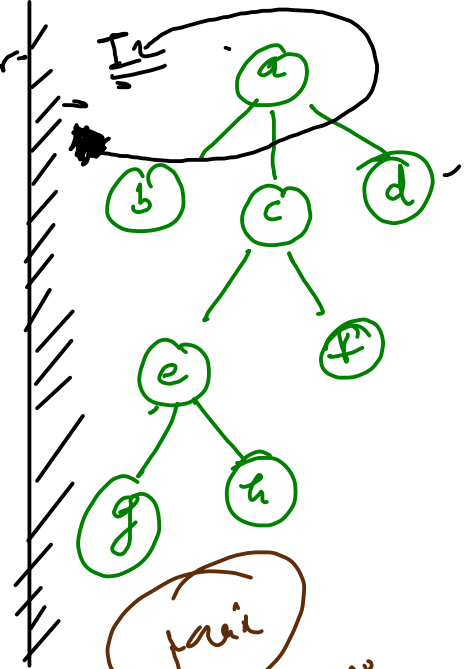


Mirror

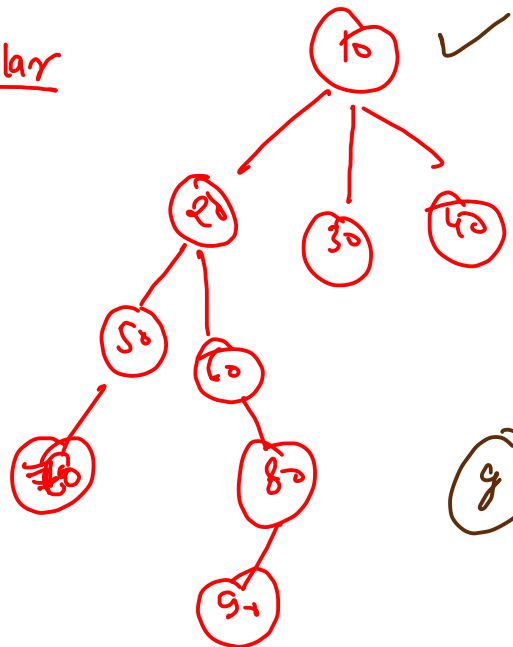


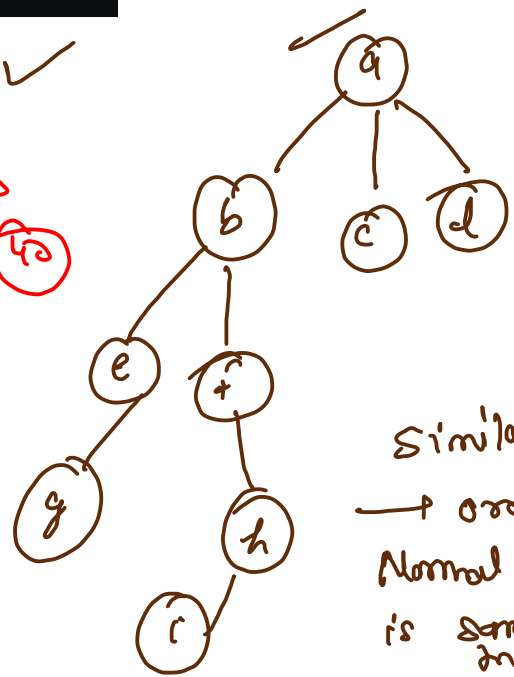
```java
public static boolean isMirror(Node n1, Node n2) {
    if(n1.children.size() != n2.children.size())
      return false;

      int n = n1.children.size();
      for(int i = 0; i < n; i++) {
          Node ch1 = n1.children.get(i);
          Node ch2 = n2.children.get(n - 1 - i);

          if(isMirror(ch1, ch2) == false)
              return false;
      }
      return true;
}

public static boolean IsSymmetric(Node node) {
  // write your code here
  return isMirror(node, node) == true;
}
```



