Assignment 10

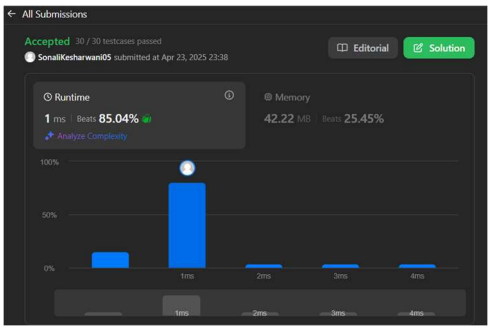
Q1. Pascal's Triangle

Code:

```java
class Solution {
    public List<List<Integer>> generate(int numRows) {
        List<List<Integer>> result = new ArrayList<>();
        if (numRows == 0) {
            return result;
        }

        if (numRows == 1) {
            List<Integer> firstRow = new ArrayList<>();
            firstRow.add(1);
            result.add(firstRow);
            return result;
        }

        result = generate(numRows - 1);
        List<Integer> prevRow = result.get(numRows - 2);
        List<Integer> currentRow = new ArrayList<>();
        currentRow.add(1);

        for (int i = 1; i < numRows - 1; i++) {
            currentRow.add(prevRow.get(i - 1) + prevRow.get(i));
        }

        currentRow.add(1);
        result.add(currentRow);
```

```java
        return result;
    }
}
```

Output:

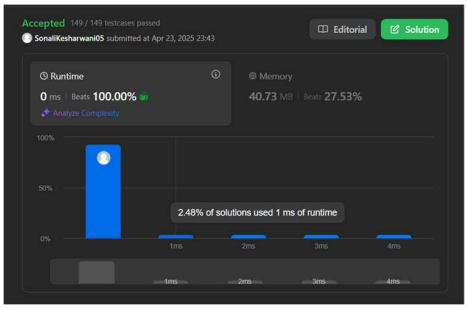


Q2. Hamming Distance

Code:

```java
public class Solution {
    public int hammingDistance(int x, int y) {
        return Integer.bitCount(x ^ y);
    }
}
```

Output:

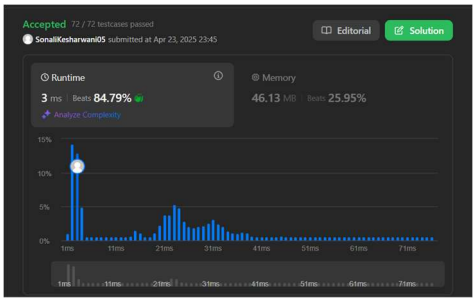


Q3. Task Scheduler

Code:

```java
class Solution {
    public int leastInterval(char[] tasks, int n) {
        int[] freq = new int[26];
        for (char task : tasks) {
            freq[task - 'A']++;
        }
        Arrays.sort(freq);
        int chunk = freq[25] - 1;
        int idle = chunk * n;

        for (int i = 24; i >= 0; i--) {
            idle -= Math.min(chunk, freq[i]);
        }

        return idle < 0 ? tasks.length : tasks.length + idle;
    }
}
```

Output:



Q4. Number of 1 Bits

Code:

```java
class Solution {
    public int hammingWeight(int n) {
    String x = Integer.toBinaryString(n);
    int count =0;
    for(int i=0;i<x.length();i++){

    if(x.charAt(i)=='1'){
        count++;

            }
    }
    return count;
    }
}
```

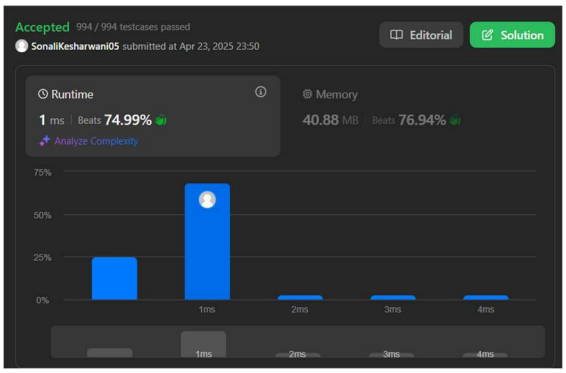
Output:

Q5. Divide Two Integers

Code:

```java
class Solution {
    public int divide(int dividend, int divisor) {
        if (dividend == Integer.MIN_VALUE && divisor == -1) {
            return Integer.MAX_VALUE; // Overflow
        }
        double ans=(dividend/divisor);
        int ans1=(int) ans;
        return ans1;
    }
}
```

