24th August 2021

1. ✓ Iterator in Generic Tree
2. ✓ Iterator in Binary Search Tree
3. ✓ Root to All leaf path in Binary Tree
4. ✓ All Single child in Binary Tree
5. ✓ Count Single Child in Binary Tree

28th August 2021 (Moring)

1. Inorder Moris Traversal
2. Pre Order Moris Traversal
3. Post order Moris Traversal
4. ✓ Iterator of Binary tree using Moris traversal

28th August 2021 (Evening)

1. Path Sum in Binary Tree 2
2. Diameter of Binary Tree (All Methods)
3. Maximum path sum in B/w two leaf
4. All nodes distance K in B.T.
5. Burning Tree

29th August 2021 (Morning)

1. Burning Tree 2
2. max. width of Binary Tree
3. Convert BST to Doubly LL
4. Convert Sorted DLL to BST
5. Path sum in Binary Tree

Iterator and Iterable

Tuesday, 24 August 2021 8:15 PM

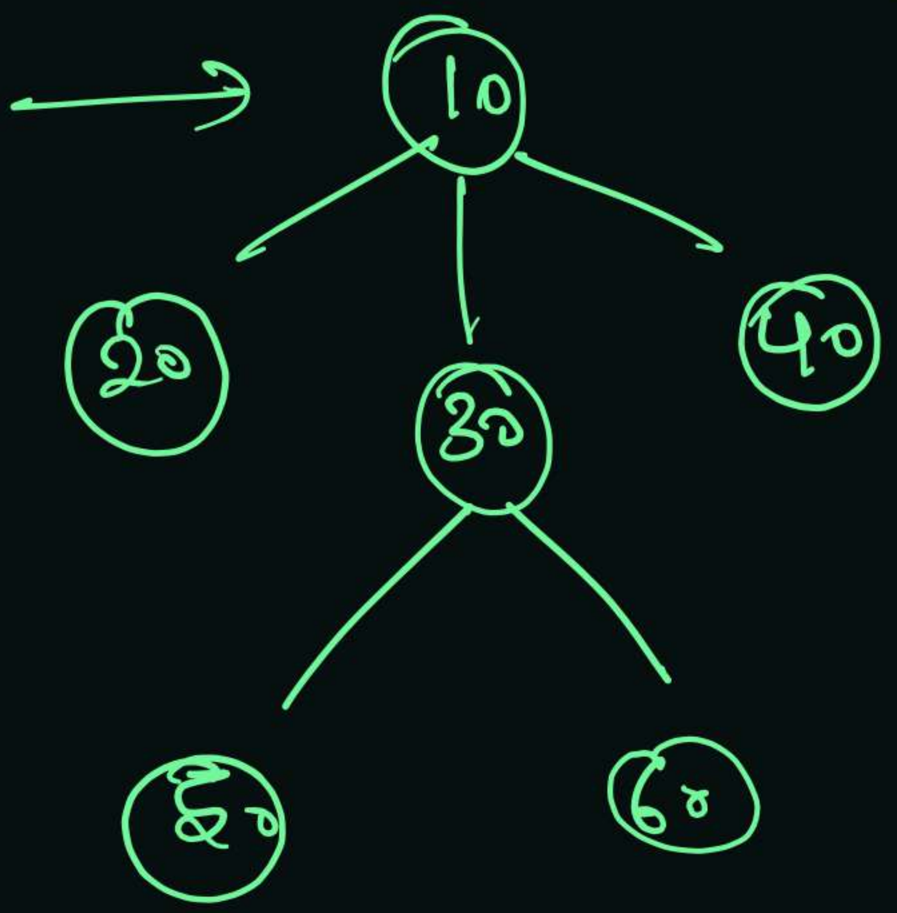
$\underbrace{\text{For Each loop}}_{\text{User's perspective}} \longrightarrow \underbrace{\text{Iterable and Iterator}}_{\text{creator's perspective}}$

```
For(int val : list) {
    System.out.println(val);      ⟹
}
```

```
Iterator<Integer> itr = list.iterator();
 while( itr.hasNext()) {
     System.out.println(itr.next());
}
```

Requirement

Gt ⟶



generictree

```
for(int val : gt){
    System.out.println(val);
}
```

preorder of Gtree ⟶ 10 20 30 50 60 40

Interface → Interface is basically contract which hold the signature of methods.

```
Interface I {
    void fun( int data1, int data2){
    }
}
class A implements I {
    ________
    ________
    void fun( int data1, int data2) {
        ________
        ________ // Body
    }
}
```

Application of Interface →

1. what is Interface → contract which holds signature of method

   $\underbrace{I}_{\text{Interface}}$ obj = new $\underbrace{A()}_{\text{class which implements Interface}}$;

2. There is an Iterface called Iterable implemented by Generic Tree Class

Iterable → method → iterator()

implements Iterable< Integer> → method → iterator()