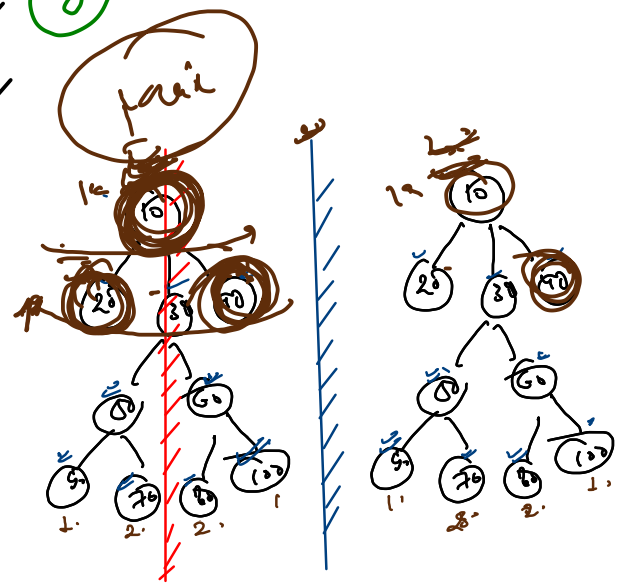


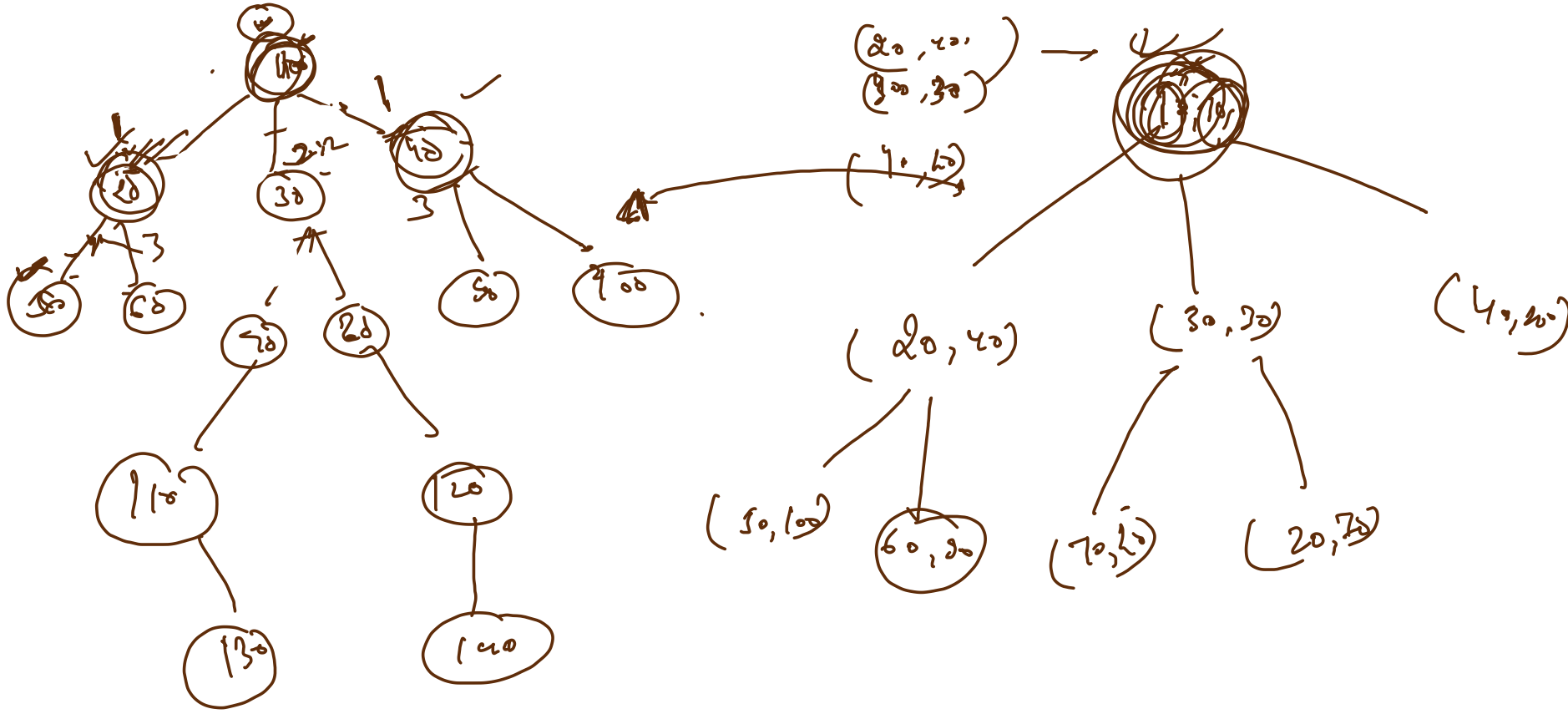
Similar

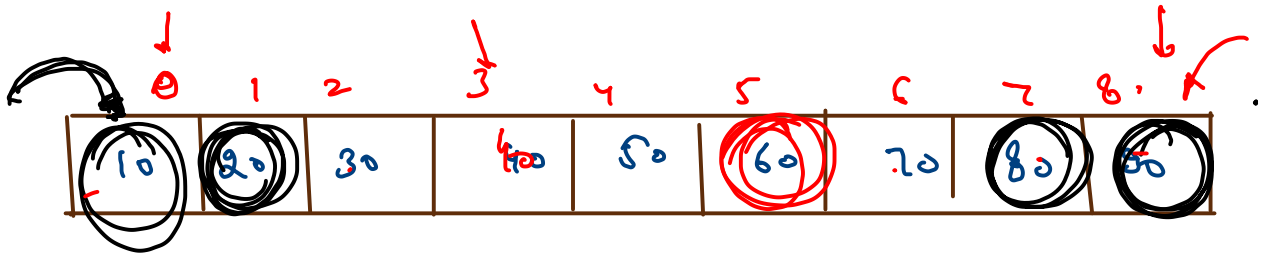




Similar.
→ order of Normal traversal is same, in both trees.




(20, 40)
(30, 30)
(40, 20)
(10, 50)
(20, 40)
(30, 30)
(40, 20)
