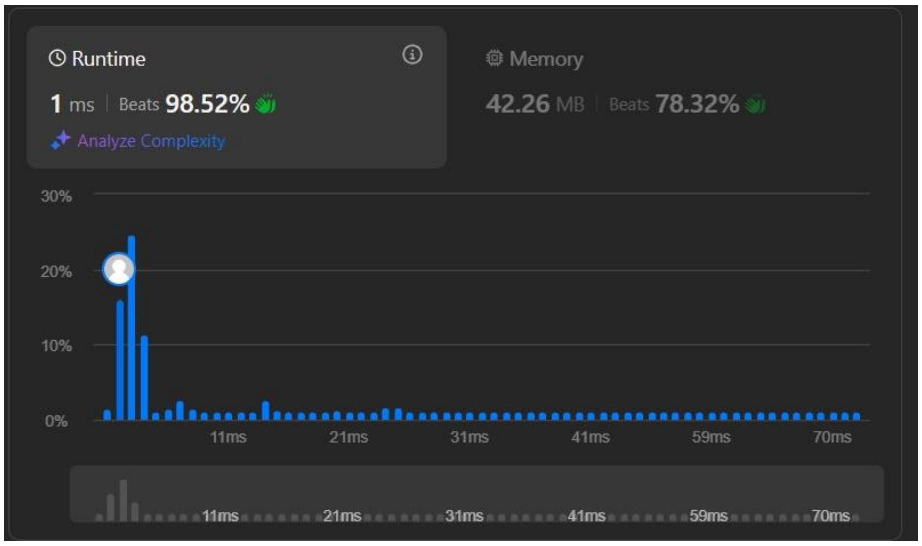
1. Binary Tree Inorder Traversal

```
class Solution {
    List<Integer> res = new ArrayList<>();
    public List<Integer> inorderTraversal(TreeNode root) {
        if (root != null) {
            inorderTraversal(root.left);
            res.add(root.val);
            inorderTraversal(root.right);
        }
        return res;
    }
}
```


Runtime

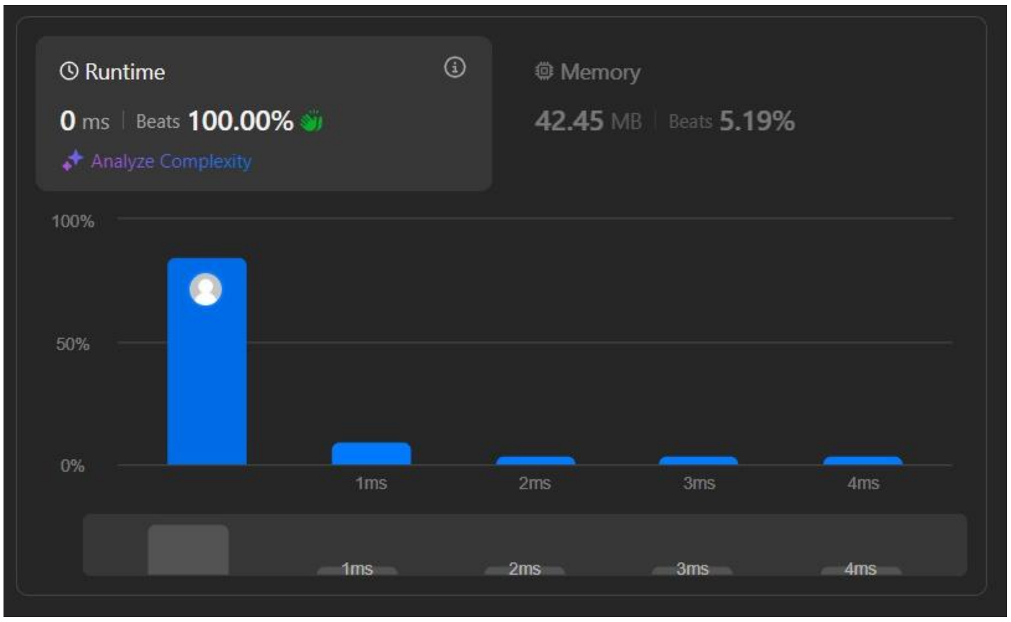
1 ms | Beats 98.52%

Memory

42.26 MB | Beats 78.32%

2. Symmetric Tree

```
class Solution {
    public boolean isSymmetric(TreeNode root) {
        return root == null || isSymmetric(root.left, root.right);
    }
    boolean isSymmetric(TreeNode l, TreeNode r) {
        return l == null || r == null ? l == r : l.val == r.val &&
isSymmetric(l.left, r.right) && isSymmetric(l.right, r.left);
    }
}
```


Runtime

0 ms | Beats 100.00%

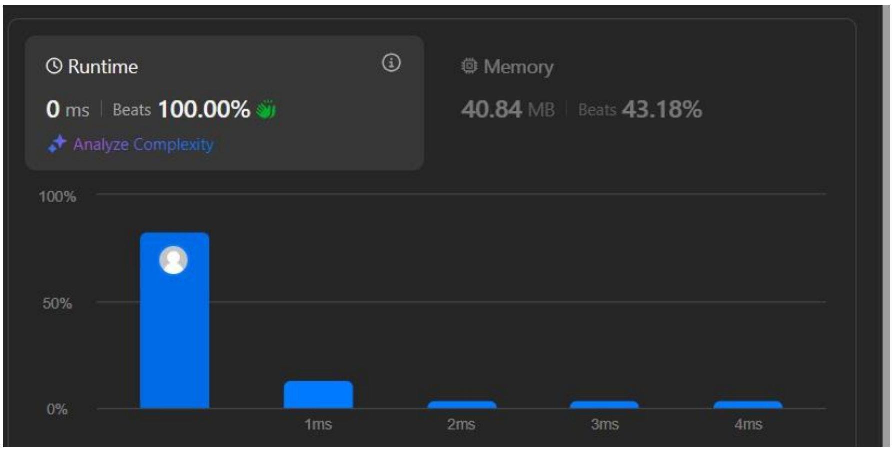
Analyze Complexity

Memory

42.45 MB | Beats 5.19%

3. Validate Binary Search Tree

```
class Solution {
    public boolean isValidBST(TreeNode root) {
        return isValidBST(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }
    boolean isValidBST(TreeNode n, long min, long max) {
        return n == null || (n.val > min && n.val < max && isValidBST(n.left,
min, n.val) && isValidBST(n.right, n.val, max));
    }
}
```


Runtime

0 ms | Beats 100.00%

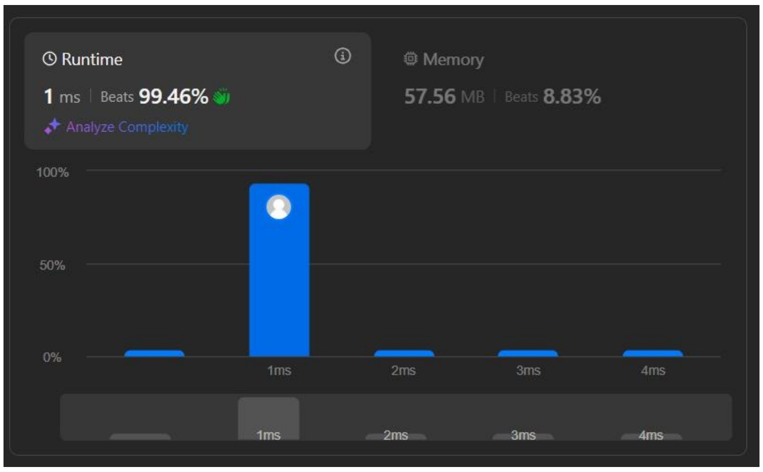
Analyze Complexity

Memory

40.84 MB | Beats 43.18%

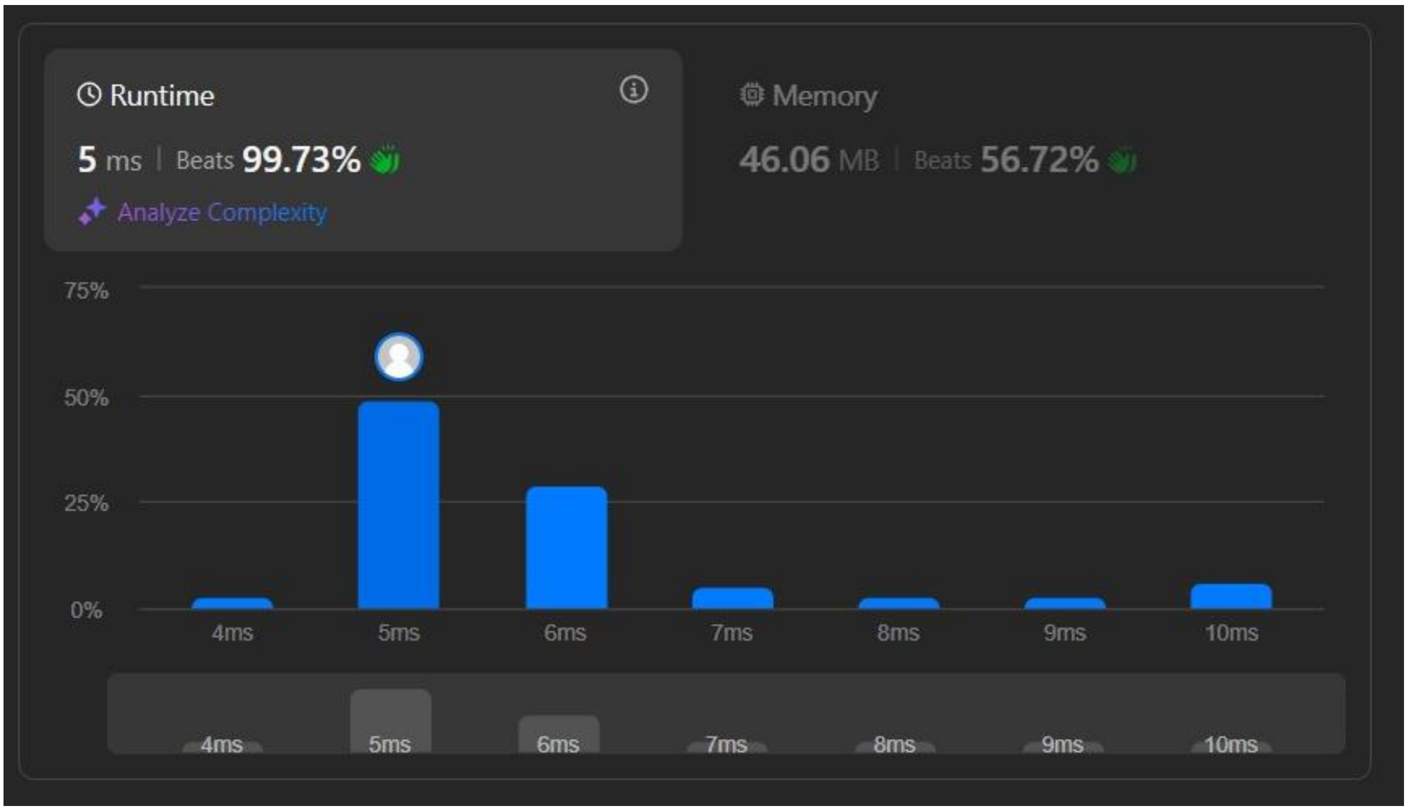
4. Kth Smallest Element in a BST

```
class Solution {
    int k, res;
    public int kthSmallest(TreeNode root, int k) {
        this.k = k;
        inorder(root);
        return res;
    }
    void inorder(TreeNode n) {
        if (n == null) return;
        inorder(n.left);
        if (--k == 0) res = n.val;
        inorder(n.right);
    }
}
```


