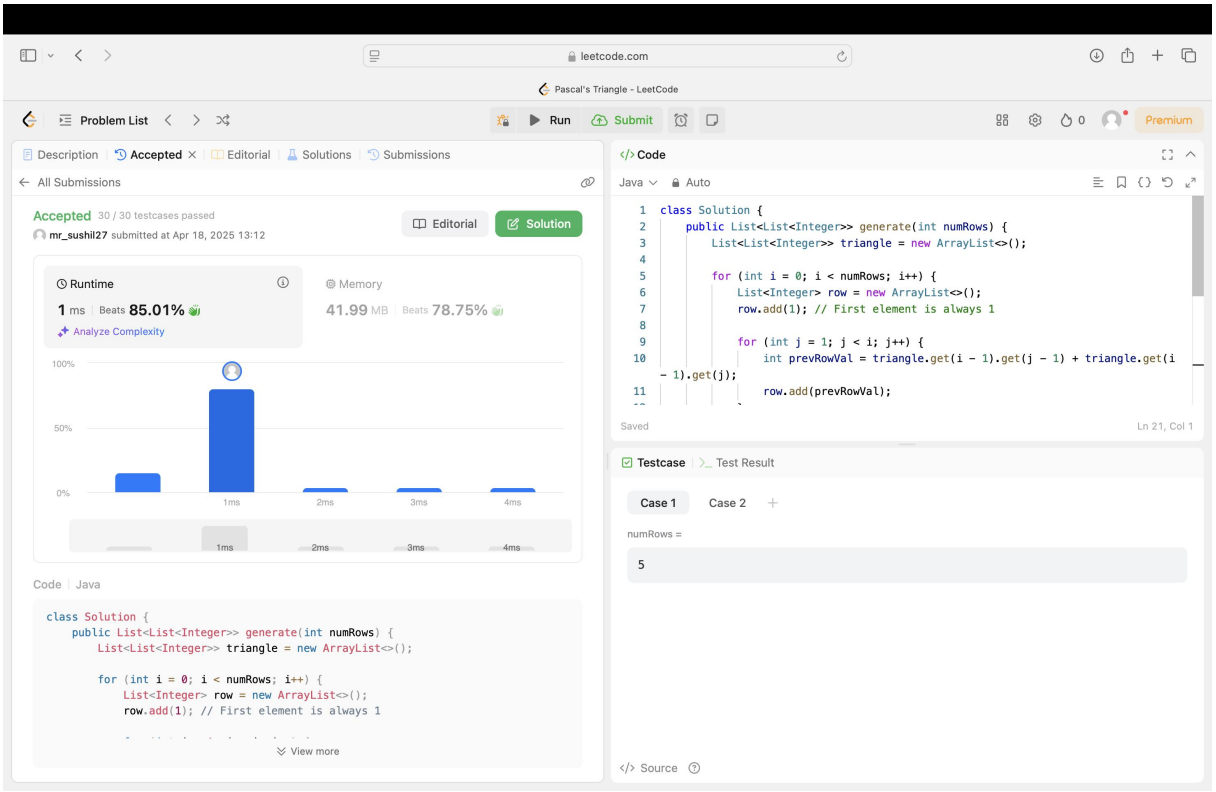
AP Assignment 10

Name: Sushil Kumar **UID: 22BCS16771** **Section: 22BCS_IOT614-B**

Ques 1https://leetcode.com/problems/pascals-triangle/description/

Code:

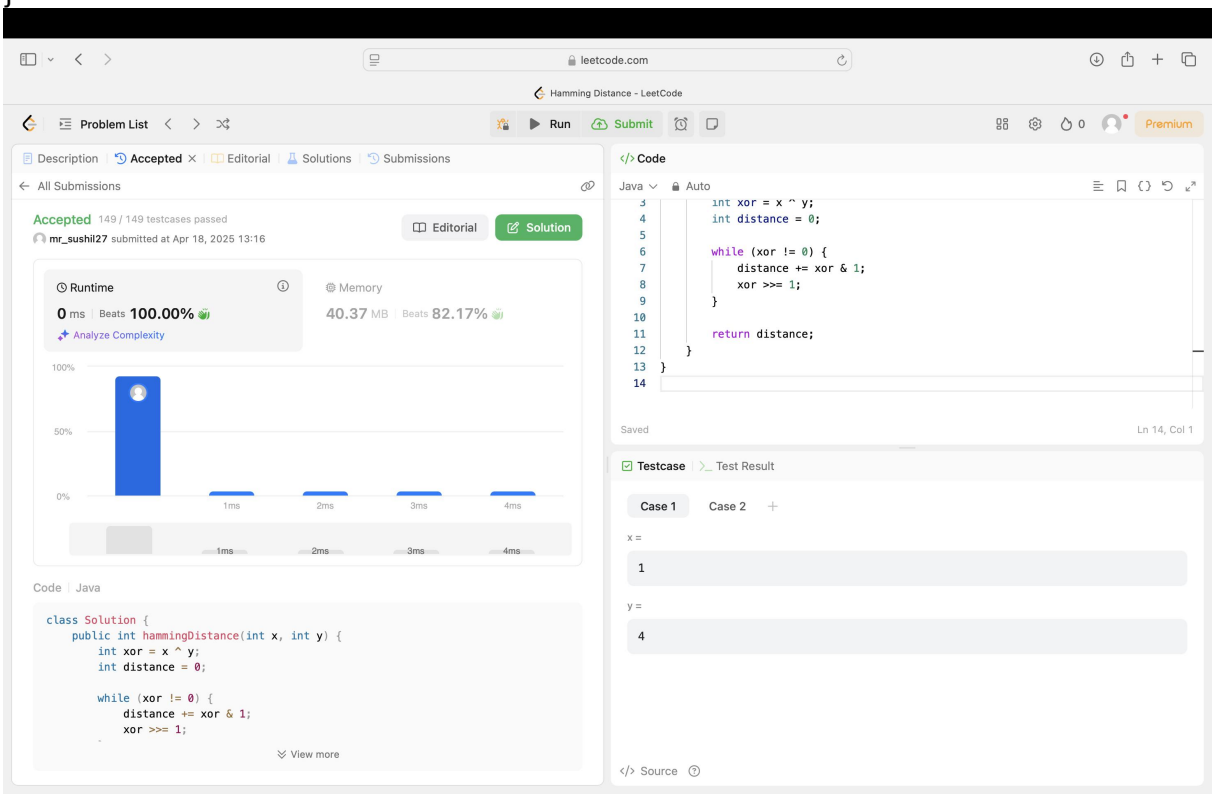
```
class Solution {
    public List<List<Integer>> generate(int numRows) {
        List<List<Integer>> triangle = new ArrayList<>();

        for (int i = 0; i < numRows; i++) {
            List<Integer> row = new ArrayList<>();
            row.add(1); // First element is always 1

            for (int j = 1; j < i; j++) {
                int prevRowVal = triangle.get(i - 1).get(j - 1) + triangle.get(i - 1).get(j);
                row.add(prevRowVal);
            }

            if (i > 0) row.add(1); // Last element is 1 (for rows > 1)
            triangle.add(row);
        }

        return triangle;
    }
}
```



Q2. https://leetcode.com/problems/hamming-distance/

CODE:

```
class Solution {
    public int hammingDistance(int x, int y) {
        int xor = x ^ y;
        int distance = 0;

        while (xor != 0) {
            distance += xor & 1;
            xor >>= 1;
        }

        return distance;
    }
}
```