(50, 10)
(60, 20)
(70, 10)
(20, 70)



i = 3

i = 0

i = 0

condition

i → Normal index for first 'i'

n-1-i → Index for last (i)

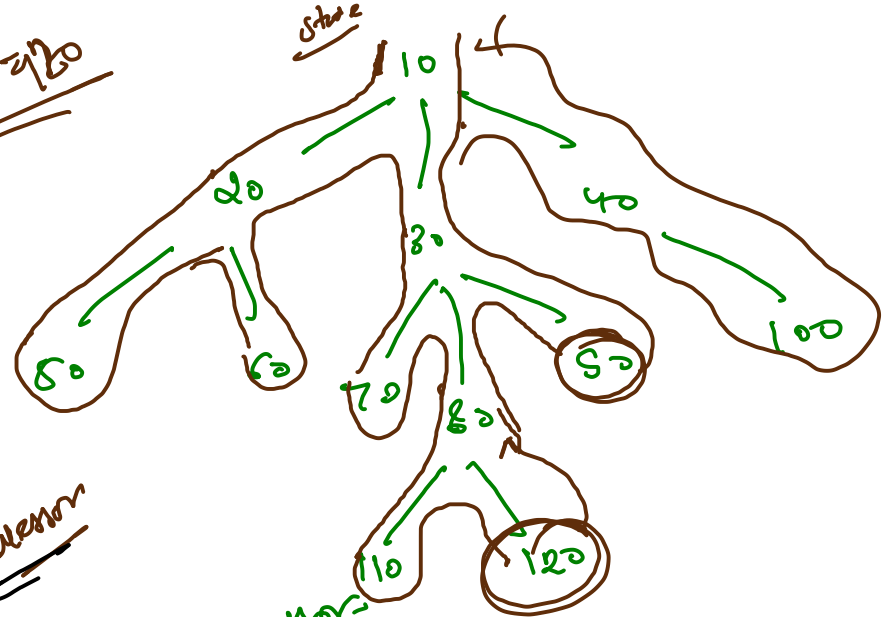
n-1.
8-3 = 5

loop → i = 0;
i < arr.length;
i++

To print → 90 80 -- 20 10 Reverse array.

