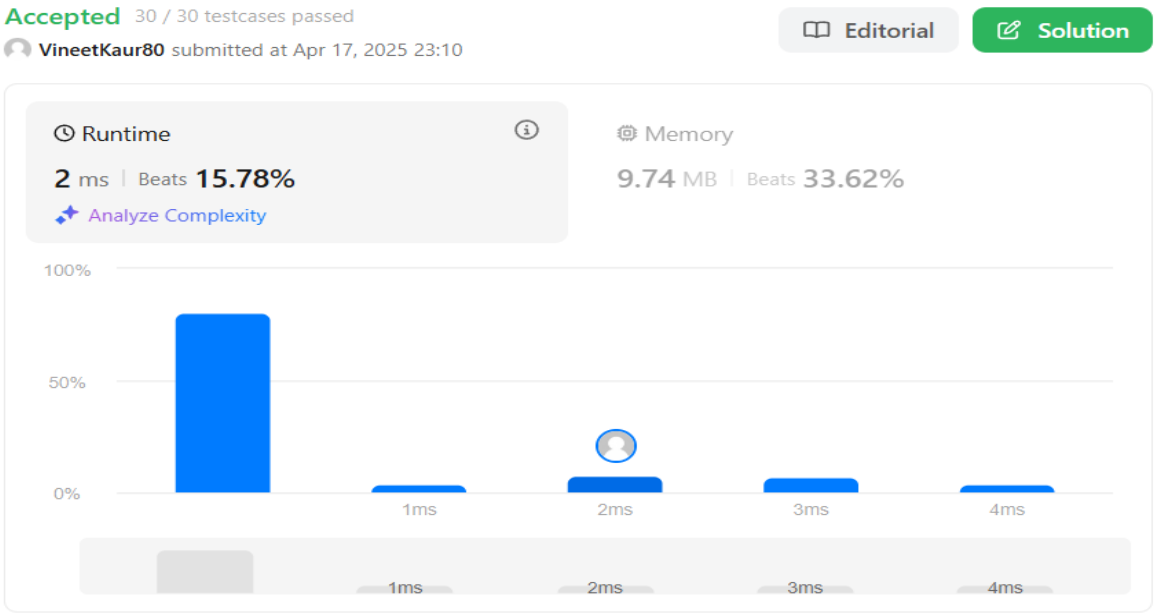
10th experiment

Pascal's Triangle

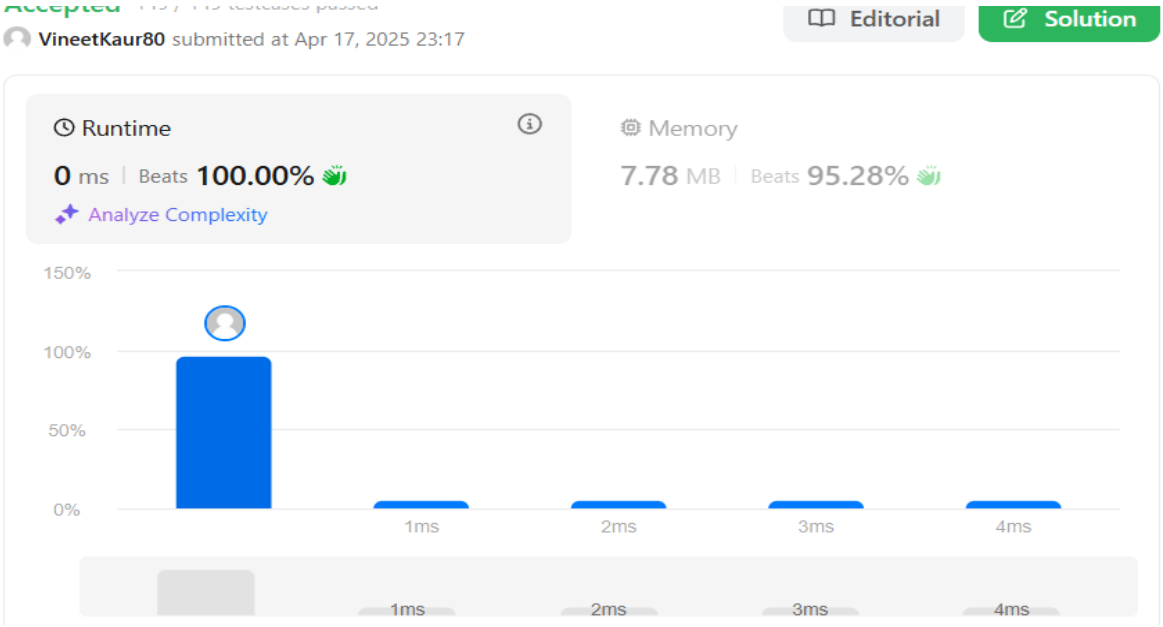
```
class Solution {
 public:
  vector<vector<int>> generate(int numRows) {
    vector<vector<int>> ans;
  for (int i = 0; i < numRows; ++i)
      ans.push_back(vector<int>(i + 1, 1));
  for (int i = 2; i < numRows; ++i)
      for (int j = 1; j < ans[i].size() - 1; ++j)
        ans[i][j] = ans[i - 1][j - 1] + ans[i - 1][j];
 return ans;
  }
};
```



Hamming Distance

```
class Solution {
public:
    int hammingDistance(int x, int y) {
        int ans = x ^ y;
        int count = 0;
```

```
        while(ans){
            count += (ans & 1);
            ans >>= 1;
        }
        return count;
    }
};
```



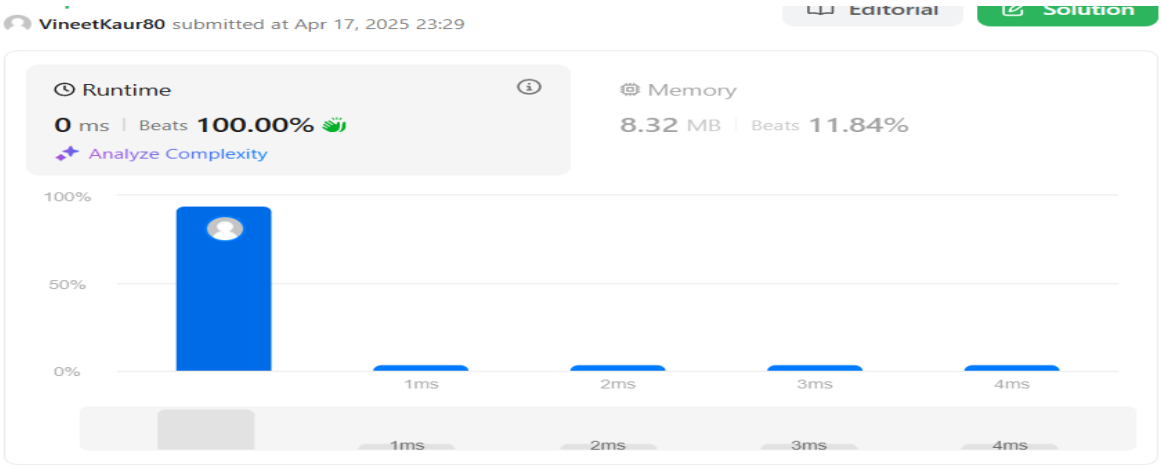
Task Scheduler

```
class Solution {
public:
    int leastInterval(vector<char>& tasks, int n) {
        unordered_map<char,int> m;
        int maxi = 0; int count = 0;
        for(int i = 0; i<tasks.size(); i++){
            m[tasks[i]]++;
            maxi = max(maxi, m[tasks[i]]);
        }
        for(auto it: m){
            if(it.second == maxi)count++;
        }
```

```
        int x = (maxi-1)*n + count-1 + maxi;
        int total = tasks.size();
        return total>x ? total: x;
    }
};
```

Number of 1 Bits

```
class Solution {
public:
    int hammingWeight(int n) {
        int count = 0;
        for(int i = 31; i >= 0; i--){
            if(((n >> i) & 1) == 1)
                count++;
        }
        return count;
    }
};
```



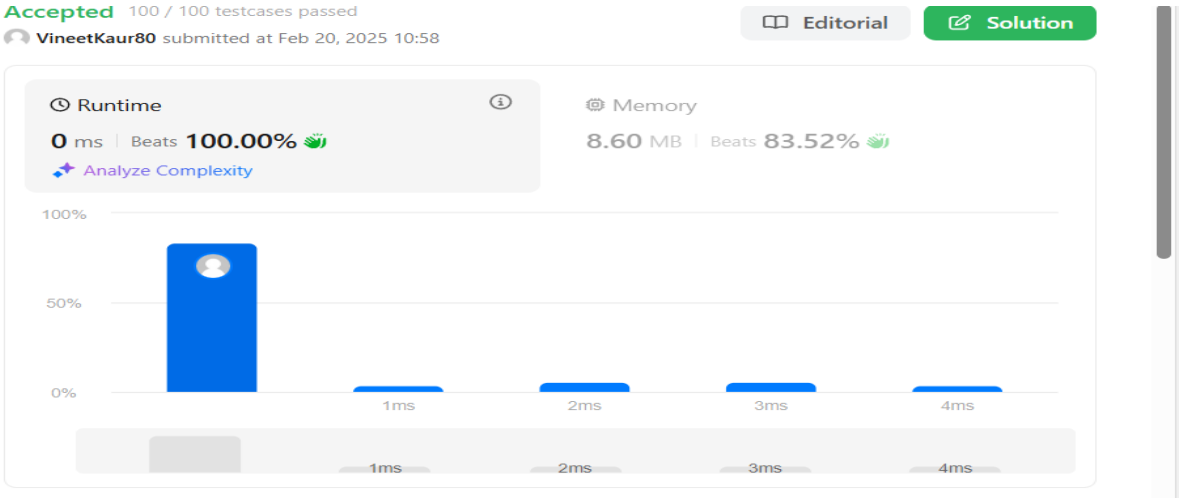
Valid Parenthesis

```
class Solution {
public:
    bool isValid(string s) {
        stack<char> s1;
        for(int i=0;i<s.size();i++){
            if(s[i]=='{' || s[i]=='[' || s[i]=='(' ){
                s1.push(s[i]);
            }else{
                if(s1.empty()){
                    return false;
                }else{
                    char ch=s1.top();
                    s1.pop();
                    if((s[i]==']' && ch=='[') || (s[i]=='}' && ch=='{' ) || (s[i]==')' && ch=='(') ){
                        continue;
                    }else{
                        return false;
                    }
                }
            }
        }
```

```
        return s1.empty();
    }
};
```

Divide Two Integers

```
class Solution {
public:
    int divide(int dividend, int divisor) {
        // Handle overflow case (INT_MIN divided by -1)
        if (dividend == INT_MIN && divisor == -1) {
            return INT_MAX;
        }
        int result;
        result = dividend / divisor;


        return result;
    }
};
```



Trapping Rain Water

```
class Solution {
public:
    int trap(vector<int>& height) {
        int n = height.size();
        if (n == 0)
            return 0;
        vector<int> leftMaxHeight(n, 0);
        vector<int> rightMaxHeight(n, 0);
        leftMaxHeight[0] = height[0];
        rightMaxHeight[n - 1] = height[n - 1];

        for (int i = 1; i < n; i++) {
            leftMaxHeight[i] = max(leftMaxHeight[i - 1], height[i]);
        }
        for (int i = n - 2; i >= 0; i--) {
            rightMaxHeight[i] = max(rightMaxHeight[i + 1], height[i]);
        }

        int totalTrappedWater = 0;
        for (int i = 0; i < n; i++) {
            totalTrappedWater +=
```