Runtime
1 ms | Beats 99.46%
Memory
57.56 MB | Beats 8.83%

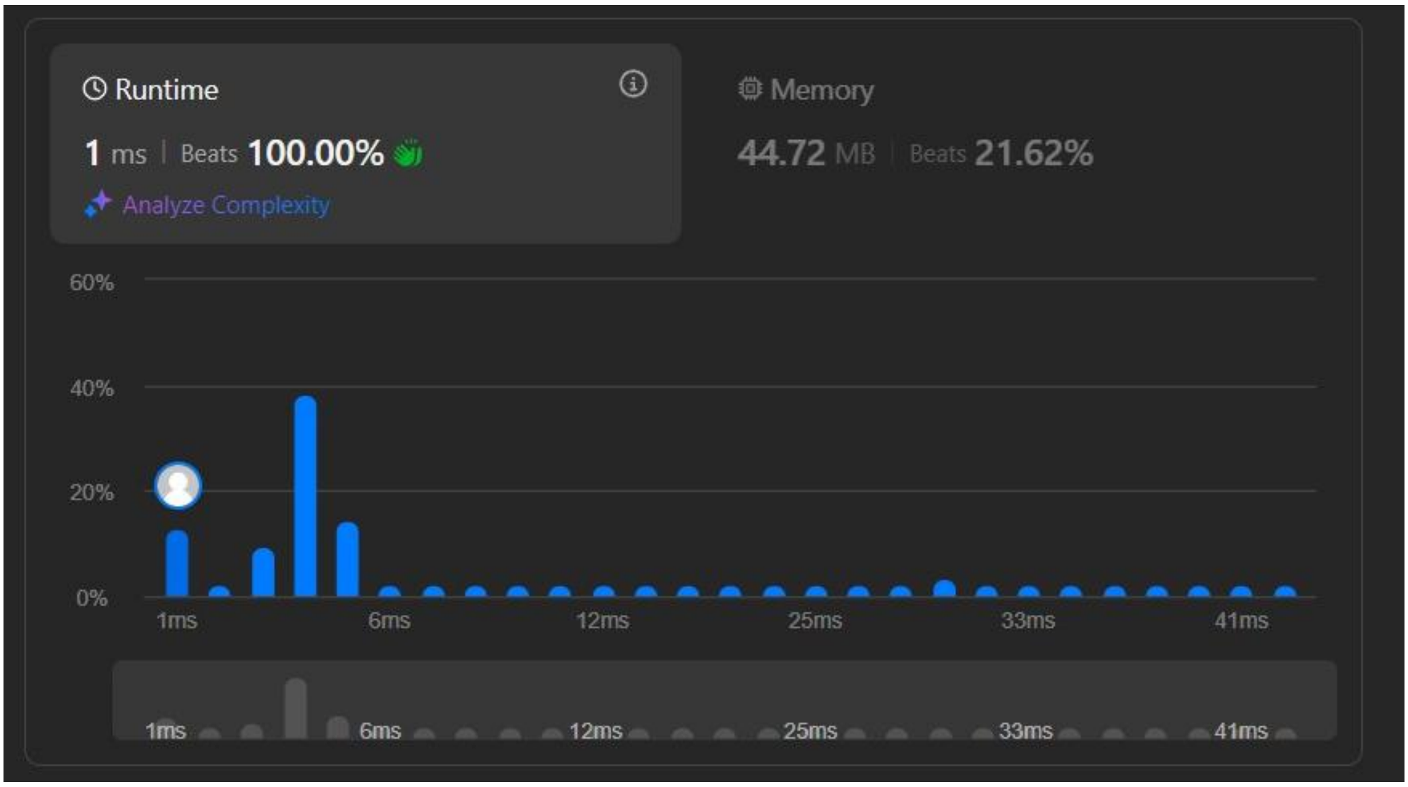
5. Binary Tree Level Order Traversal

```
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> res = new ArrayList<>();
        Queue<TreeNode> q = new LinkedList<>();
        if (root != null) q.add(root);
        while (!q.isEmpty()) {
            List<Integer> level = new ArrayList<>();
            for (int i = q.size(); i > 0; i--) {
                TreeNode n = q.poll();
                level.add(n.val);
                if (n.left != null) q.add(n.left);
                if (n.right != null) q.add(n.right);
            }
            res.add(level);
        }
        return res;
    }
}
```



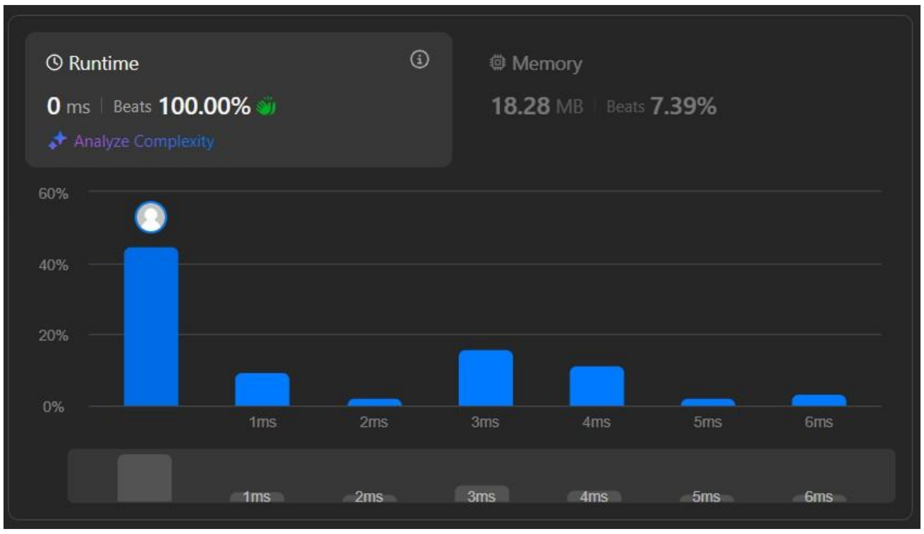
6. Binary Tree Level Order Traversal II

```
class Solution {
    public List<List<Integer>> levelOrderBottom(TreeNode root) {
        List<List<Integer>> res = new ArrayList<>();
        Queue<TreeNode> q = new LinkedList<>();
        if (root != null) q.add(root);
        while (!q.isEmpty()) {
            List<Integer> level = new ArrayList<>();
            for (int i = q.size(); i > 0; i--) {
                TreeNode n = q.poll();
                level.add(n.val);
                if (n.left != null) q.add(n.left);
                if (n.right != null) q.add(n.right);
            }
            res.add(0, level);
        }
        return res;
    }
}
```


Runtime
1 ms | Beats 100.00%
Analyze Complexity
Memory
44.72 MB | Beats 21.62%

7. Maximum Depth of Binary Tree

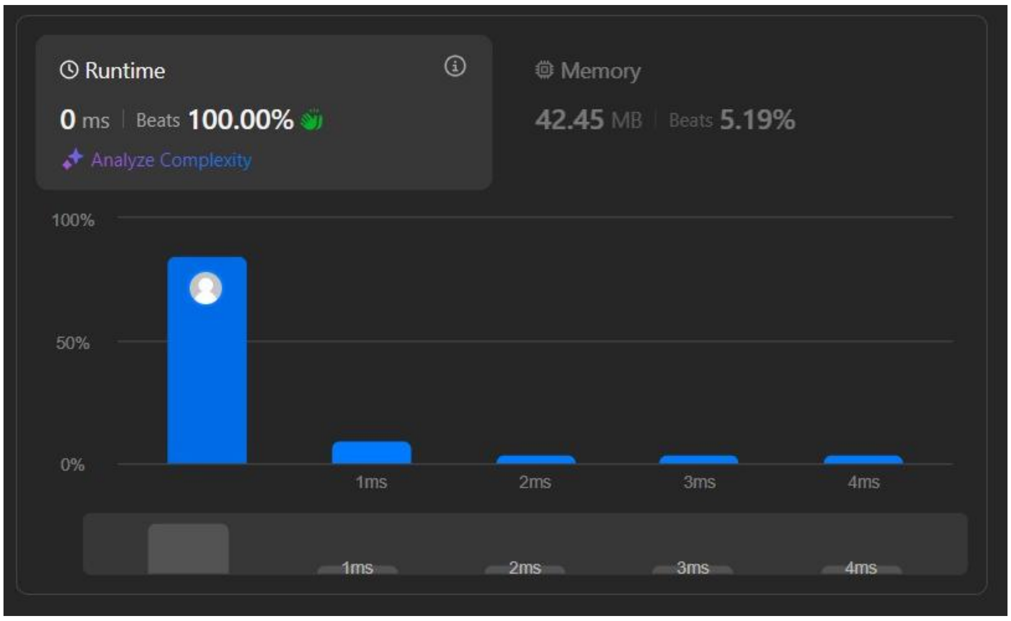
```
class Solution {
    public int maxDepth(TreeNode root) {
        return root == null ? 0 : 1 + Math.max(maxDepth(root.left),
maxDepth(root.right));
    }
}
```


Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
18.28 MB | Beats 7.39%

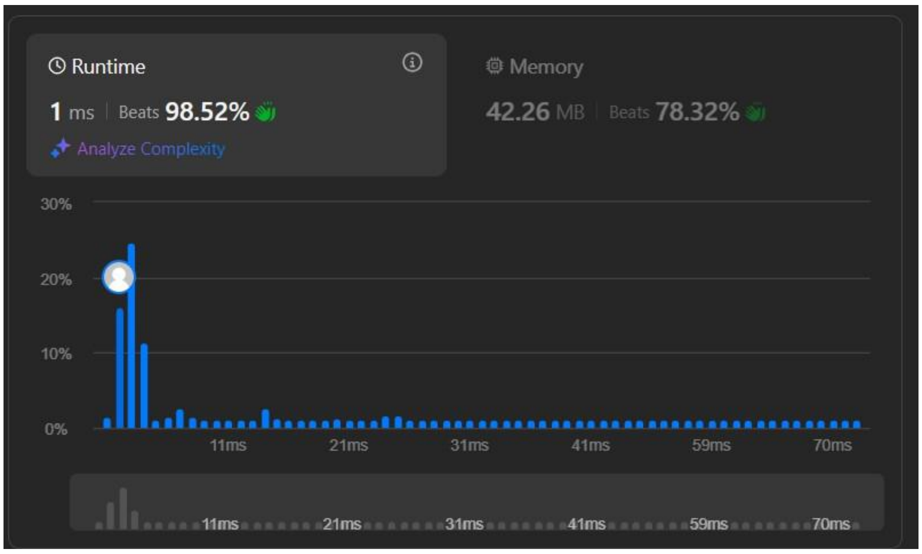
8. Binary Tree Zigzag Level Order Traversal

```
    class Solution {
   public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
       List<List<Integer>> res = new ArrayList<>();
       Queue<TreeNode> q = new LinkedList<>();
       if (root != null) q.add(root);
       boolean rev = false;
       while (!q.isEmpty()) {
           LinkedList<Integer> level = new LinkedList<>();
           for (int i = q.size(); i > 0; i--) {
               TreeNode n = q.poll();
               if (rev) level.addFirst(n.val);
               else level.add(n.val);
               if (n.left != null) q.add(n.left);
               if (n.right != null) q.add(n.right);
           }
           res.add(level);
           rev = !rev;
       }
       return res;
   }
}
```

9. Binary Tree Right Side View



```
    class Solution {
   public List<Integer> rightSideView(TreeNode root) {
       List<Integer> res = new ArrayList<>();
       Queue<TreeNode> q = new LinkedList<>();
       if (root != null) q.add(root);
       while (!q.isEmpty()) {
           for (int i = q.size(); i > 0; i--) {
                TreeNode n = q.poll();
                if (i == 1) res.add(n.val);
                if (n.left != null) q.add(n.left);
                if (n.right != null) q.add(n.right);
            }
        }
        return res;
    }
}
```