Predessor & Successor.

Node = 120



60.

pred. = 60

succ = 70

pred. = ~~-1~~ ~~10~~ ~~20~~ ~~30~~ ~~50~~ ~~30~~ ~~70~~ ~~80~~ 110

succ = ~~-1~~ 90

state = ~~0~~ ~~1~~ 2

predessor

successor

Preorder traversal

10 20 50 60 30 70 80 110 120 90 40 100

Space → O(1)
time → O(n)

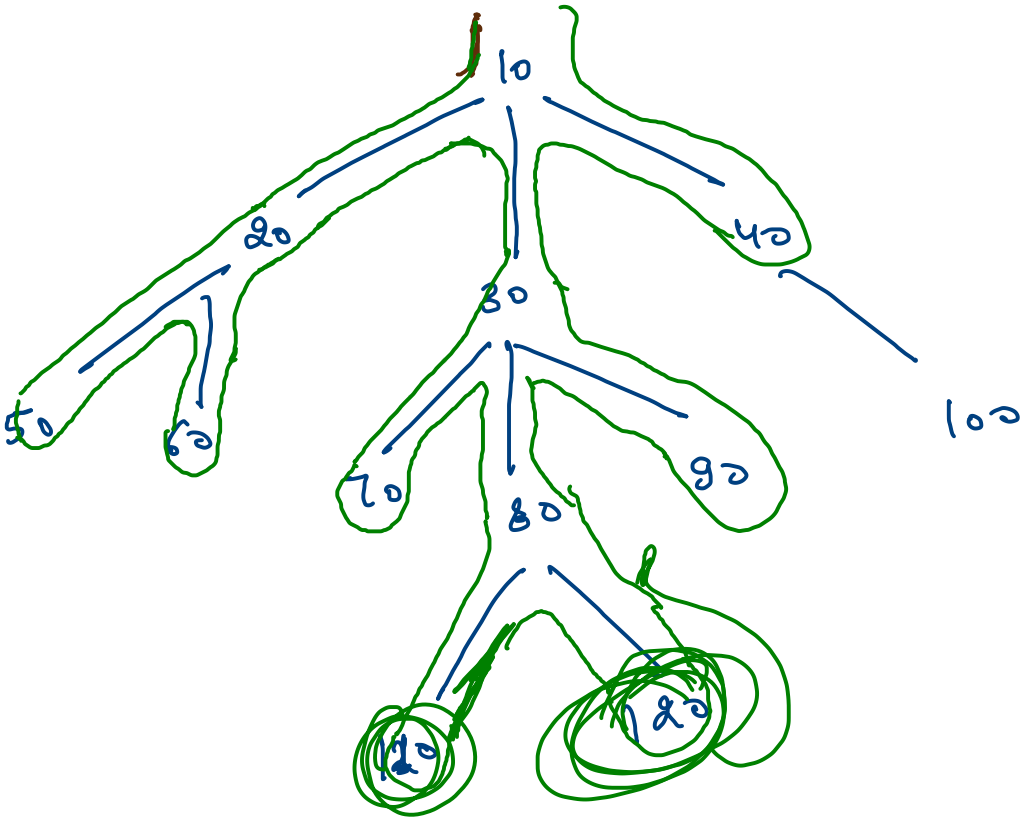
120

Node Not found
state = 0

Node found Now
state = 1

Node is Already know
state = 2

Node for which we have to find pred & succ..



