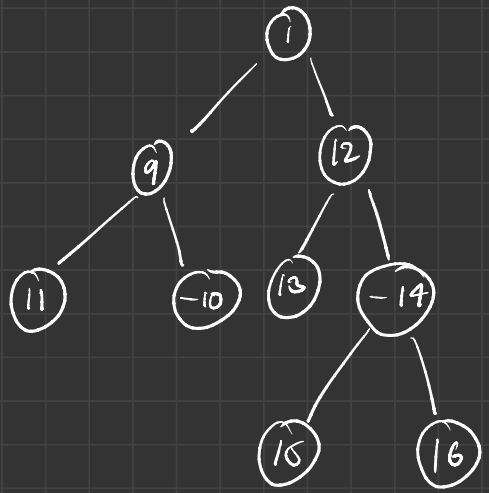
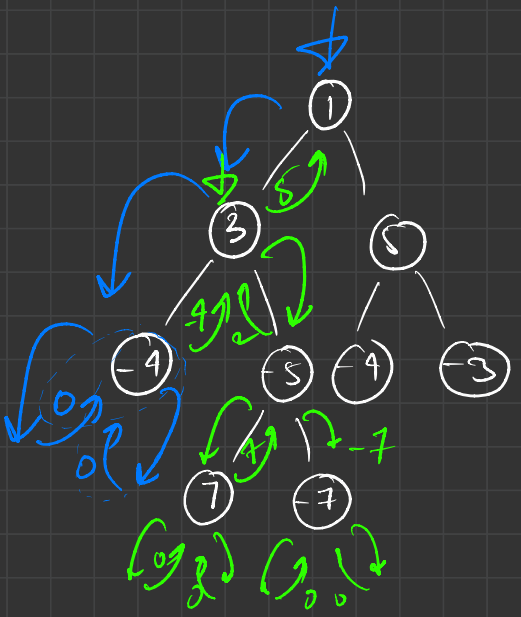
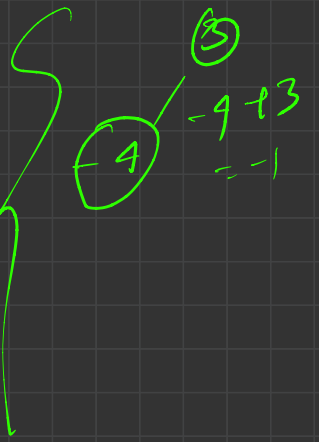
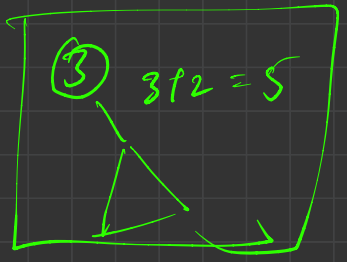
Binary Tree Maximum Path Sum

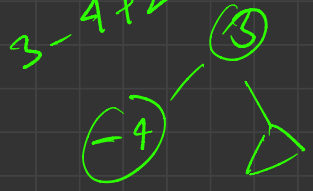


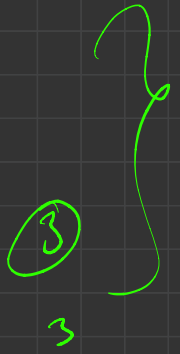
```java
public int BestPathSum(TreeNode root) {
    if (root == null) {
        return 0;
    }

    int bestPathSumFromLeft = Math.max(BestPathSum(root.left), 0);
    int bestPathSumFromRight = Math.max(BestPathSum(root.right), 0);

    return Math.max(bestPathSumFromLeft + root.val, bestPathSumFromRight + root.val);
}
```