Runtime

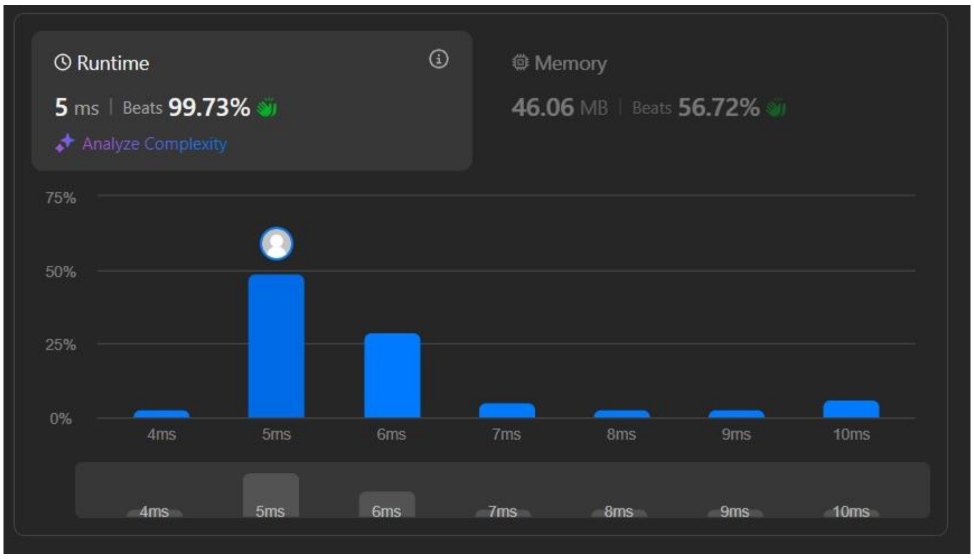
1 ms | Beats 98.52%

Memory

42.26 MB | Beats 78.32%

10. Find Bottom Left Tree Value

```
     class Solution {
    public int findBottomLeftValue(TreeNode root) {
        Queue<TreeNode> q = new LinkedList<>();
        q.add(root);
        int res = root.val;
        while (!q.isEmpty()) {
            TreeNode n = q.poll();
            res = n.val;
            if (n.right != null) q.add(n.right);
            if (n.left != null) q.add(n.left);
        }
        return res;
    }
}
```


Runtime

5 ms | Beats 99.73%

Analyze Complexity

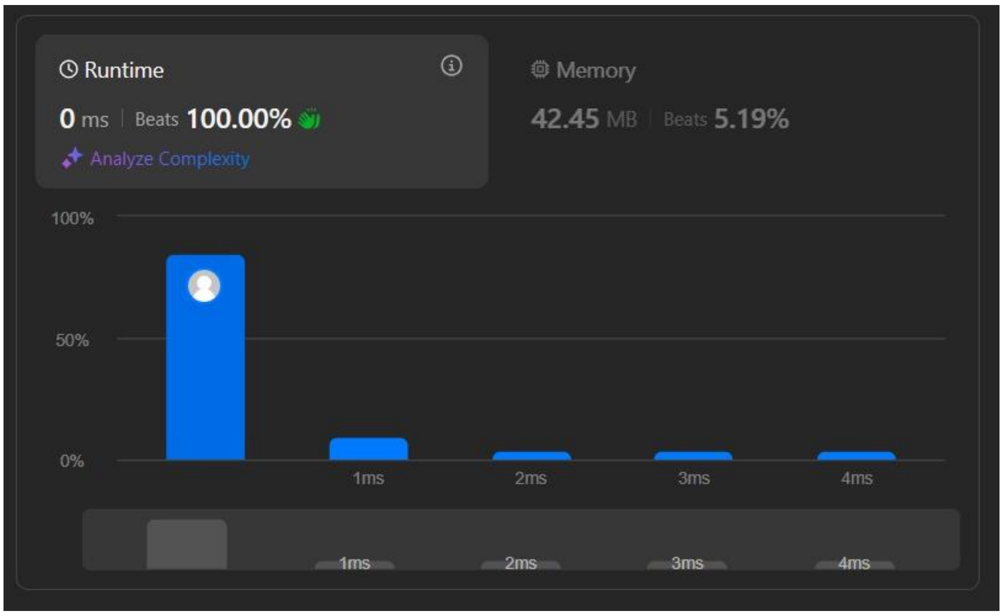
Memory

46.06 MB | Beats 56.72%

11. Binary Tree Maximum Path Sum

```
class Solution {
    int max = Integer.MIN_VALUE;
    public int maxPathSum(TreeNode root) {
        dfs(root);
        return max;
    }
    int dfs(TreeNode n) {
        if (n == null) return 0;
        int l = Math.max(dfs(n.left), 0), r = Math.max(dfs(n.right), 0);
        max = Math.max(max, l + r + n.val);
        return n.val + Math.max(l, r);
    }

}
```


