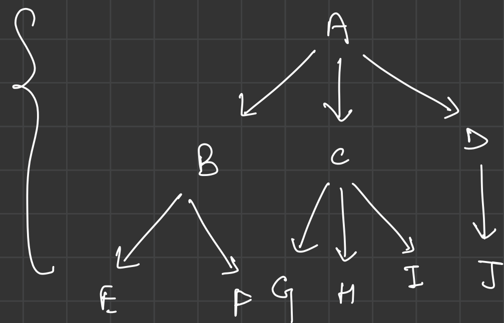
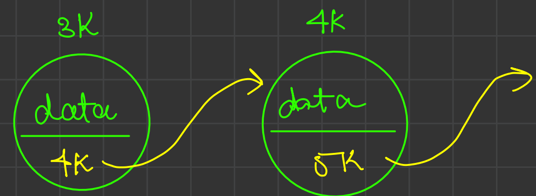
Binary Trees

Store your office employees data!

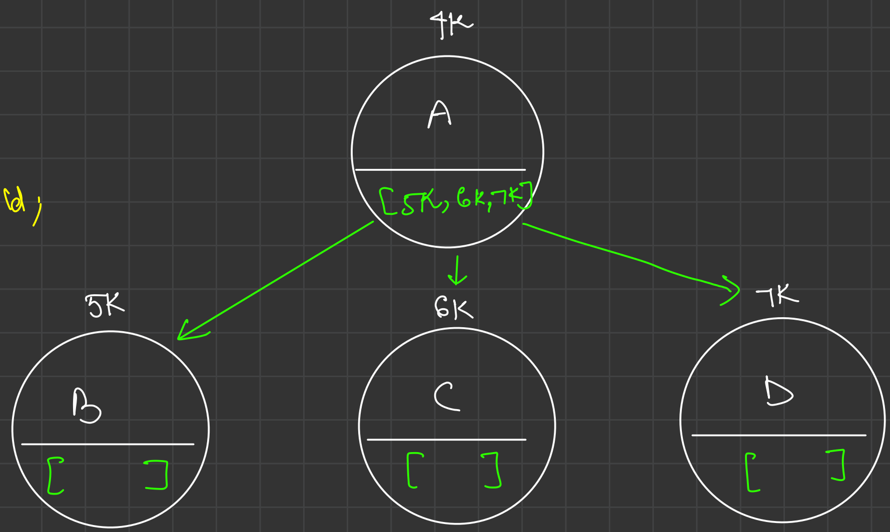
Trees!