Return type of iterator.

```
public Iterator<Integer> iterator() {

}
```

3. Another class implement Iterator

Iterator → hasNext()
→ next

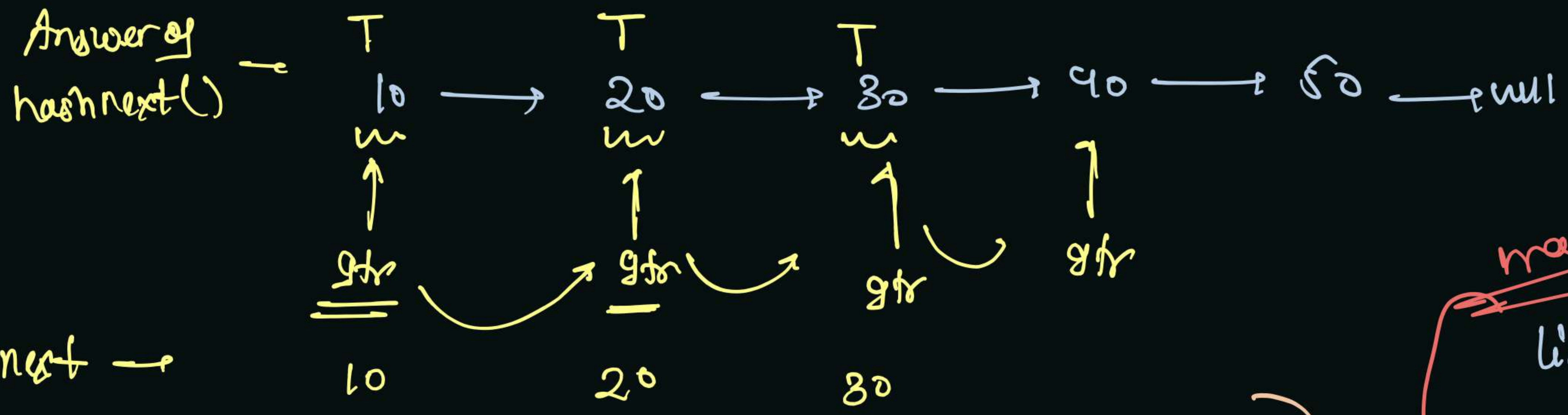
Iterator <Integer> itr = new Class ();

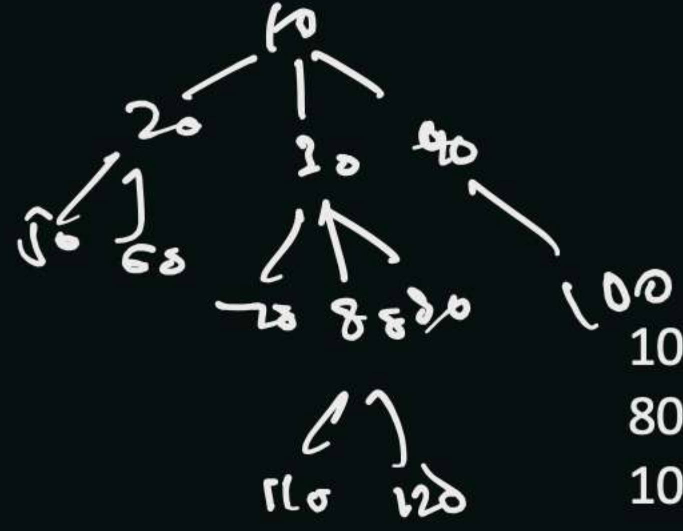
Class which is impled Iterator

4. Iterator interface → hasNext() → If value is available → return true
otherwis → False

next() → 1. → return current value and move toword next.

Iterator in LinkedList →





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

```
Node itr = head;

public boolean hasNext() {
    if(itr == null) return false
    else
     return true;
}
```

```
public int next() {
    int val = itr.data;
    itr = itr.next;

    return val;
}
```

class which is implementing iterator

main→

```
LinkedList list = new Li----

Iterator<Integer> itr = list.iterator();
                 True.
while( list.hasNext()) {
  system.out.println(list.next);

}
```

```
public Iterator<Integer> iterator() {
    Iterator<Integer> itr = new GTPreorderIterator(root);
    return itr;
}
```