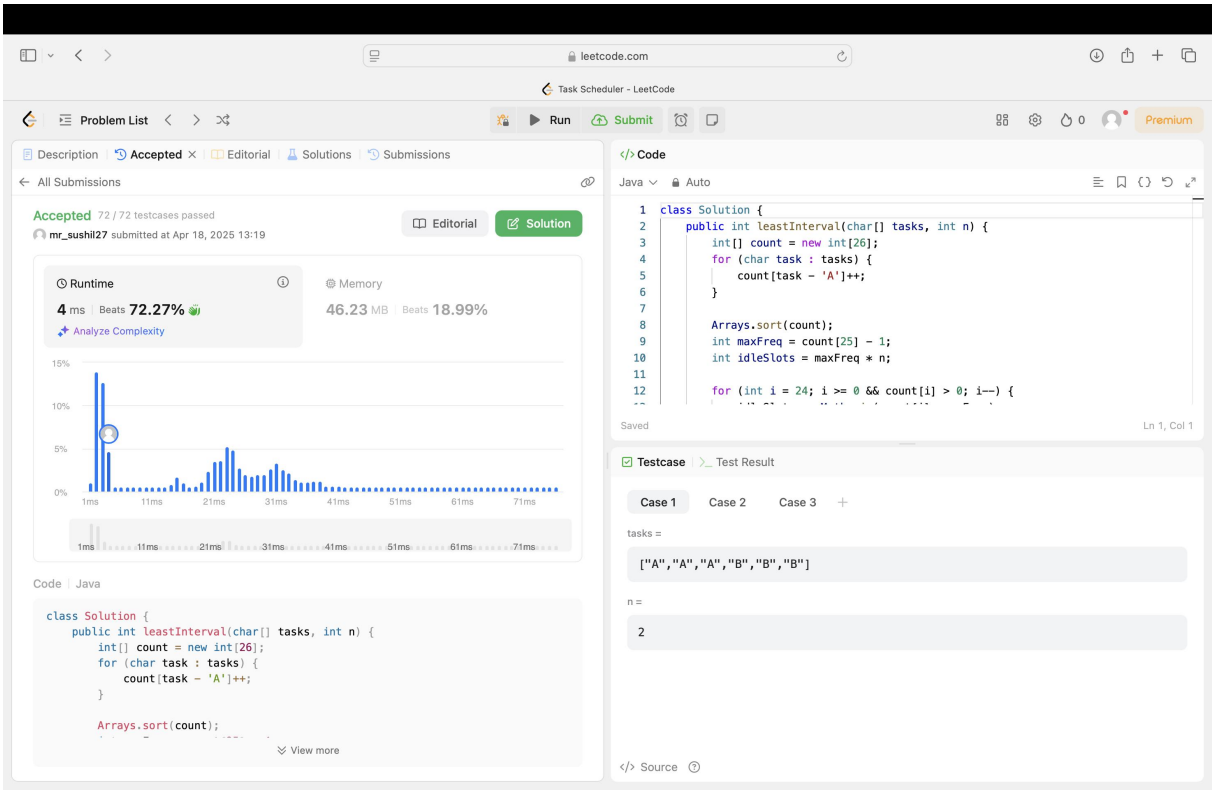


Q3. https://leetcode.com/problems/task-scheduler/description/

Code:

```
class Solution {
    public int leastInterval(char[] tasks, int n) {
        int[] count = new int[26];
        for (char task : tasks) {
            count[task - 'A']++;
        }

        Arrays.sort(count);
        int maxFreq = count[25] - 1;
        int idleSlots = maxFreq * n;

        for (int i = 24; i >= 0 && count[i] > 0; i--) {
```