linked list




```
class Node {

    int data;

    Node[] child;

}
```

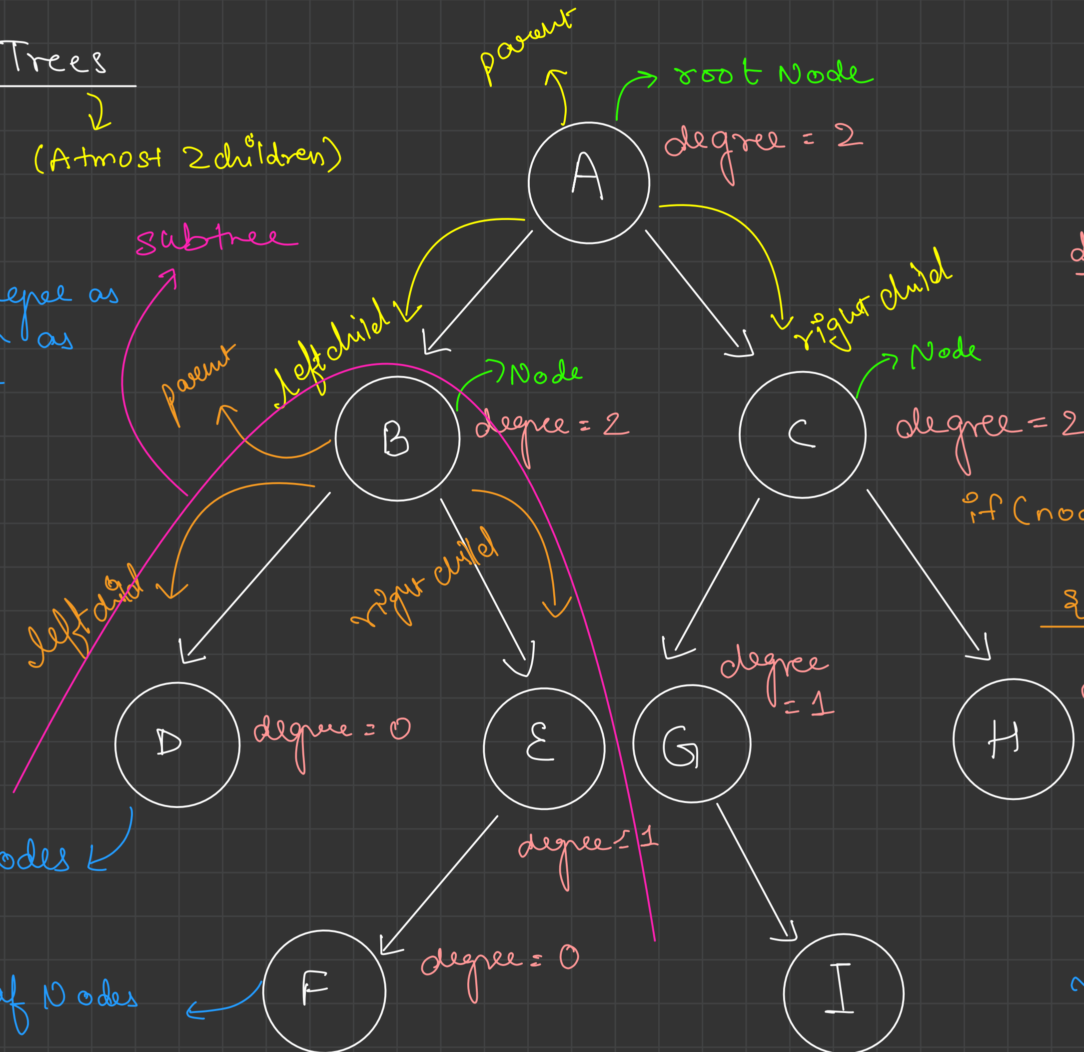


Binary Trees

(Atmost 2 children)

Nodes with degree as zero is known as leaf Node

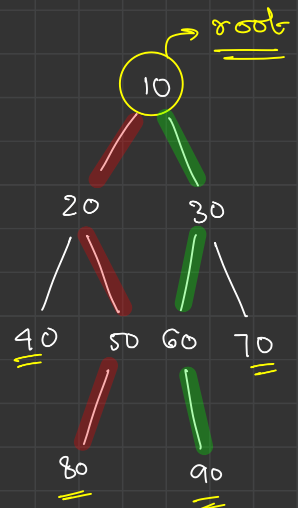
siblings
- B, C
- D, E
- G, H



degree:
No. of children

```
if (node.left == null &&
    node.right == null)
  { leaf Node }
```

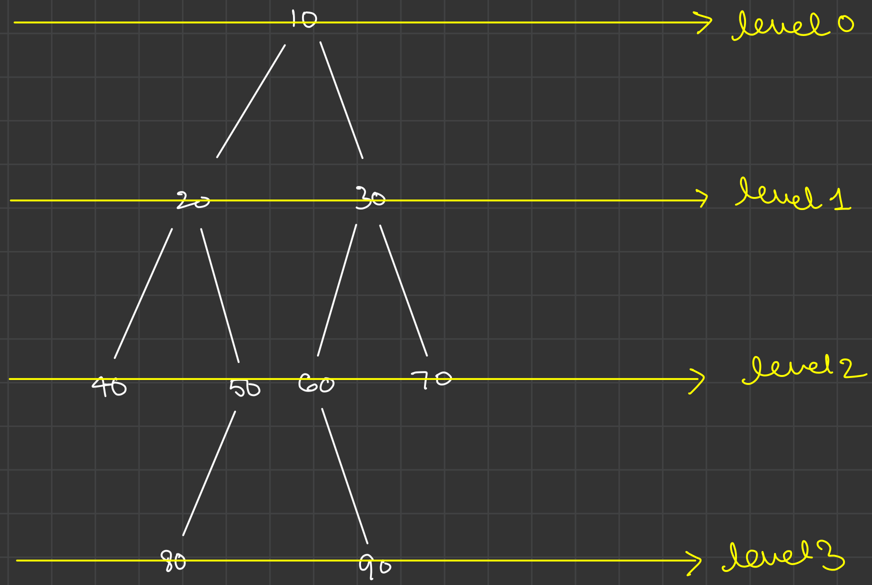
```
class Node {
  int data;
  Node left;
  Node right;
}
```