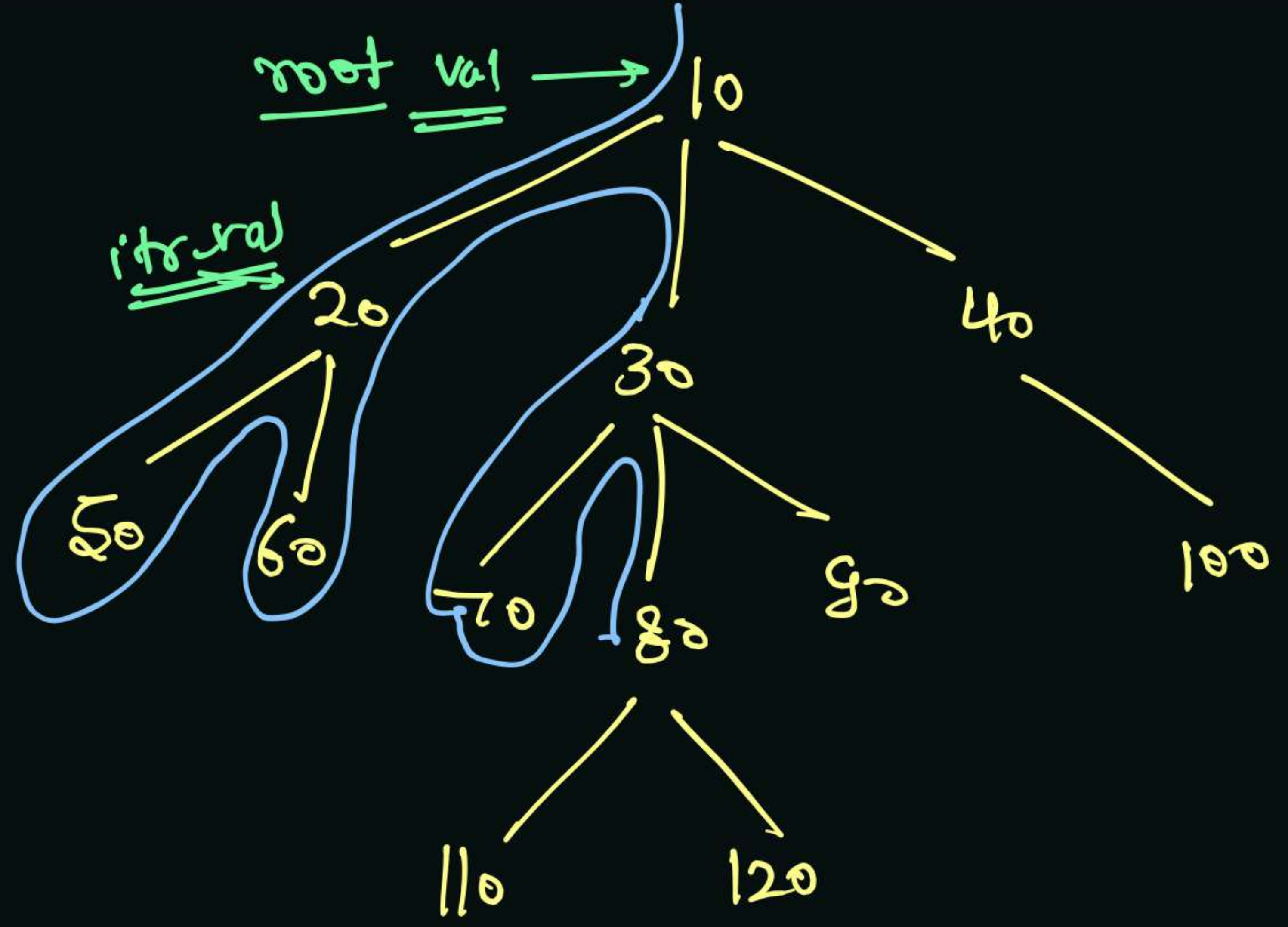
```
public GTPreorderIterator(Node root) {
    st = new Stack<>();
    st.push(new Pair(root, 0));
    next();
}

public boolean hasNext() {
    if(itr_val == null) return false;
    else return true;
}
```

```
Iterator<Integer> itr = gt.iterator();
while(itr.hasNext()) {
    System.out.print(itr.next() + " ");
}
```

```
public Integer next() {
    Integer val = itr_val;
    itr_val = null;
    while(st.size() > 0) {
        Pair top = st.peek();
        if(top.state == 0) {
            itr_val = top.node.data;
            top.state++;
            break;
        } else if(top.state >= 1 && top.state <= top.node.children.size()) {
            Node child = top.node.children.get(top.state - 1);
            st.push(new Pair(child, 0));
            top.state++;
        } else {
            st.pop();
        }
    }
    return val;
}
```