Pair class {

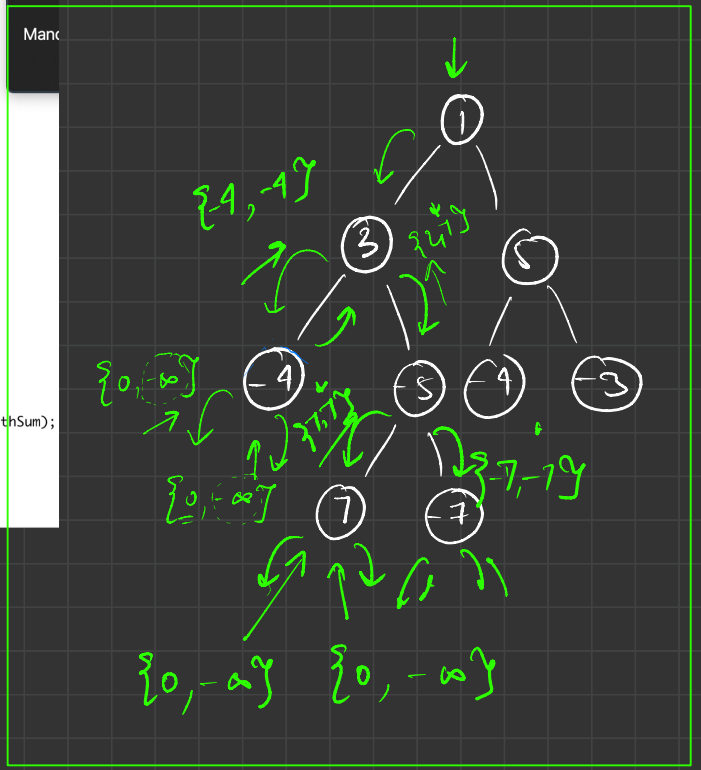
max PathSum;

Sum

```java
class Pair {
    int maxPathSum = 0;
    int sum = 0;

    Pair (int sum, int maxPathSum) {
        this.sum = sum;
        this.maxPathSum = maxPathSum;
    }
}

public Pair BestPathSum(TreeNode root) {
    if (root == null) {
        return new Pair(0, Integer.MIN_VALUE);
    }

    Pair LeftPair = BestPathSum(root.left);
    Pair RightPair = BestPathSum(root.right);

    int bestPathSumFromLeft = Math.max(LeftPair.sum, 0);
    int bestPathSumFromRight = Math.max(RightPair.sum, 0);
    int currMaxPathSum = bestPathSumFromLeft + root.val + bestPathSumFromRight;

    int overallMaxPathSum = Math.max(currMaxPathSum, Math.max(LeftPair.maxPathSum, RightPair.maxPathSum));

    return new Pair(Math.max(bestPathSumFromLeft + root.val, bestPathSumFromRight + root.val), overallMaxPathSum);
}

public int maxPathSum(TreeNode root) {
    return BestPathSum(root).maxPathSum;
}
```

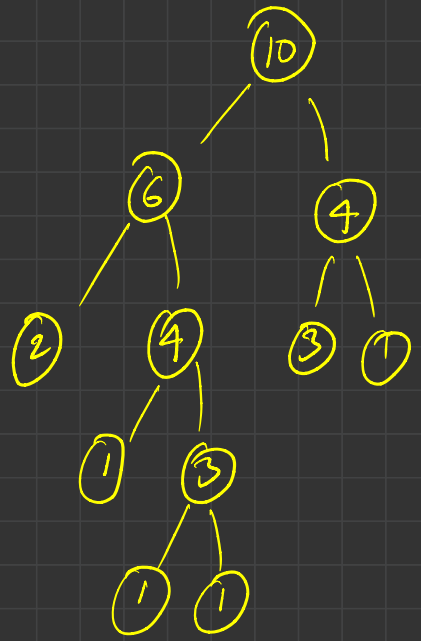
TC: O(N)

SC: O(ht)



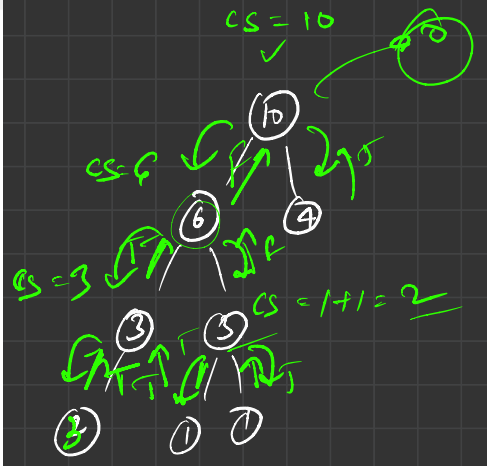
Children Sum Property

def Node
is holds this property true



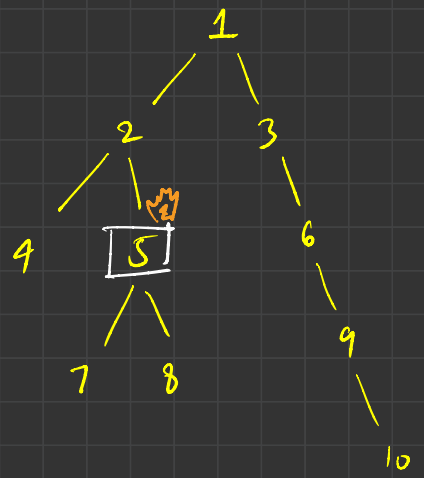
NOTE

```java
//Function to check whether all nodes of a tree have the value
//equal to the sum of their child nodes.
public static int isSumProperty(Node root)
{
    // add your code here
    if (root == null) {
        return 1;
    }

    // if node is a leaf node then return true
    if (root.left == null && root.right == null) {
        return 1;
    }

    int childSum = 0;
    if (root.left != null) {
        childSum += root.left.data;
    }
    if (root.right != null) {
        childSum += root.right.data;
    }

    int isFollowedInLeftSubTree = isSumProperty(root.left);
    int isFollowedInRightSubTree = isSumProperty(root.right);

    if (isFollowedInLeftSubTree == 0 || isFollowedInRightSubTree == 0 || childSum != root.data) {
        return 0;
    } else {
        return 1;
    }

}
```

TC: O(N)
SC: O(h)

Burning of a Binary tree

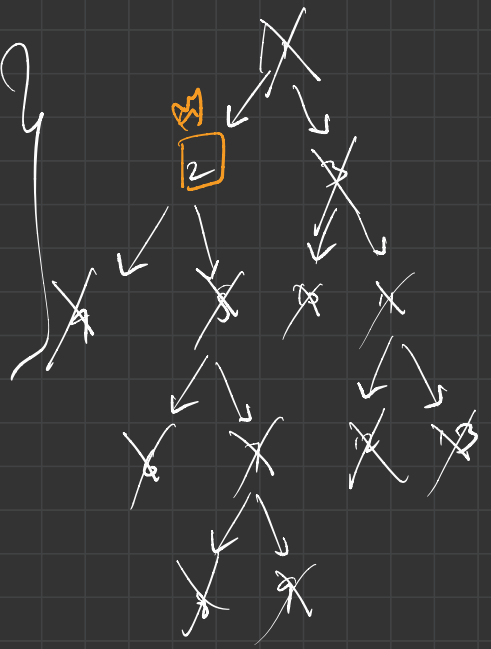


① Rotten Oranges

② K distance Away from a target in BT

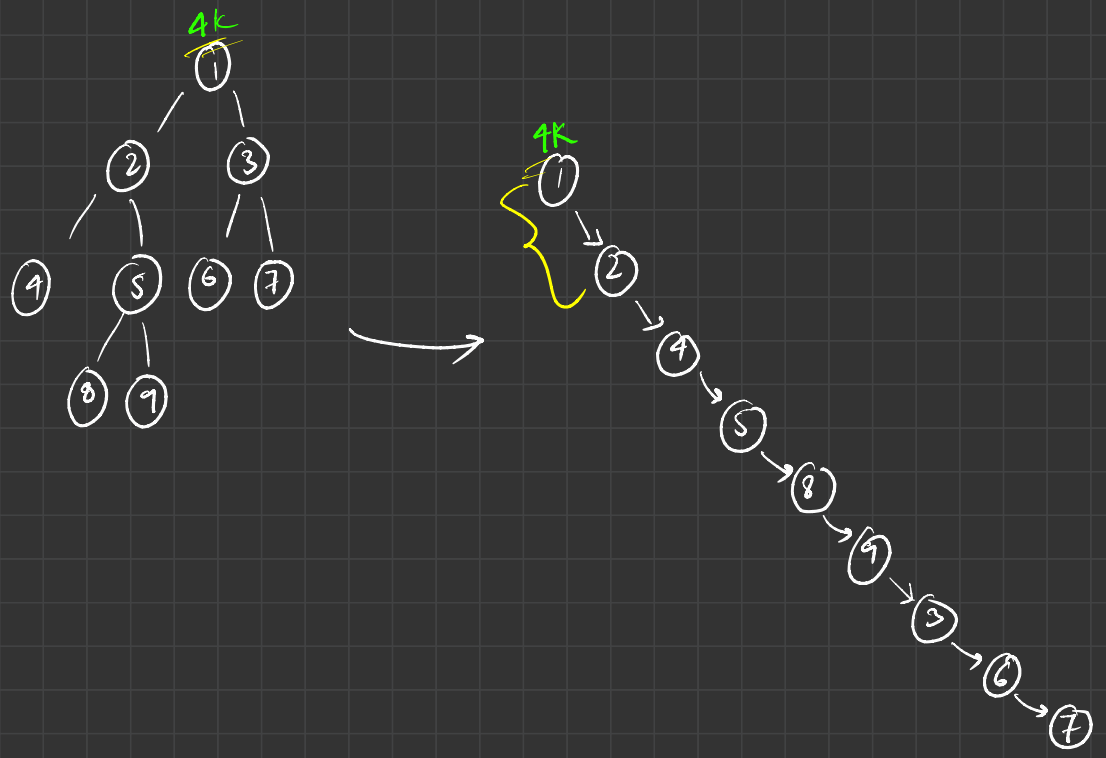
t=1 | 2
t=2 | 9, 5, 1
t=3 | 6, 7, 3
t=4 | 8, 9, 10, 11
t=5 | 12, 13