height of a Binary Tree

{ dist. b/w root node and the deepest leaf node in terms of edges }

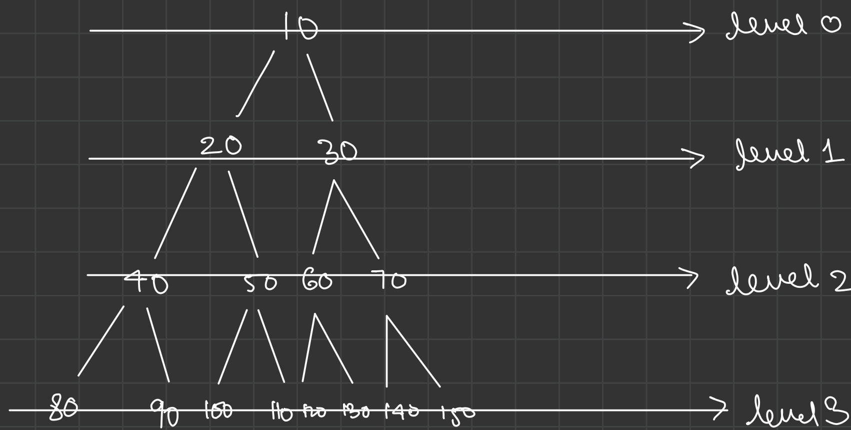
$\boxed{\text{height} = 3}$



10
level 0
20
30
level 1
40
50
60
70
level 2
80
90
level 3

Perfect Binary Tree

{A tree where no. of nodes a level h is 2^h}



{1 Node} → 2^0

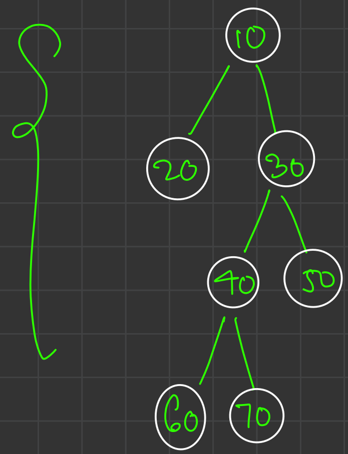
{2 Nodes} → 2^1

{4 Nodes} → 2^2

→ 2^3

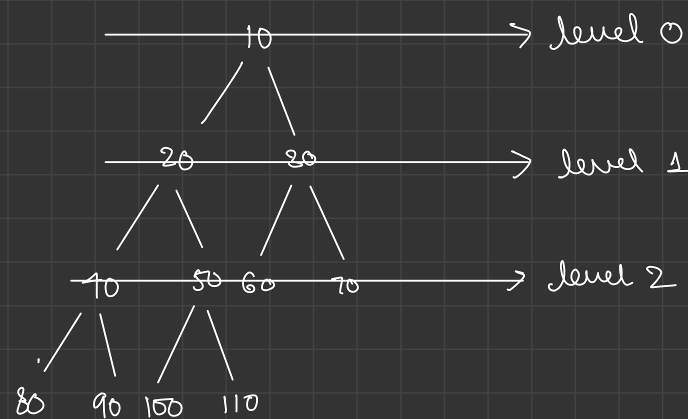
Full Binary Tree

→ where each has either zero or two children



Complete Binary Tree

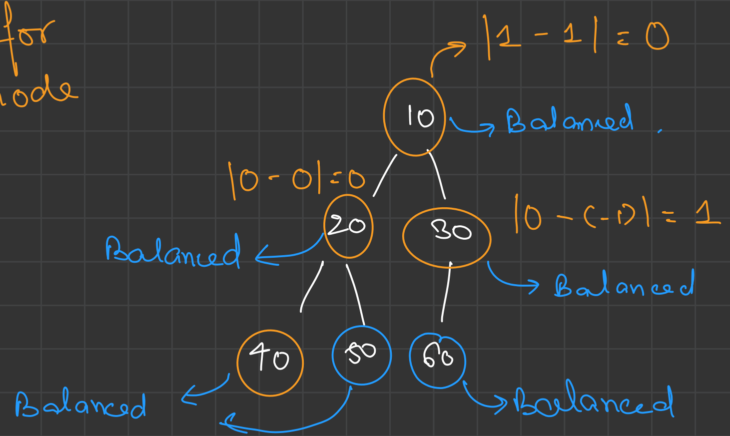
where each level is completely filled, except the last level, and nodes in last level are as left positioned as possible.