Node → 110

pred : ~~10 20 50 60 30 70 80~~

succ = ~~120~~

state = ~~0~~ 2

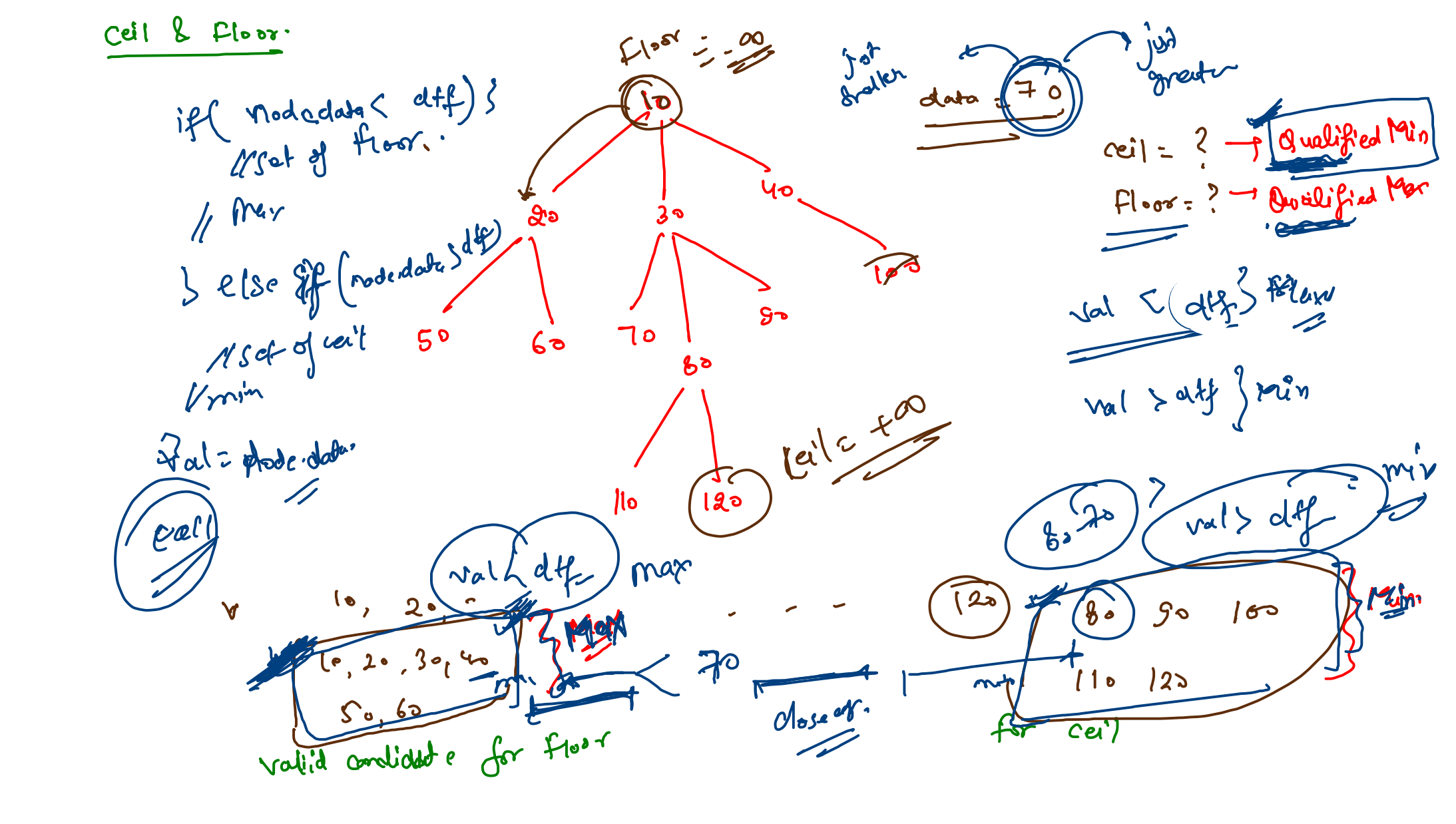
state = 0
dtf

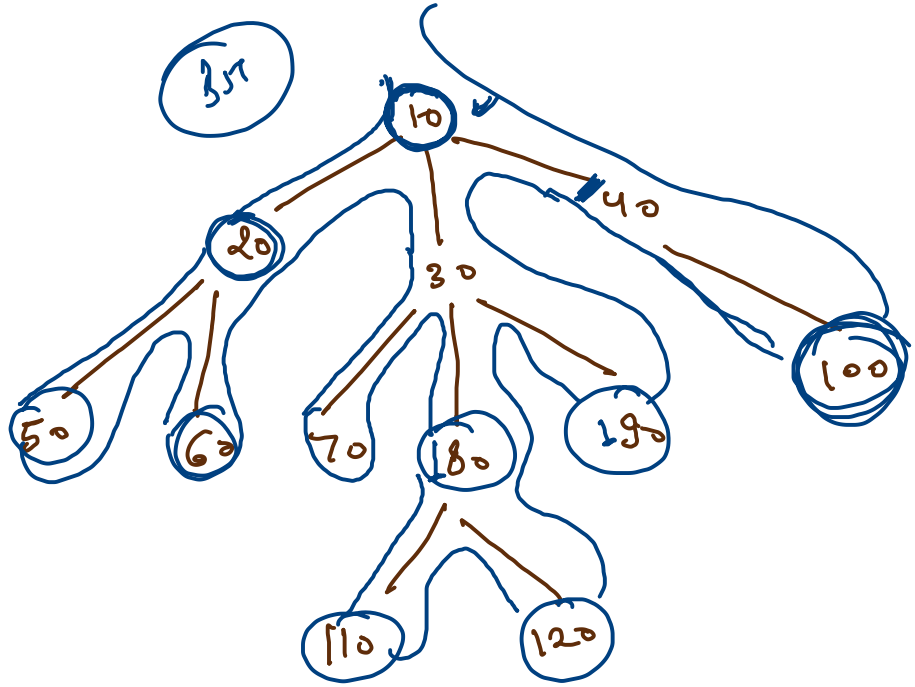
if (node.data == dtf)
{ state ++;
else
{ pred.set
}

state = 1
set succ
state

if (state == 2)
call

Ceil & Floor.


if(nodedata < diff) {
// Set of floor.
// Max
} else if (nodedata > diff)
// Set of ceil
// min
Val = node.data
Floor = -∞
10
20
30
40
50
60
70
80
90
100
110
120
just smaller
just greater
data = 70
ceil = ? → Qualified Min
Floor = ? → Qualified Max
Val < diff } floor
Val > diff } min
Ceil = +∞
ceil
val < diff
max
10, 20, 30, 40
50, 60
valid candidate for floor
70
close off.
val > diff
min
80, 90, 100
110, 120
for ceil



Initialisation

data = 70

ceil = ∞ (Min) Q.