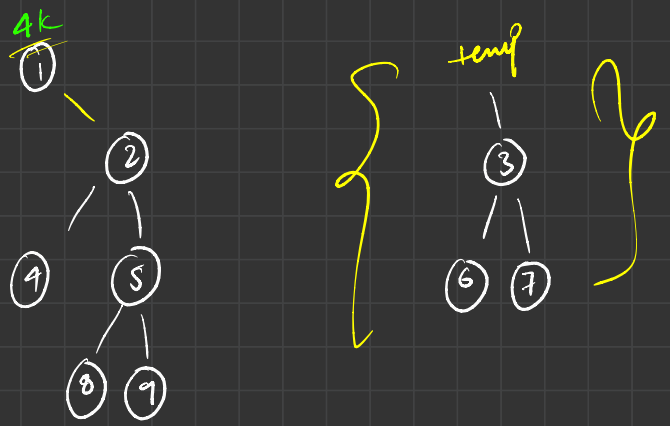
↳ 9

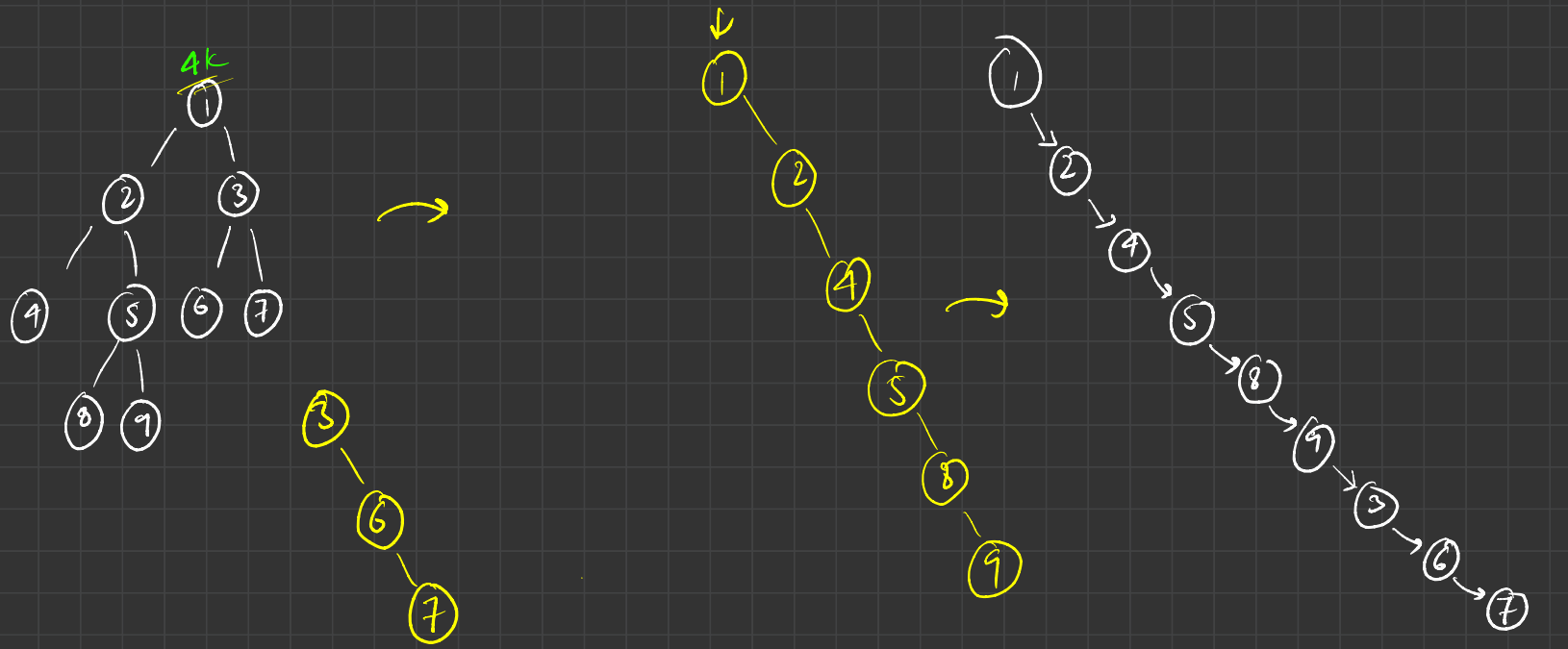


5 levels → time = level − 1

flatten Binary tree to a linked list



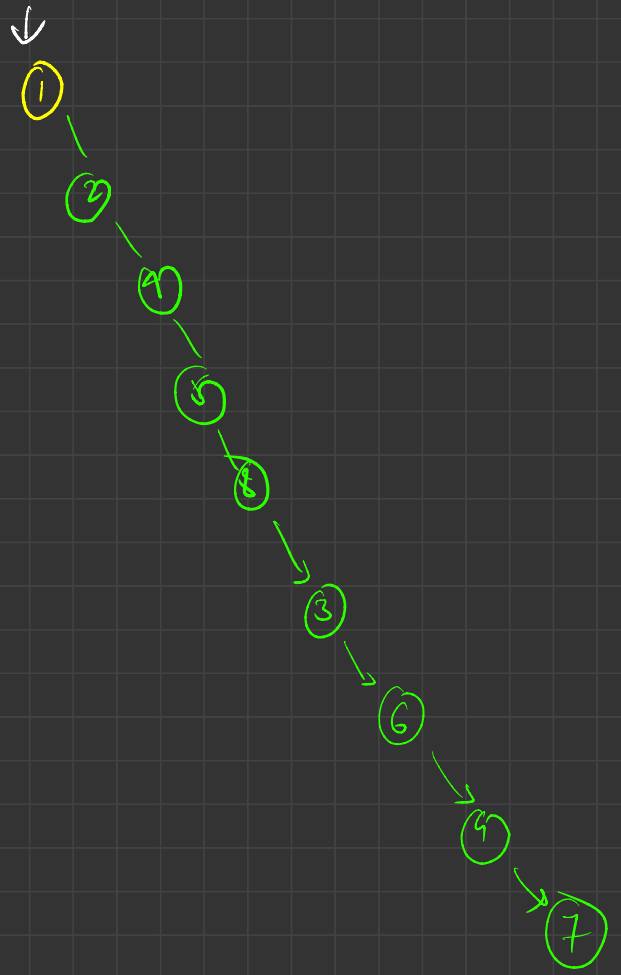

4k
1
2
4
5
8
9
temp
3
6
7

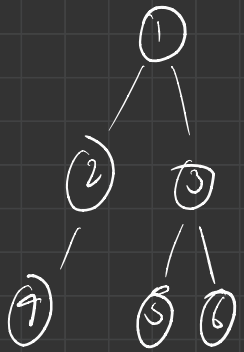

4k
1
2
3
4
5
6
7
8
9
3
6
7
1
2
4
5
8
9
1
2
4
5
8
9
3
6
7

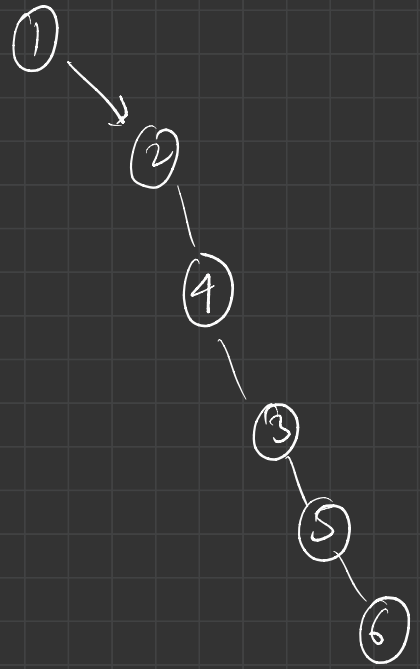
```java
// TC: O(N^2)
public void flatten(TreeNode root) {
    if (root == null) {