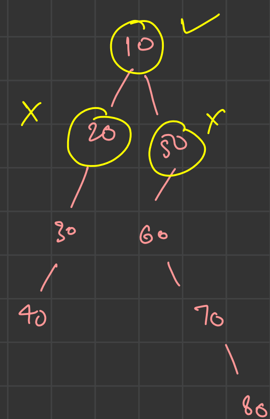


Balanced Binary Tree

$$\left| \text{height of left Subtree} - \text{height of right Subtree} \right| \leq 1$$

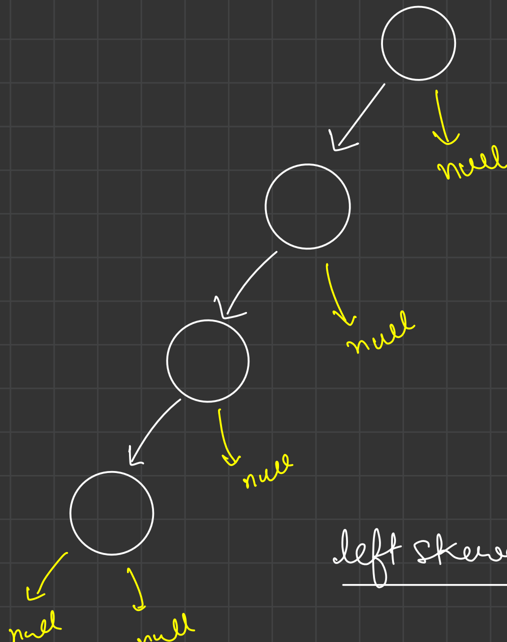
valid for each node



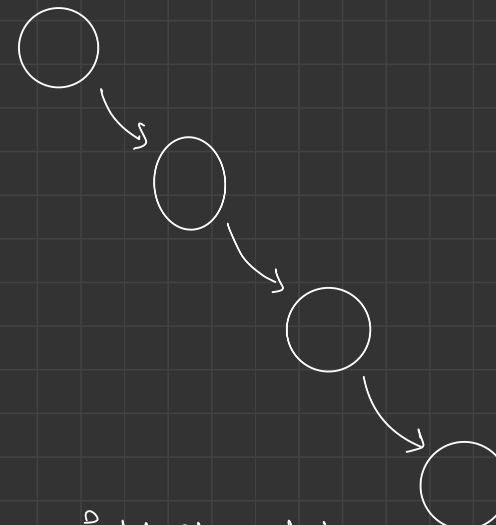


Hence this is a balanced Binary tree

Skew Tree



left skewed tree

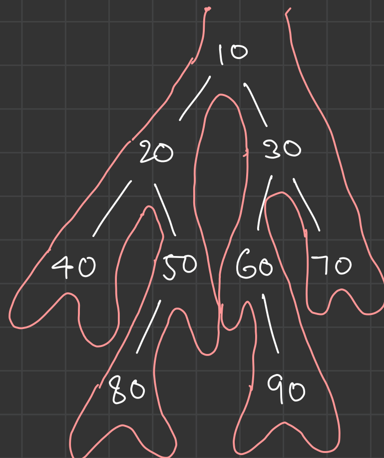


right skewed tree

Traversal over trees

① pre-order traversal

print()
left
right



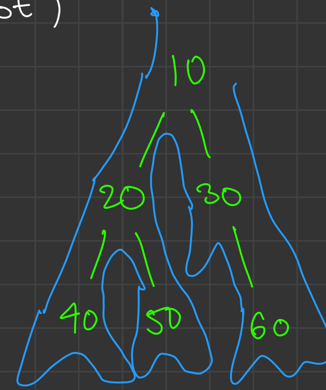
o/p

10 20 40 50 80 30 60 90 70

root — left Subtree — right Subtree

root — left Subtree — right Subtree

```
void preOrder( TreeNode root)
{
  ① if (root == null)
         return;

  ② print(root.data);
  ③ preOrder(root.left);
  ④ preOrder(root.right);
}
```