PreOrder: 10 20 50 60



30, ~~0~~ 1

10, ~~0~~ ~~1~~ ~~2~~ 3

val = ~~null~~ ~~10~~ ~~20~~ ~~50~~ 60

itr = ~~null~~ ~~10~~

~~null~~ ~~20~~

~~null~~ ~~50~~

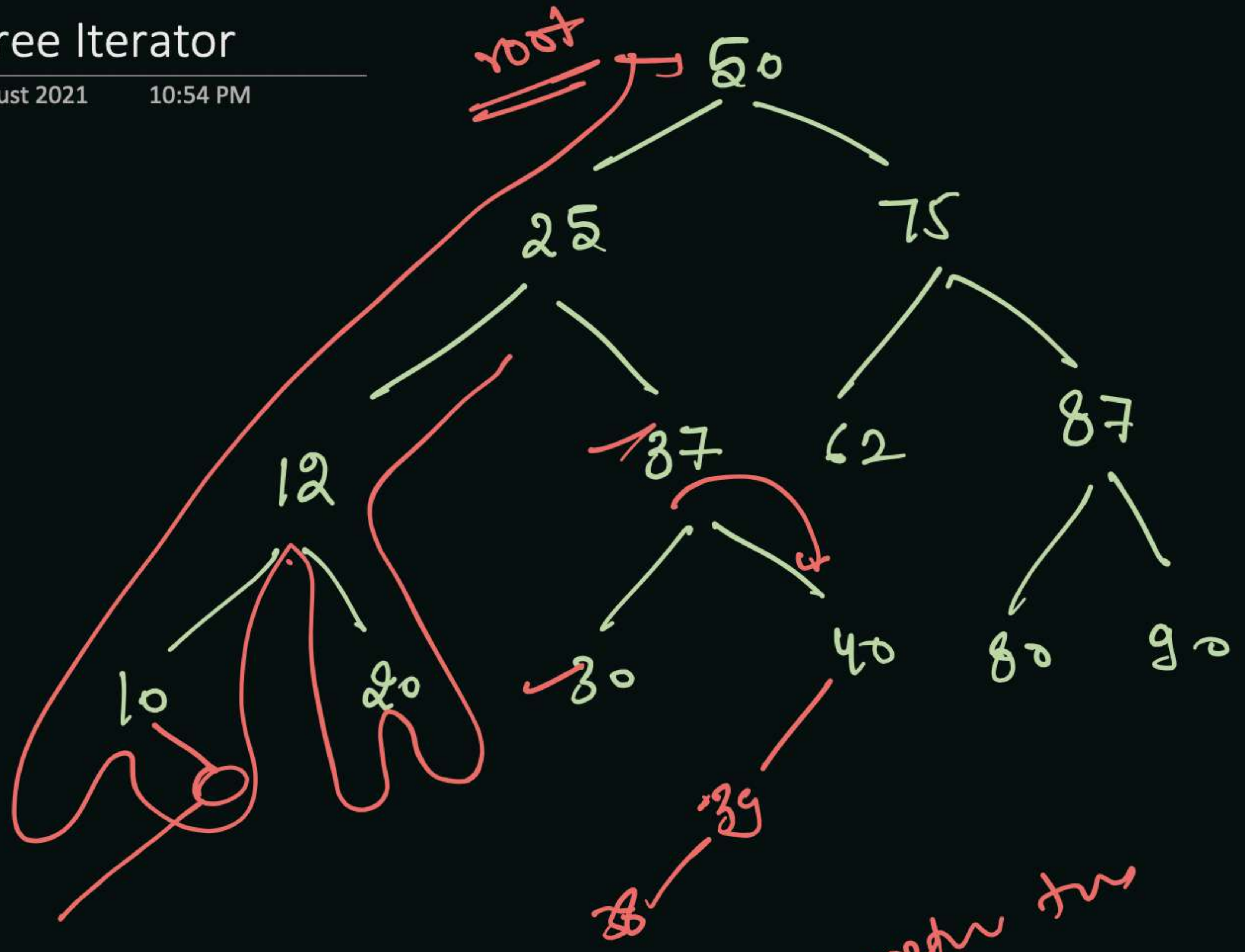
~~null~~ ~~60~~

~~null~~ 30

. . . . . .