        return;
    }

    flatten(root.left);
    flatten(root.right);

    TreeNode temp = root.right;
    root.right = root.left;
    root.left = null;
    TreeNode temp2 = root;
    while (temp2.right != null) {
        temp2 = temp2.right;
    }
    temp2.right = temp;
}
```

TC: O(N²)

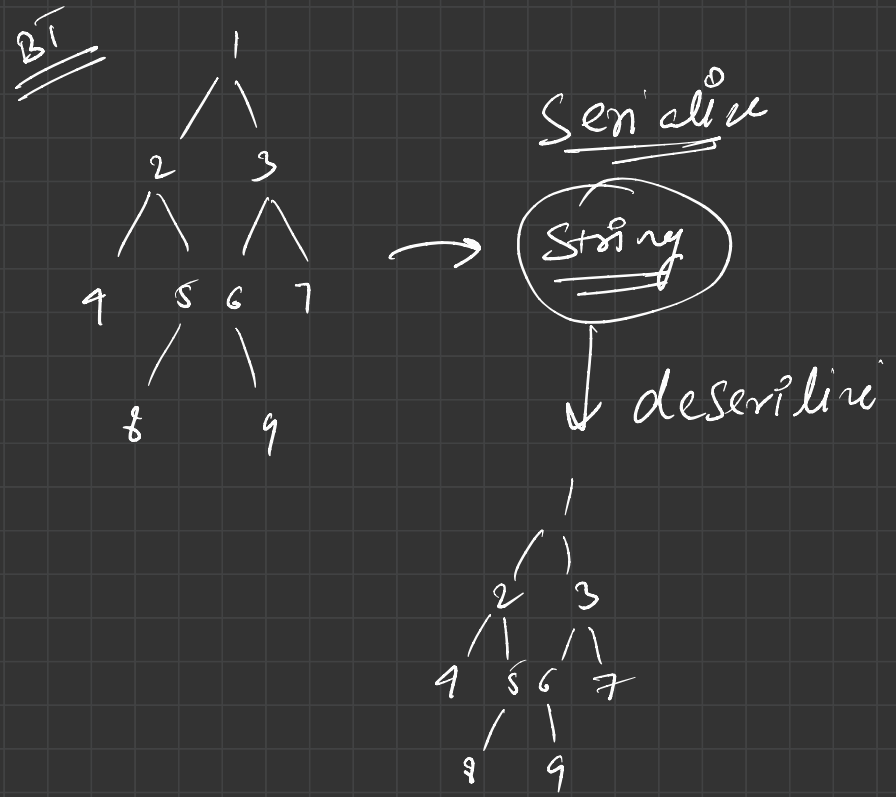
SC: O(h)




1
2
3
4
5
6


1
2
4
3
5
6

Serialize & DeSerialize A BT



{1, 2, 4, N, N, N, 3, 5, N, N, 6, N, N}