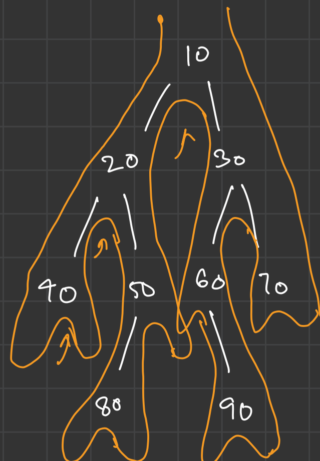
10, 20, 40, 50, 30, 60

o/p 10 20 40 50 30 60

Call Stack

In order traversal

call left
print
call right



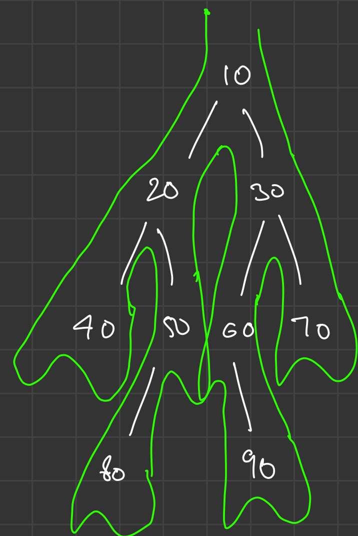
o/p

40 20 80 50 10 60 90 30 70

40 20 80 50 10 60 90 30 70

Post order traversal

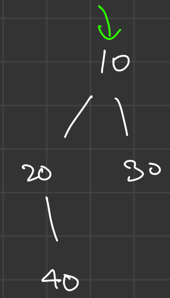
call left
call right
print (root)



o/p 40 80 50 20 90 60 70 30 10

40 80 50 20 90 60 70 30 10

```java
class Solution {
    // TC: O(N), SC: O(H)
    public static void postorderTraversal(Node root) {
        //Write your code here
        // base case
        if (root == null) {
            return;
        }

        // call left
        postorderTraversal (root.left);

        // call right
        postorderTraversal (root.right);

        // print root value
        System.out.print(root.data + " ");
    }
}
```



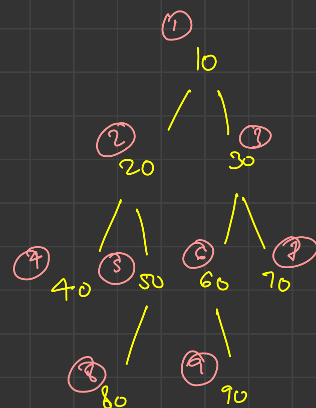
40 20 30 10

Call Stack

40 20 30 10

Size of a Binary tree

→ (No. of nodes of Binary tree)



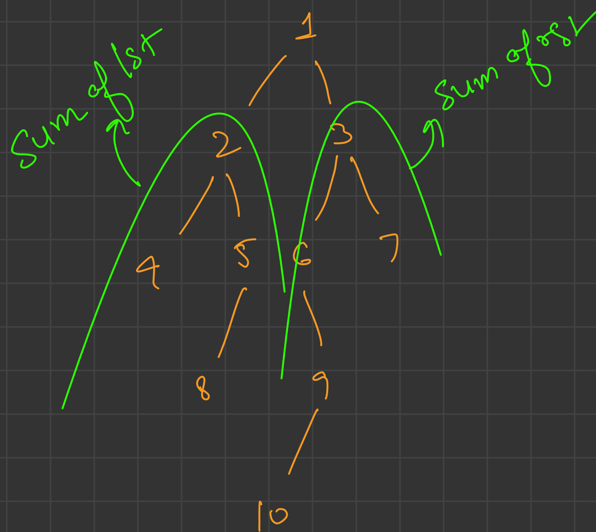
$\boxed{\text{Size} = 9}$

```
int size (root)
{ if (root == null) return 0;

  int lstSize = size(root.left);
  int rstSize = size(root.right);

  return lstSize + 1 + rstSize;
}
```