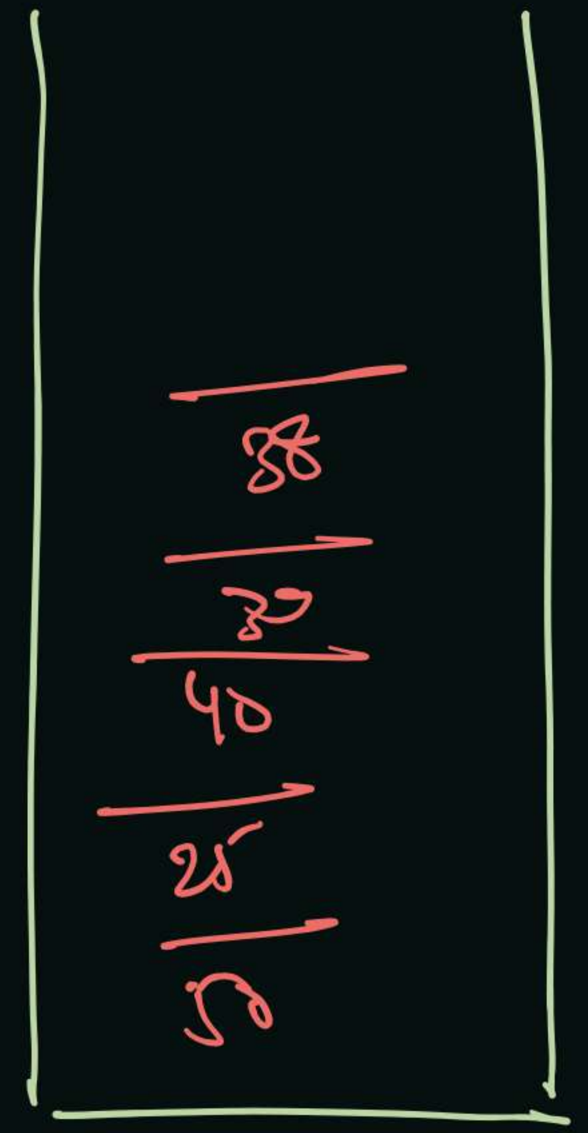
Binary Tree Iterator

Tuesday, 24 August 2021 10:54 PM



next() →

hasNext() → st.size() > 0 → return true
else → false



Method
↓
optimised way of Stack

(looks easier and easy to digest)