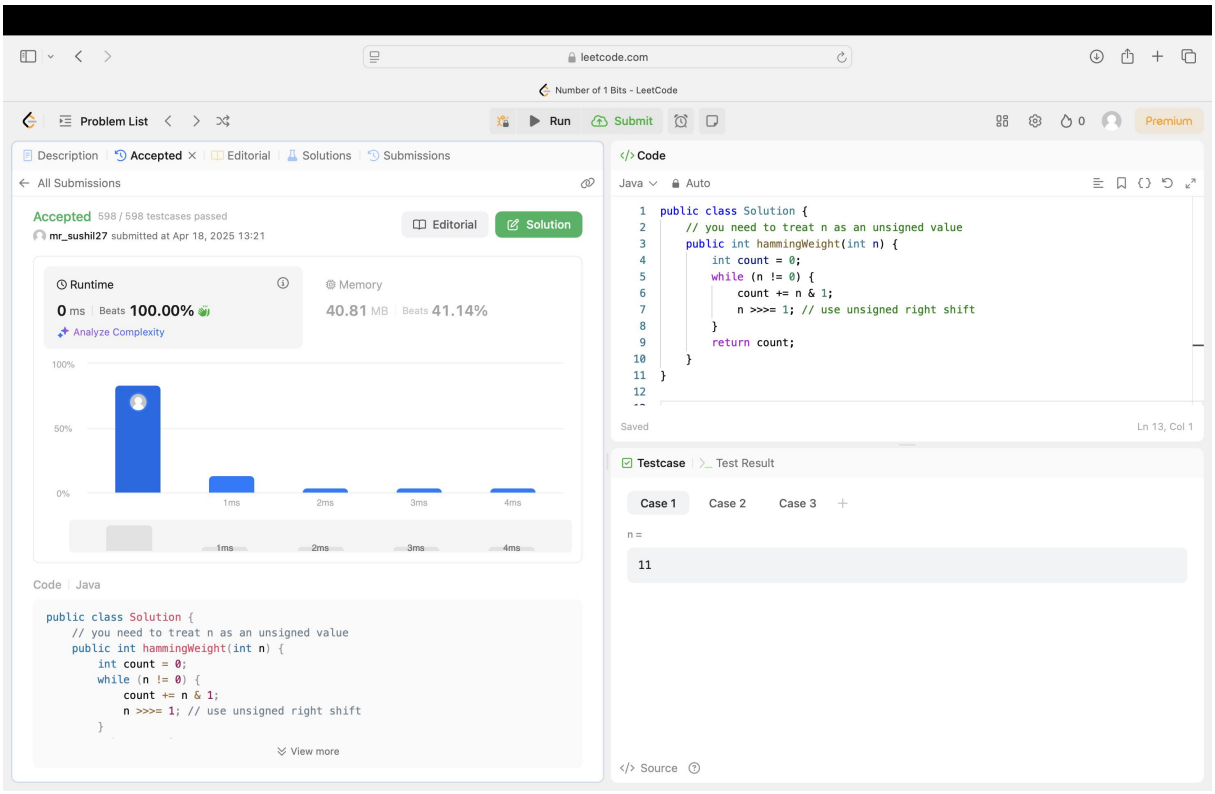
```
            idleSlots -= Math.min(count[i], maxFreq);
        }

        return idleSlots > 0 ? idleSlots + tasks.length : tasks.length;
    }
}
```



Q4.https://leetcode.com/problems/number-of-1-bits/description/

Code:

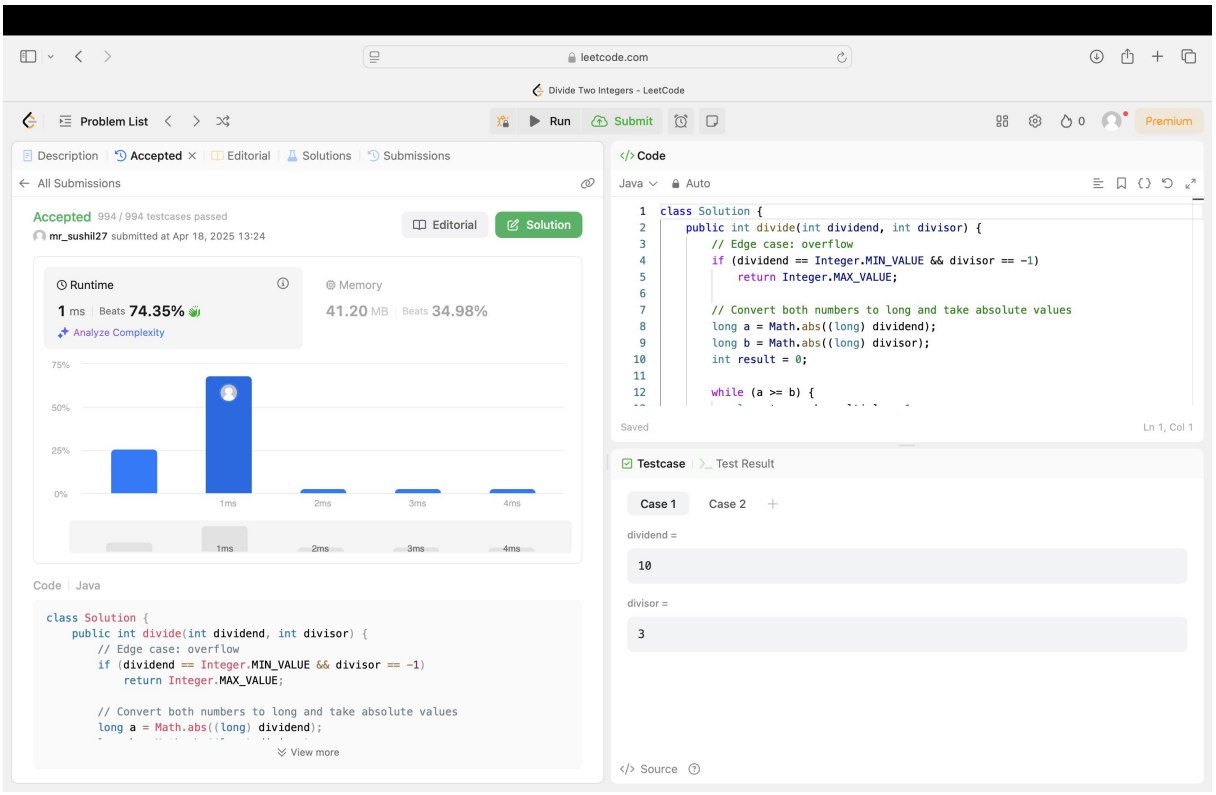
```
public class Solution {
    // you need to treat n as an unsigned value
    public int hammingWeight(int n) {
        int count = 0;
        while (n != 0) {
            count += n & 1;
            n >>>= 1; // use unsigned right shift
        }
        return count;
    }
}
```



Q5. https://leetcode.com/problems/divide-two-integers/description/

CODE:

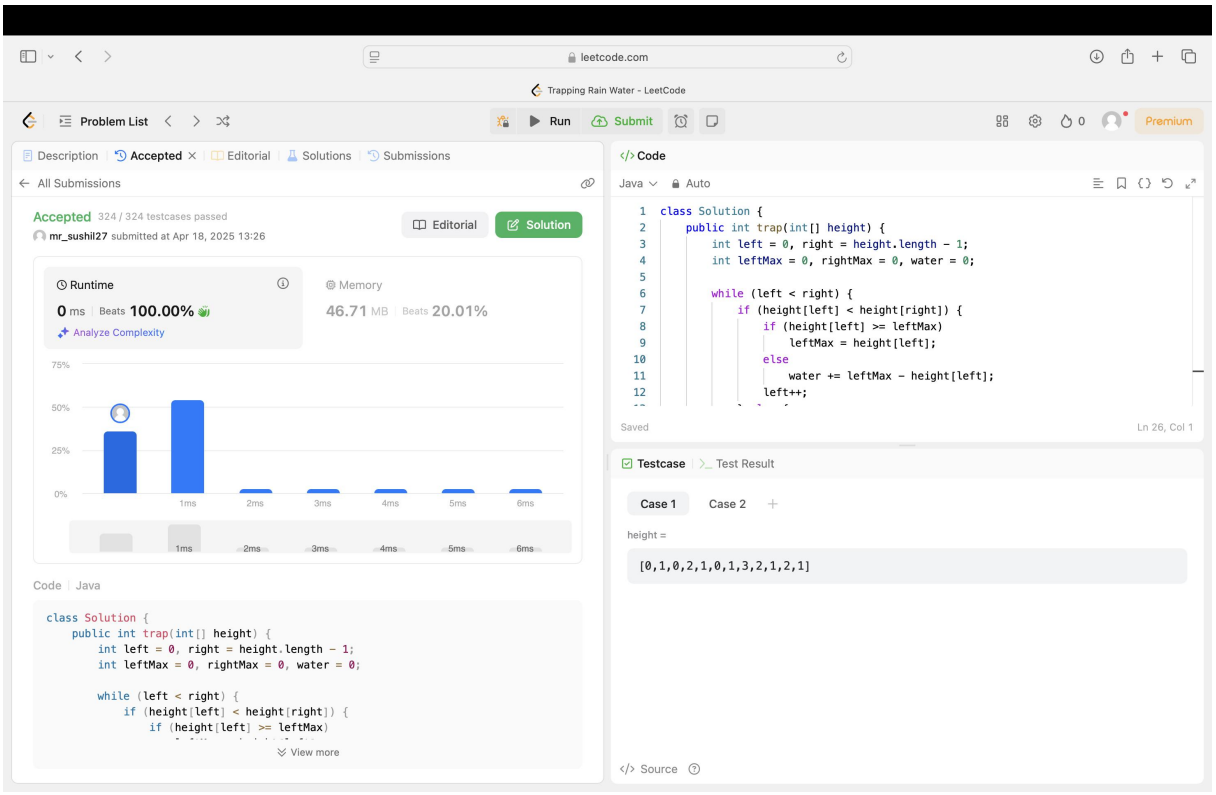
```
class Solution {
    public int divide(int dividend, int divisor) {
        // Edge case: overflow
        if (dividend == Integer.MIN_VALUE && divisor == -1)
            return Integer.MAX_VALUE;

        // Convert both numbers to long and take absolute values
        long a = Math.abs((long) dividend);
        long b = Math.abs((long) divisor);
        int result = 0;

        while (a >= b) {
            long temp = b, multiple = 1;
            while (a >= (temp << 1)) {
                temp <<= 1;
                multiple <<= 1;
            }
            a -= temp;
            result += multiple;
        }

        // Apply sign
        return (dividend > 0) == (divisor > 0) ? result : -result;
    }
}
```



Q6. https://leetcode.com/problems/trapping-rain-water/description/

CODE:-

```
class Solution {
    public int trap(int[] height) {
        int left = 0, right = height.length - 1;
        int leftMax = 0, rightMax = 0, water = 0;

        while (left < right) {
            if (height[left] < height[right]) {
                if (height[left] >= leftMax)
                    leftMax = height[left];
                else
                    water += leftMax - height[left];
                left++;
            } else {
                if (height[right] >= rightMax)
                    rightMax = height[right];
                else
                    water += rightMax - height[right];
                right--;
            }
        }

        return water;
    }
}
```

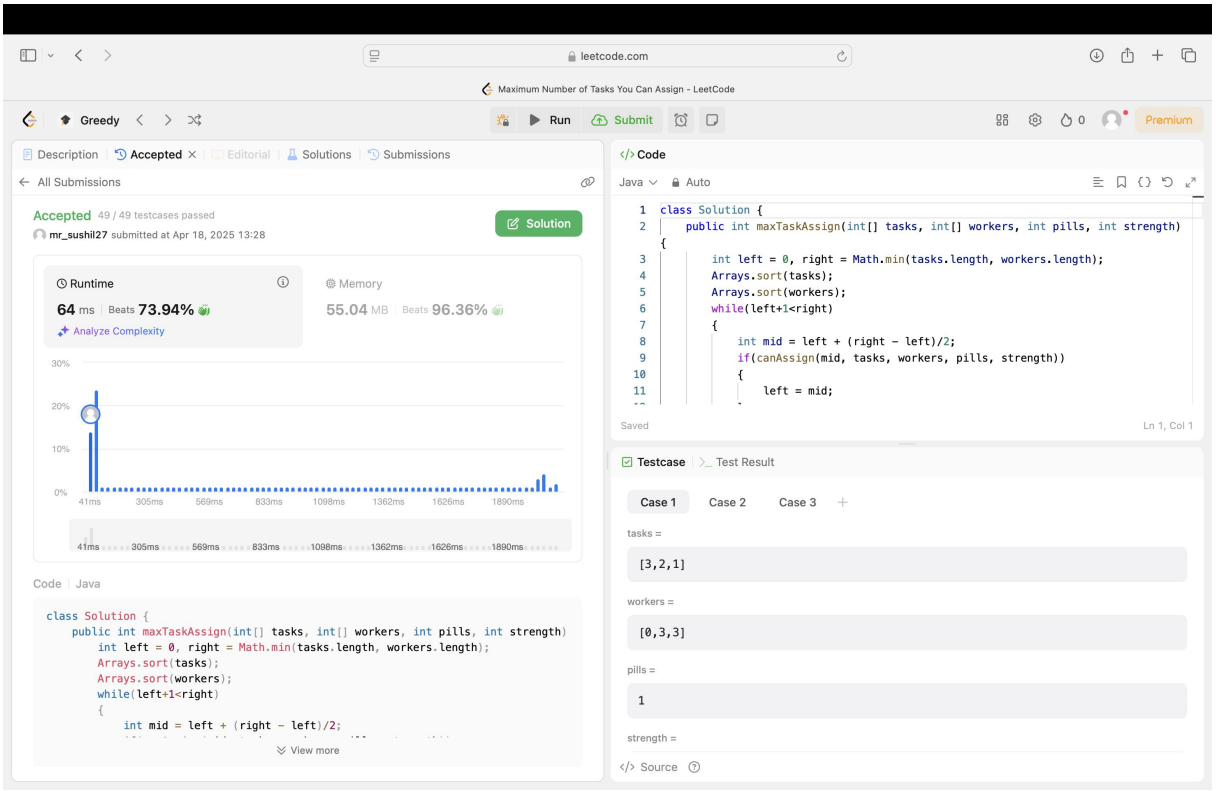


Q7. https://leetcode.com/problems/maximum-number-of-tasks-you-can-assign/description/?envType=problem-list-v2&envId=greedy

CODE:-

```
class Solution {
public int maxTaskAssign(int[] tasks, int[] workers, int pills, int strength) {
int left = 0, right = Math.min(tasks.length, workers.length);
Arrays.sort(tasks);
Arrays.sort(workers);
while(left+1<right)
{
int mid = left + (right - left)/2;
if(canAssign(mid, tasks, workers, pills, strength))
{
left = mid;
}
else
{
right = mid;
}
}
if(canAssign(right, tasks, workers, pills, strength))
{
return right;
}
else return left;
}
public boolean canAssign(int count, int[] tasks, int[] workers, int pills, int strength){
Deque<Integer> dq = new ArrayDeque<>();
int ind = workers.length - 1;
```

```
for (int i = count - 1; i >= 0; i--) {
while(ind>=workers.length-count && workers[ind]+strength>=tasks[i])
{
dq.offerLast(workers[ind]);
ind--;
}
if(dq.isEmpty())return false;
if(dq.peekFirst()>=tasks[i])
{
dq.pollFirst();
}
else
{
dq.pollLast();
pills--;
if(pills<0)return false;
}
}
return true;
}
}
```

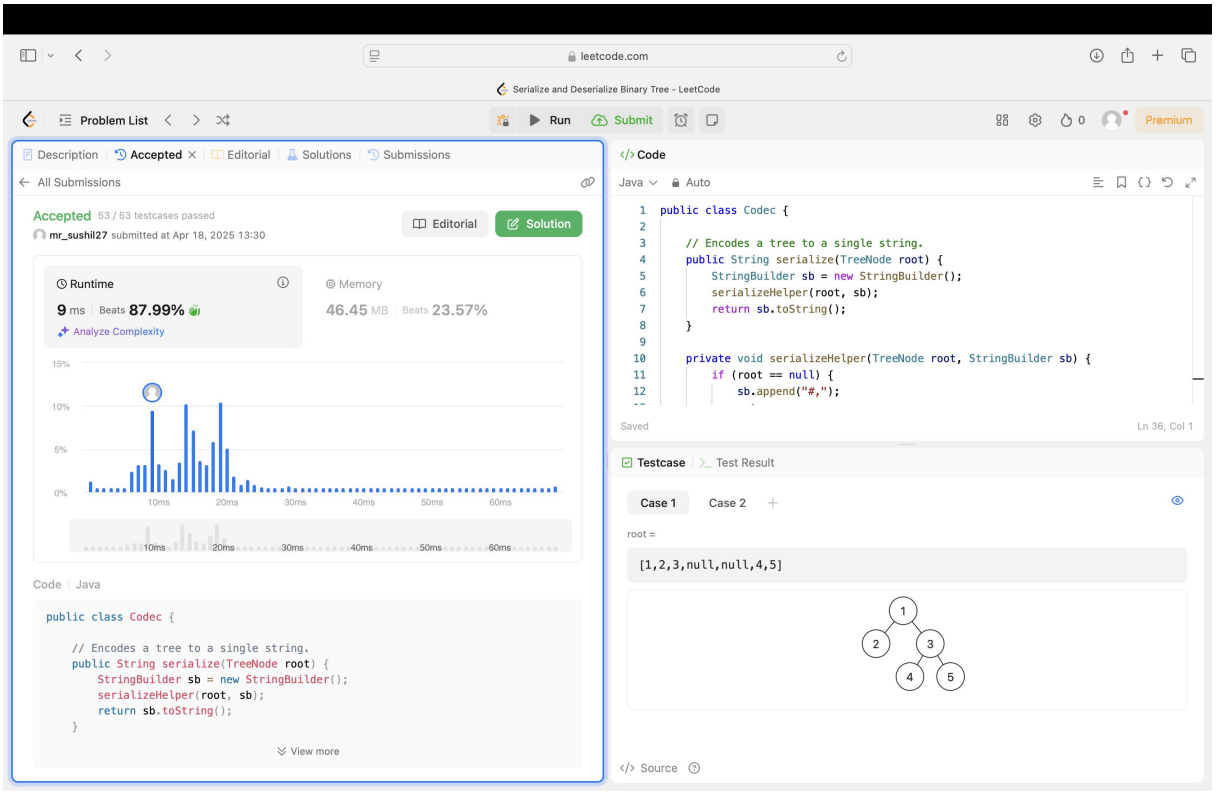


Q8. https://leetcode.com/problems/serialize-and-deserialize-binary-tree/description/

CODE:-

```
public class Codec {

    // Encodes a tree to a single string.
    public String serialize(TreeNode root) {
        StringBuilder sb = new StringBuilder();
```

```
        serializeHelper(root, sb);
        return sb.toString();
    }

    private void serializeHelper(TreeNode root, StringBuilder sb) {
        if (root == null) {
            sb.append("#,");
            return;
        }
        sb.append(root.val).append(",");
        serializeHelper(root.left, sb);
        serializeHelper(root.right, sb);
    }

    // Decodes your encoded data to tree.
    public TreeNode deserialize(String data) {
        Queue<String> nodes = new LinkedList<>(Arrays.asList(data.split(",")));
        return deserializeHelper(nodes);
    }

    private TreeNode deserializeHelper(Queue<String> nodes) {
        String val = nodes.poll();
        if (val.equals("#")) return null;

        TreeNode root = new TreeNode(Integer.parseInt(val));
        root.left = deserializeHelper(nodes);
        root.right = deserializeHelper(nodes);
        return root;
    }
}
```



```
```

Q9. https://leetcode.com/problems/lru-cache/description/

CODE:-

```
class LRUCache extends LinkedHashMap<Integer, Integer> {
    private int capacity;

    public LRUCache(int capacity) {
        super(capacity, 0.75f, true); // accessOrder = true
        this.capacity = capacity;
    }

    public int get(int key) {
        return super.getOrDefault(key, -1);
    }

    public void put(int key, int value) {
        super.put(key, value);
    }

    @Override
    protected boolean removeEldestEntry(Map.Entry<Integer, Integer> eldest) {
        return size() > capacity;
    }
}
```