Sum of tree

→ Sum of all the nodes of a tree

sum of lst + root.val + sum of rst



Sum = 55

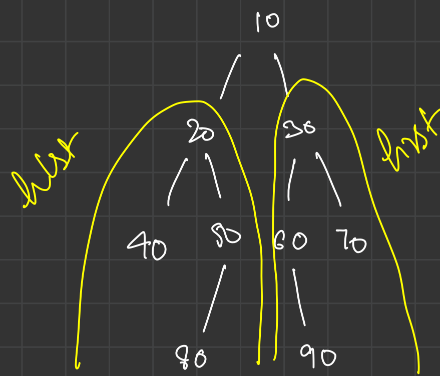
Max of tree

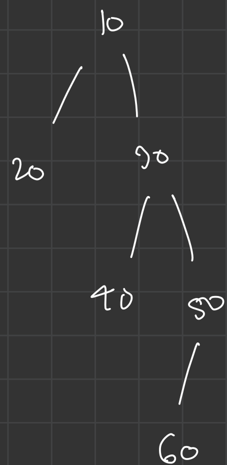
→ Max^m value in a tree

max(lmax, rmax, root.val)

height of tree

max(hlst, hrst) + 1



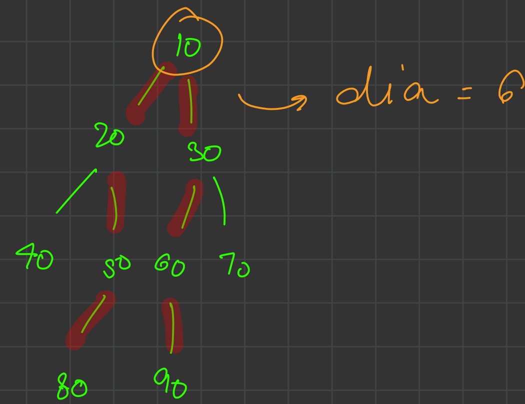


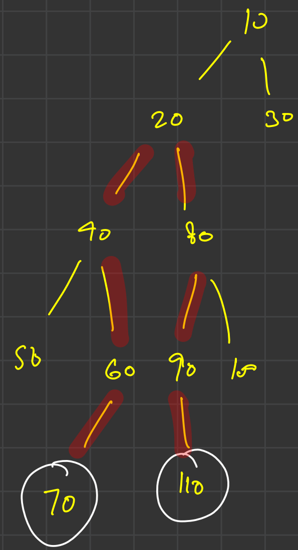
h = 3

```
public int heightOfTree(Node root) {
1    if (root == null) {
        return -1;
    }

2    int leftHeight = heightOfTree(root.left);
3    int rightHeight = heightOfTree(root.right);

4    int height = Math.max(leftHeight, rightHeight) + 1;

    return height;
}
```

Call Stack

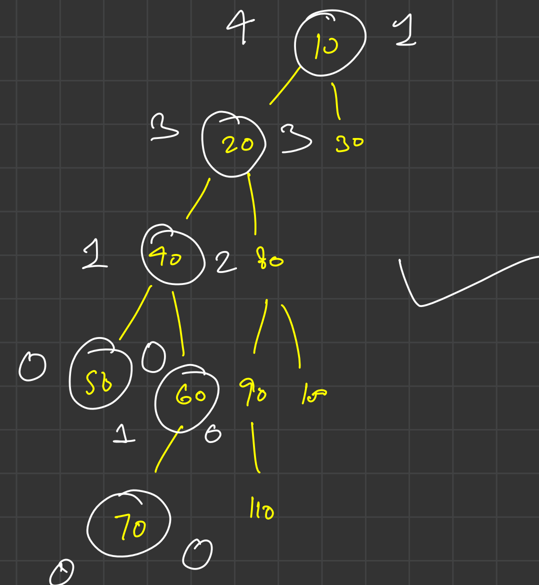
diameter of tree { max^m dist b/w any two leaf Nodes }





diameter = 6

```java
static int maxDiameter = 0;

public static void getDiameter(Node root) {
    if (root == null) {
        return;
    }

    int lstHeight = getHeight(root.left);
    int rstHeight = getHeight(root.right);

    int diameter = lstHeight + 1 + rstHeight;
    maxDiameter = Math.max(maxDiameter, diameter);

    getDiameter(root.left);
    getDiameter(root.right);
}

public static int diameter(Node root) {
    // Your code here
```



dia = ~~6~~ ~~7~~ ~~3~~ ~~4~~ ~~2~~ 1

maxdia = ~~6~~ 7

max {mydia, Bestdia left, Best dia right}