return 10 12 20 25 38 37

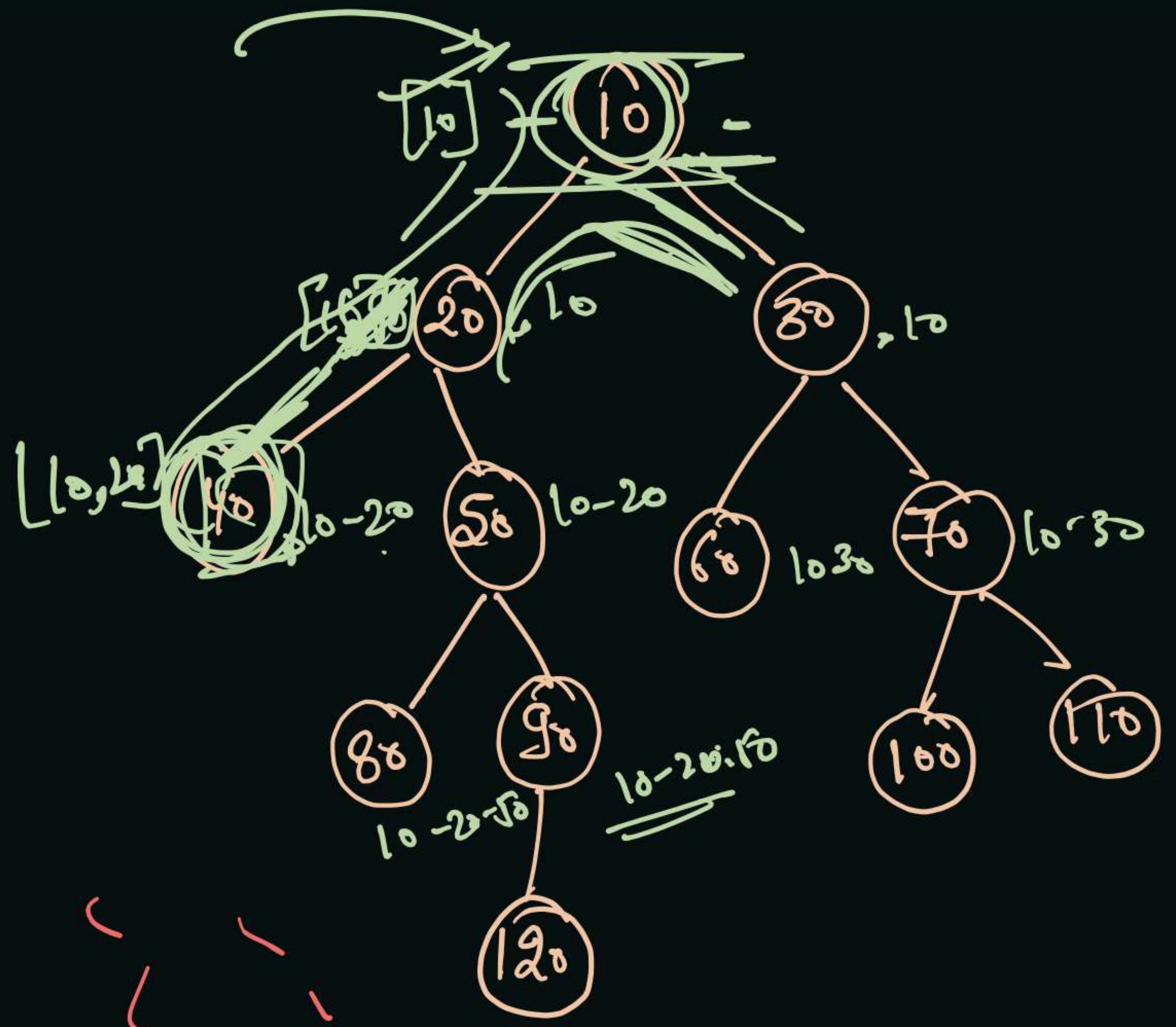
Root to all Leaf

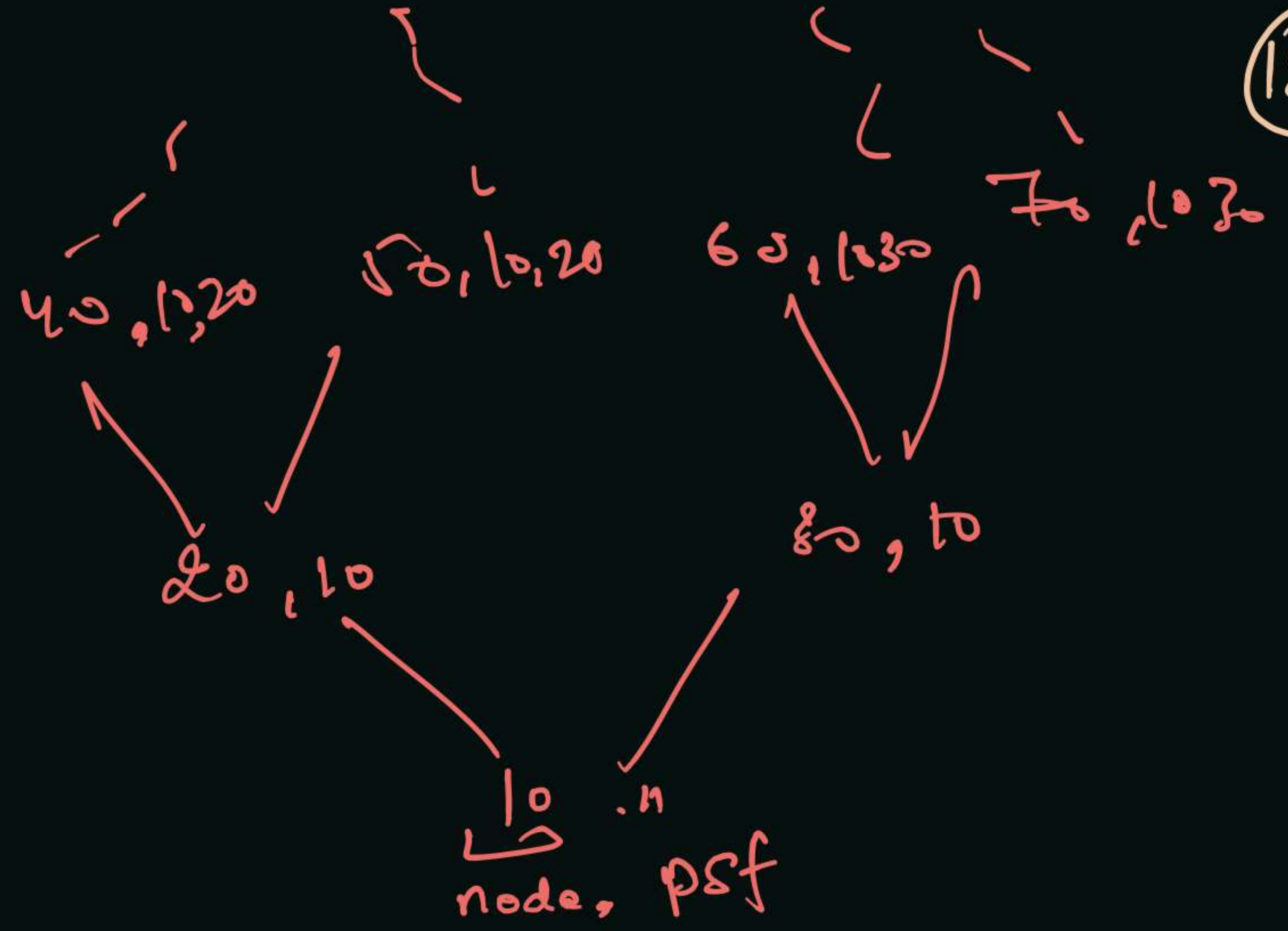
Tuesday, 24 August 2021 11:27 PM

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

res [

]