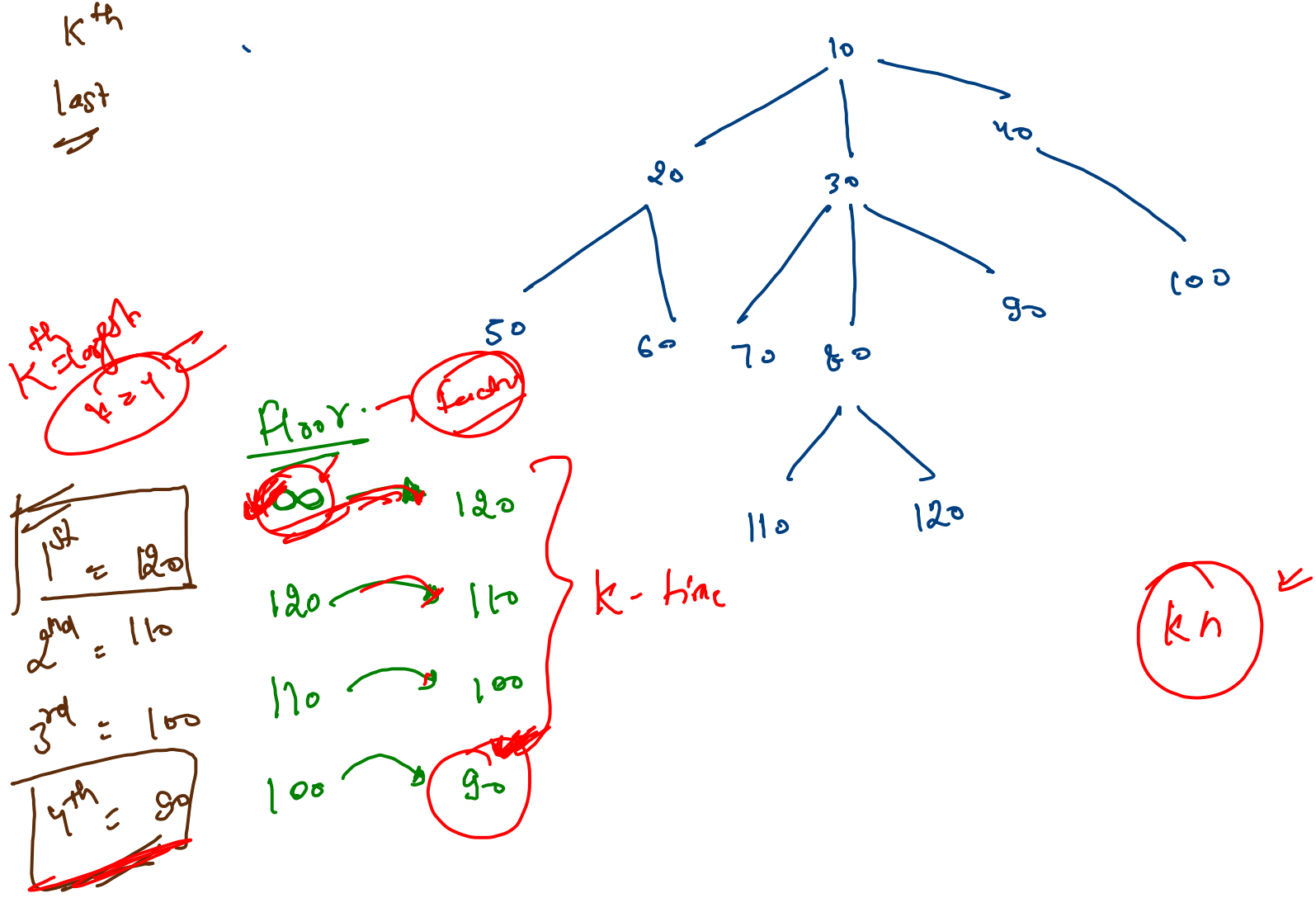
floor = −∞ (Max) Q.

Binary search {data sorted}
- Rotated sorted array

data = 70

Ceil : ~~∞~~ ~~180~~ ~~110~~ 100

floor : ~~∞~~ ~~10~~ ~~20~~ ~~50~~ 60


K th
last
10
20
30
40
50
60
70
80
90
100
110
120
K th largest
k = 1
1st = 120
2nd = 110
3rd = 100
4th = 90
floor
root
120
120
110
110
100
100
90
k - time
kn

```java
public static int kthLargest(Node node, int k){
  // write your code here

  int factor = Integer.MAX_VALUE;

  for(int i = 0; i < k; i++) {
      floor = Integer.MIN_VALUE; //Reset floor
      ceilAndFloor(node, factor);
      factor = floor;
  }
  return floor;
}
```

Factor : ∞ 120 110 100

Floor = -∞ 120 -∞ 110 -∞ 100 -∞ 90

Floor = 90