Runtime

0 ms | Beats 100.00%

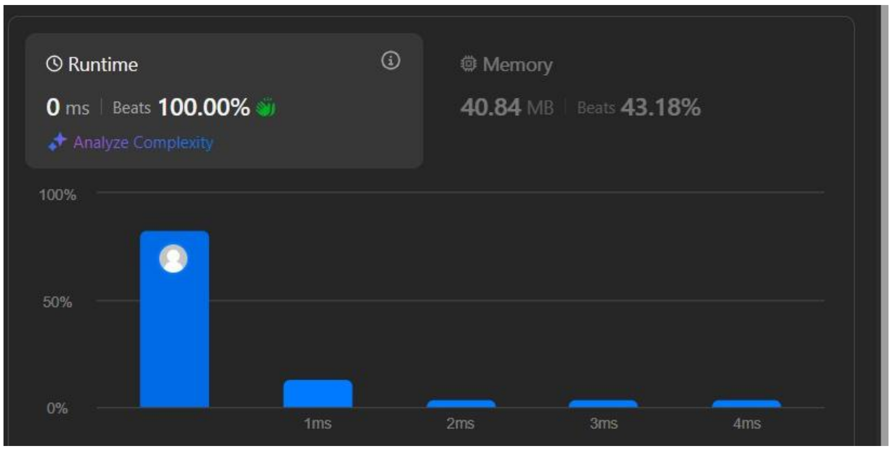
Analyze Complexity

Memory

42.45 MB | Beats 5.19%

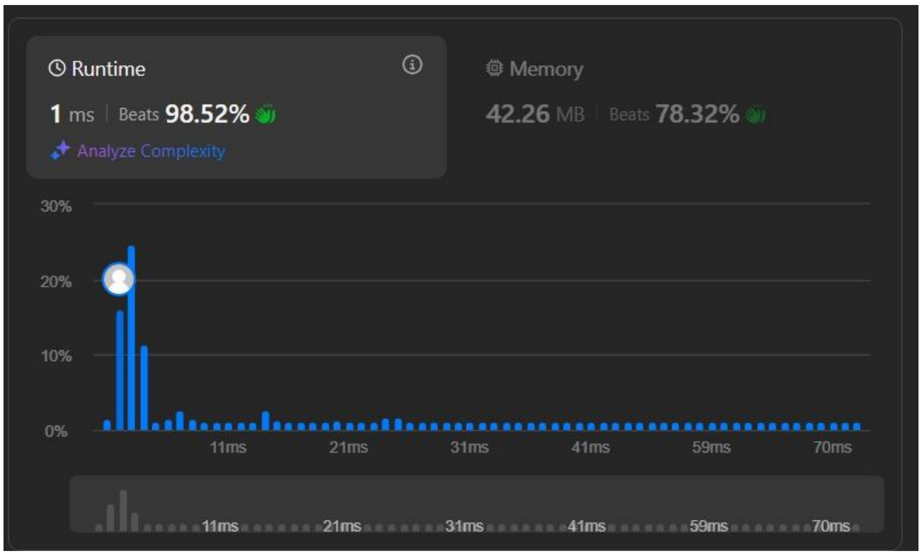
12. Vertical Order Traversal of a Binary Tree

```
class Solution {
    public List<List<Integer>> verticalTraversal(TreeNode root) {
        TreeMap<Integer, TreeMap<Integer, PriorityQueue<Integer>>> map = new
TreeMap<>();
        dfs(root, 0, 0, map);
        List<List<Integer>> res = new ArrayList<>();
        for (TreeMap<Integer, PriorityQueue<Integer>> ys : map.values()) {
            List<Integer> col = new ArrayList<>();
            for (PriorityQueue<Integer> nodes : ys.values())
                while (!nodes.isEmpty()) col.add(nodes.poll());
            res.add(col);
        }
        return res;
    }
    void dfs(TreeNode n, int x, int y, TreeMap<Integer, TreeMap<Integer,
PriorityQueue<Integer>>> map) {
        if (n == null) return;
        map.computeIfAbsent(x, k -> new TreeMap<>()).computeIfAbsent(y, k ->
new PriorityQueue<>()).add(n.val);
        dfs(n.left, x - 1, y + 1, map);
        dfs(n.right, x + 1, y + 1, map);
    }
}
```


Runtime
0 ms | Beats 100.00%
Memory
40.84 MB | Beats 43.18%

13. Construct Binary Tree from Inorder and Postorder Traversal

```
     class Solution {
    int i;
    public TreeNode buildTree(int[] in, int[] post) {
        i = post.length - 1;
        return build(in, post, 0, in.length - 1);
    }
    TreeNode build(int[] in, int[] post, int l, int r) {
        if (l > r) return null;
        TreeNode root = new TreeNode(post[i--]);
        int mid = l;
        while (in[mid] != root.val) mid++;
        root.right = build(in, post, mid + 1, r);
        root.left = build(in, post, l, mid - 1);
        return root;
    }
}
```



Runtime

1 ms | Beats 98.52%

Analyze Complexity

Memory

42.26 MB | Beats 78.32%