Euler
Diagram

res = [~~[[10, 20, 40]~~, [10, 20, 50],

subres = [10, 20, 40, 50, 30

1k

At leaf

Don't Do this

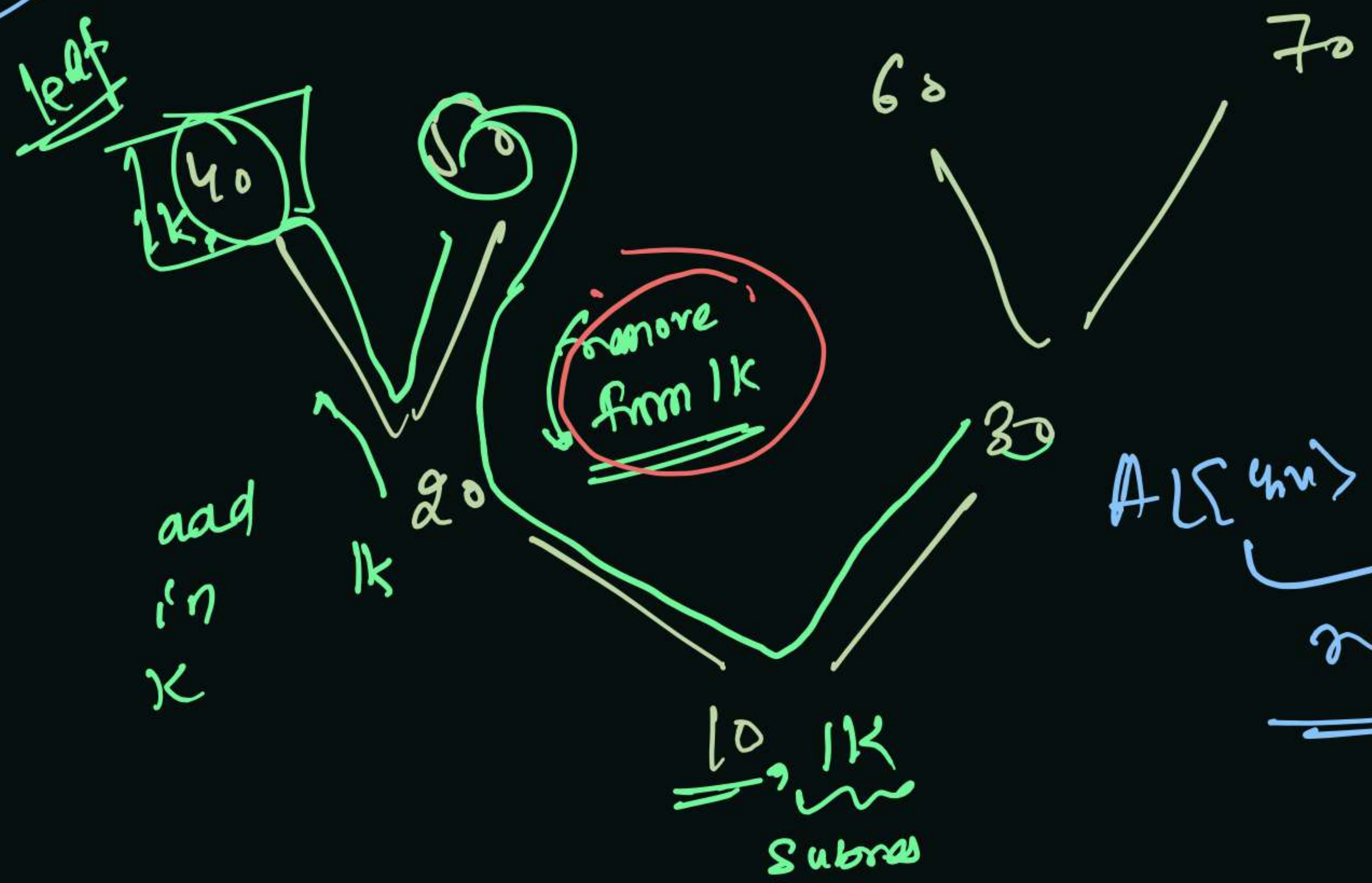
~~res.add(subres);~~

subres.remove

At base case which is leaf node

make a new array list

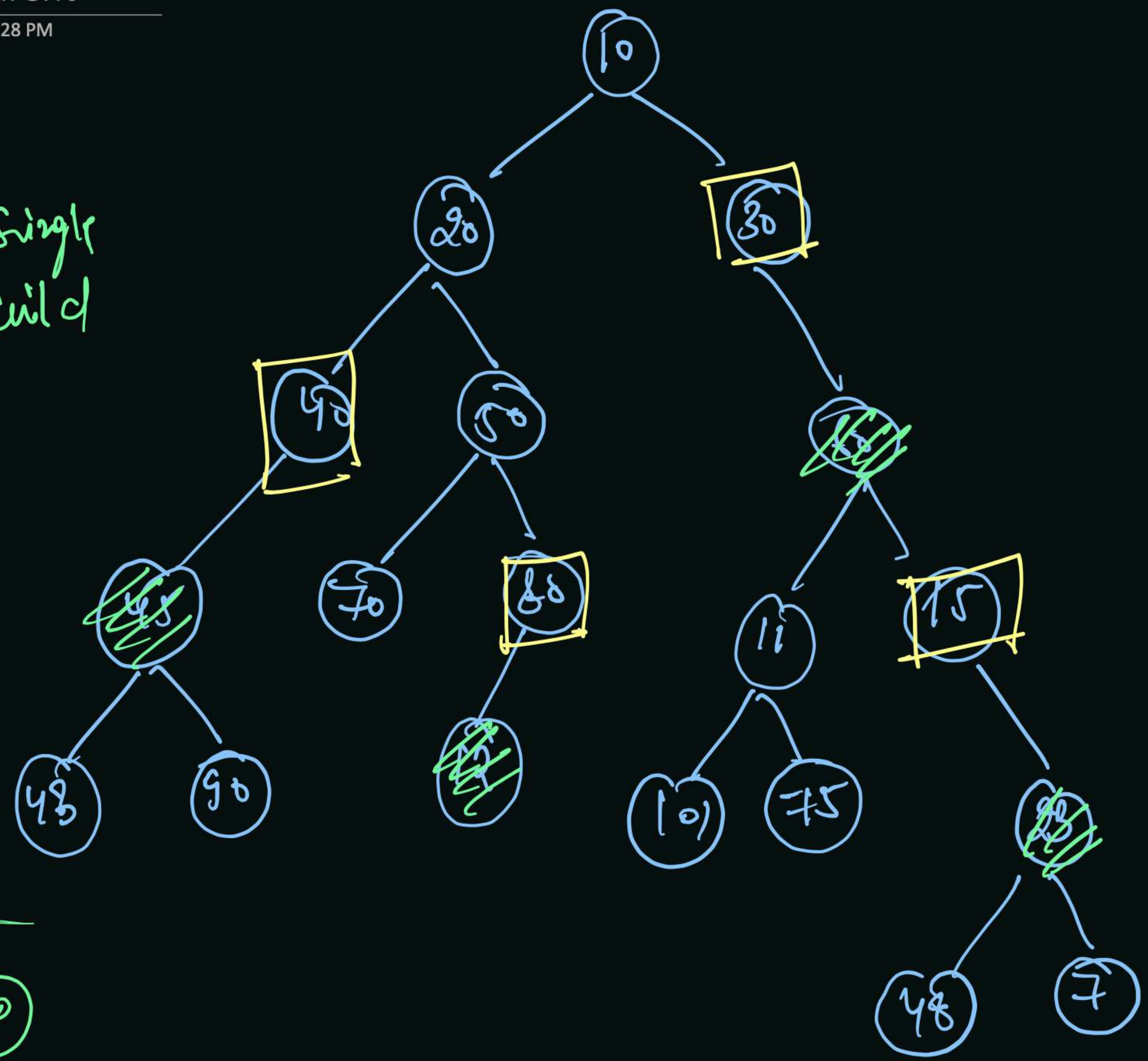
~~AL<Int> temp = subres;~~
~~res.add(temp);~~

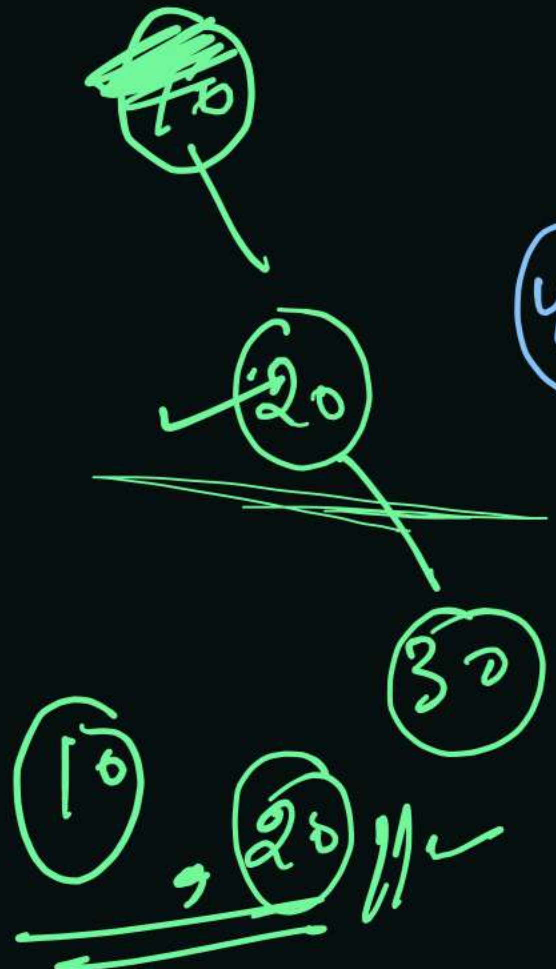


All Single Child Parent

Tuesday, 24 August 2021 11:28 PM

Green - Single Child





if(root == null

||

leaf

skip

root.left == null ||

root.right == null

Single child parent