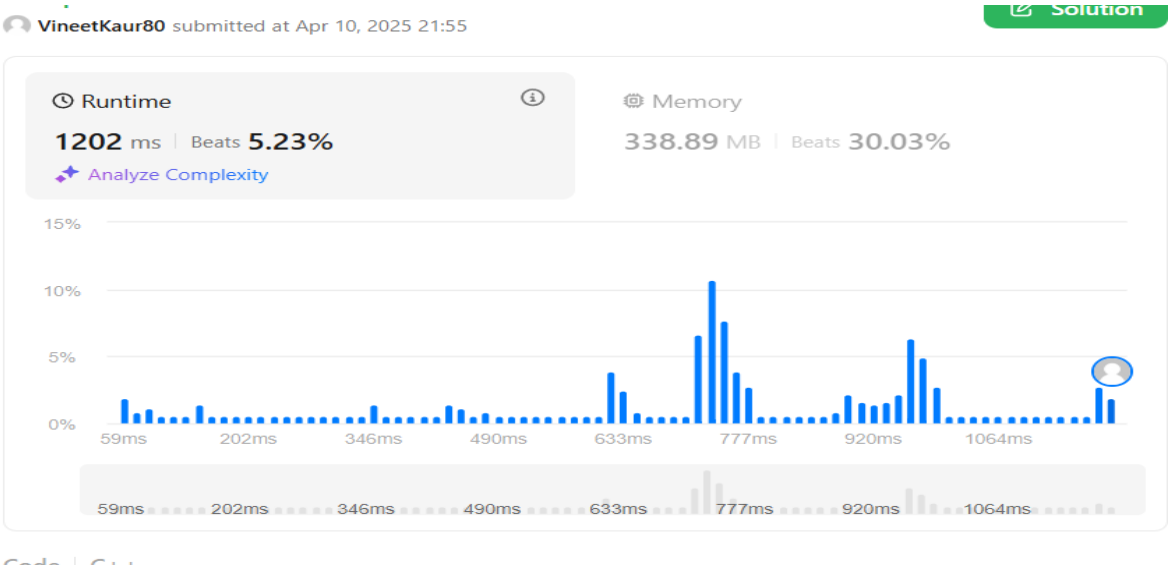
```
            min(leftMaxHeight[i], rightMaxHeight[i]) - height[i];
        }

        return totalTrappedWater;
    }
};
```

Max Number of Tasks You Can Assign

```
class Solution {
public:
    bool check(vector<int>& tasks, vector<int>& workers, int pills, int strength,int index)
    {
        multiset<int> st;
        for(auto it:workers)
        {
            st.insert(it);
        }
        for(int i=index-1;i>=0;i--)
        {
            auto it=st.lower_bound(tasks[i]);
            if(it!=st.end())
            {
                st.erase(it);
            }
```

```
        else
        {
            if(pills<=0)
            {
                return false;
            }
            else
            {
                it=st.lower_bound(tasks[i]-strength);
                if(it!=st.end())
                {
                    st.erase(it);
                    pills--;
                }
                else
                {
                    return false;
                }
            }
        }
    }
    return true;
}
int maxTaskAssign(vector<int>& tasks, vector<int>& workers, int pills, int strength) {
    sort(tasks.begin(),tasks.end());
    sort(workers.begin(),workers.end());
    int low=0;
    int high=min(workers.size(),tasks.size());
    while(low<high)
    {
        int mid=(low+high+1)/2;
        if(check(tasks,workers,pills,strength,mid)==true)
```

```
        {
            low=mid;
        }
        else
        {
            high=mid-1;
        }
    }
    return high;
  }
};
```



Serialize and Deserialize Binary Tree

```
class Codec {
public:

    void buildString(TreeNode* root, string &res)
    {
        if(root == NULL)
        {   res += "null,";
            return;
        }
```

```
    res += to_string(root->val) + ",";
    buildString(root->left, res);
    buildString(root->right, res);
}

string serialize(TreeNode* root)
{
    string res = "";
    buildString(root, res);
    return res;
}

TreeNode* buildTree(queue<string> &q)
{
    string s = q.front();
    q.pop();

    if(s == "null")
        return NULL;

    TreeNode* root = new TreeNode(stoi(s));
    root->left = buildTree(q);
    root->right = buildTree(q);
    return root;
}

TreeNode* deserialize(string data)
{
    string s = "";
    queue <string> q;
```

```
        for(char c: data)
        {
            if(c == ',')
            {
                q.push(s);
                s = "";
            }
            else
                s += c;
        }

        return buildTree(q);
    }
};
```