Output:

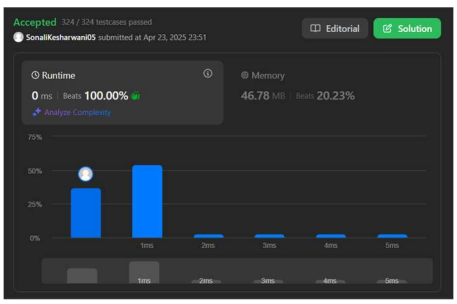


Q6. Trapping Rain Water

Code:

```java
class Solution {
    public int trap(int[] height) {
        int i=0,left_max=height[0],sum=0;
        int j=height.length-1,right_max=height[j];
        while (i<j)
        {
            if(left_max <= right_max)
            {
                sum+=(left_max-height[i]);
                i++;
                left_max=Math.max(left_max,height[i]);
            }
            else
            {
                sum+=(right_max-height[j]);
                j--;
                right_max=Math.max(right_max,height[j]);
            }
        }
        return sum;

    }
}
```

Output:



Q7. Max Number of Tasks You Can Assign

Code:

```java
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;
    }
```

```java
    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;
    }
```

Output:

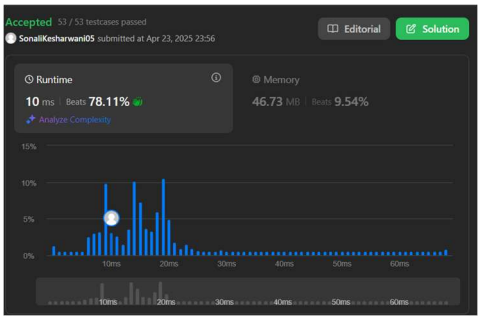
Q8. Serialize and Deserialize Binary Tree

Code:

```java
public class Codec {
    private static final String spliter = ",";
    private static final String NN = "X";
    public String serialize(TreeNode root) {
        StringBuilder sb = new StringBuilder();
        buildString(root, sb);
        return sb.toString();
    }
    private void buildString(TreeNode node, StringBuilder sb) {
        if (node == null) {
            sb.append(NN).append(spliter);
        } else {
            sb.append(node.val).append(spliter);
            buildString(node.left, sb);
            buildString(node.right,sb);
        }
    }
    public TreeNode deserialize(String data) {
        Deque<String> nodes = new LinkedList<>();
        nodes.addAll(Arrays.asList(data.split(spliter)));
        return buildTree(nodes);
    }
```

```java
    private TreeNode buildTree(Deque<String> nodes) {
        String val = nodes.remove();
        if (val.equals(NN)) return null;
        else {
            TreeNode node = new TreeNode(Integer.valueOf(val));
            node.left = buildTree(nodes);
            node.right = buildTree(nodes);
            return node;
        }
    }
}
```

Output:



Q9. LRU Cache

Code:

```java
class LRUCache {
    class Node {
        int key;
        int val;
        Node prev;
        Node next;

        Node(int key, int val) {
            this.key = key;
            this.val = val;
        }
    }

    Node head = new Node(-1, -1);
    Node tail = new Node(-1, -1);
    int cap;
    HashMap<Integer, Node> m = new HashMap<>();

    public LRUCache(int capacity) {
        cap = capacity;
        head.next = tail;
        tail.prev = head;
    }
```

```
    private void addNode(Node newnode) {
        Node temp = head.next;

        newnode.next = temp;
        newnode.prev = head;

        head.next = newnode;
        temp.prev = newnode;
    }

    private void deleteNode(Node delnode) {
        Node prevv = delnode.prev;
        Node nextt = delnode.next;

        prevv.next = nextt;
        nextt.prev = prevv;
    }
```

```
    public int get(int key) {
        if (m.containsKey(key)) {
            Node resNode = m.get(key);
            int ans = resNode.val;

            m.remove(key);
            deleteNode(resNode);
            addNode(resNode);

            m.put(key, head.next);
            return ans;
        }
        return -1;
    }

    public void put(int key, int value) {
        if (m.containsKey(key)) {
            Node curr = m.get(key);
            m.remove(key);
            deleteNode(curr);
        }

        if (m.size() == cap) {
            m.remove(tail.prev.key);
            deleteNode(tail.prev);
        }
```

```
        if (m.size() == cap) {
            m.remove(tail.prev.key);
            deleteNode(tail.prev);
        }

        addNode(new Node(key, value));
        m.put(key, head.next);
    }
}
```

Output: