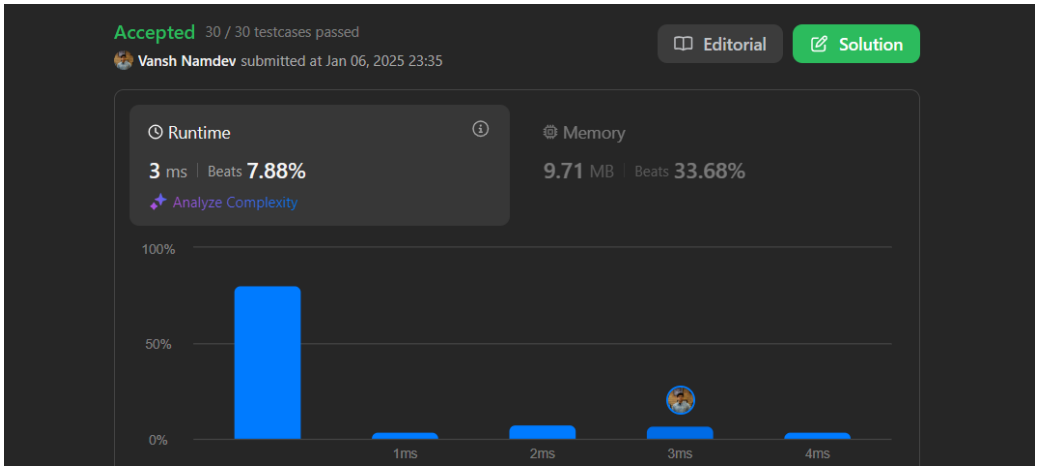
NAME- Vansh Namdev| UID- 22BCS10714 | SECTION- 601/A 1

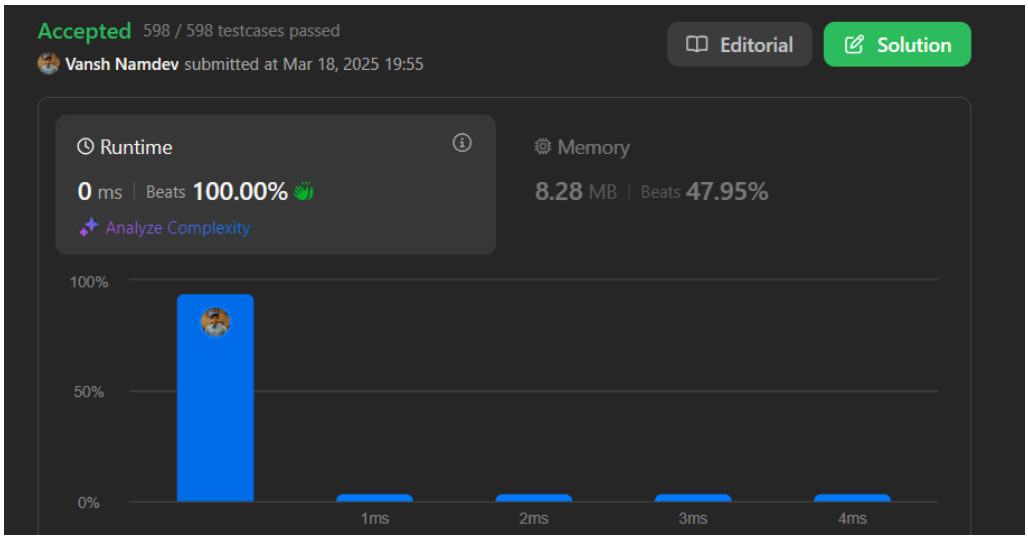
1. Pascal's Triangle

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

};
```



2. Number of 1 Bits

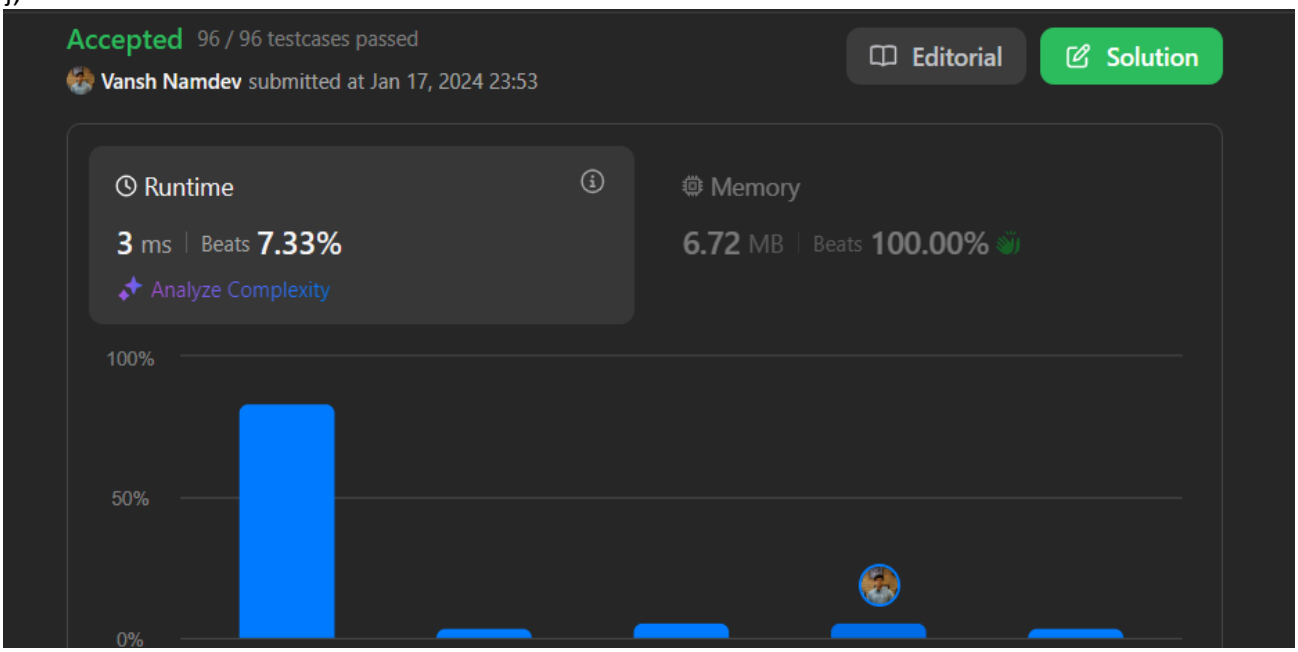
```
class Solution {
public:
    int hammingWeight(uint32_t n) {
        int res = 0;
        for (int i = 0; i < 32; i++) {
            if ((n >> i) & 1) {
                res += 1;
            }
        }
        return res;
    }
```



3. };

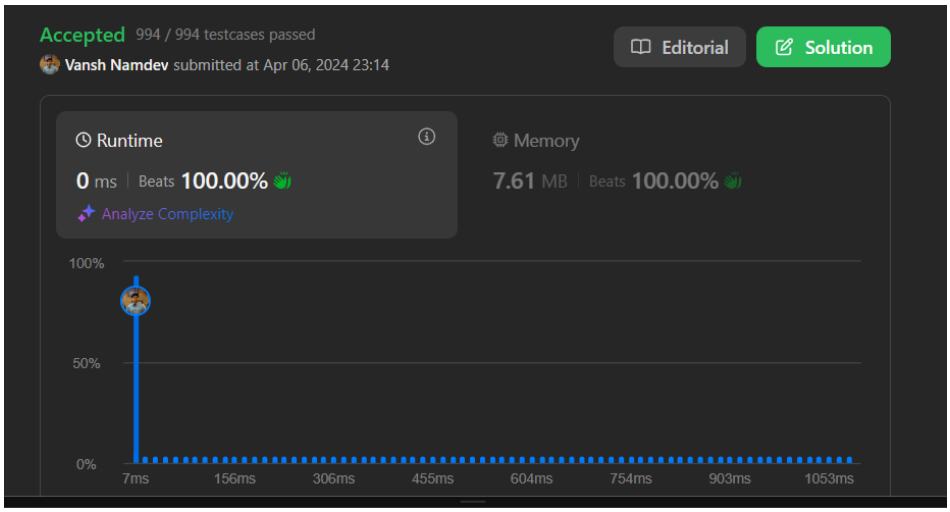
3. Valid parenthesis:

```
class Solution {
public:
    bool flag=false;
    bool isValid(string s) {
        stack<char> st;
        for(char c:s)
        {
            if(c=='('||c=='{'||c=='[')
            {
                st.push(c);
            }else{
                if(st.empty()||
                (c==')'&& st.top()!='('||
                (c=='}'&& st.top()!='{'||
                (c==']'&& st.top()!='['))
                {
                    return false;
                }
                st.pop();
            }
        }
        return st.empty();
    }
};
```



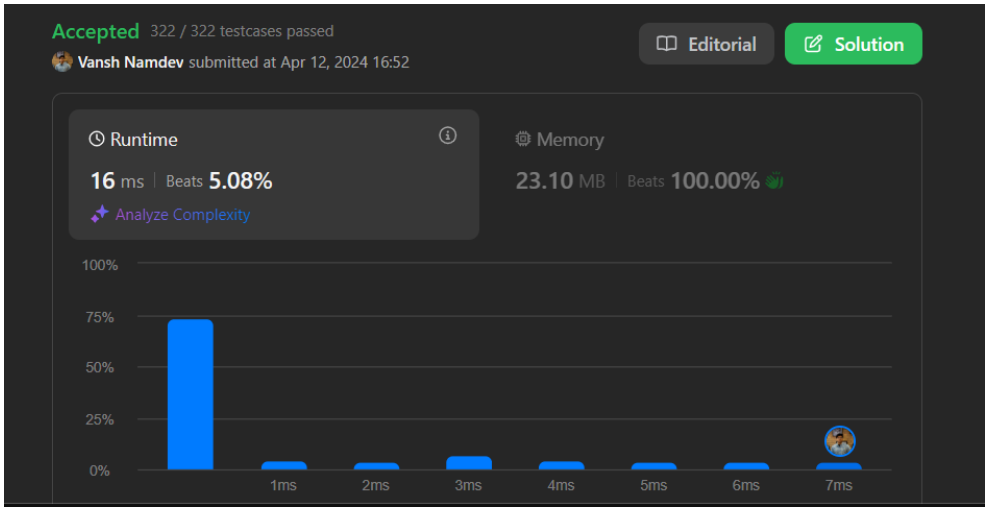
4. Divide Two Integers

```
class Solution {
public:
    long long divide(int dividend, int divisor) {
        long long quotiont=(long double)dividend/divisor;
        cout<<quotiont<<endl;
        if(quotiont<pow(-2,31))
        {
            return pow(-2,31);
        }
        else if(quotiont>pow(2,31)-1)
        {
            return pow(2,31)-1;
        }
        return quotiont;
    }`};
```



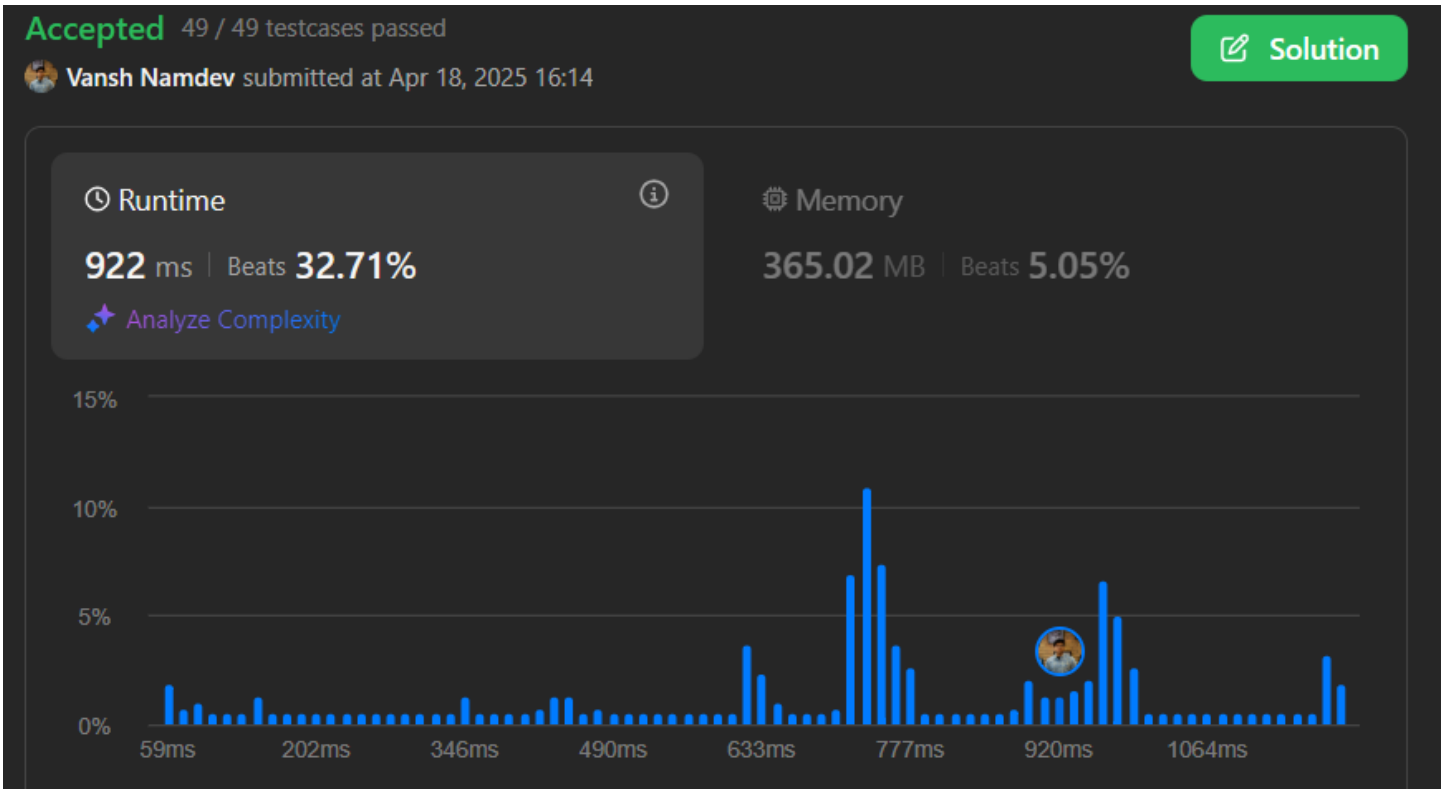
5. Trapping Rain Water

```
class Solution {
public:
    int trap(vector<int>& height) {
        const int n=height.size();
        vector<int> leftMax(n);
        vector<int> rightMax(n);
        int ans=0;

        for(int i=0;i<n;i++){
            leftMax[i]= i==0? height[i]:max(height[i],leftMax[i-1]);
        }
        for(int i=n-1;i>=0;--i){
            rightMax[i]= i==n-1? height[i]:max(height[i],rightMax[i+1]);
        }
        for(int i=0;i<n;i++){
            ans+=min(leftMax[i],rightMax[i])-height[i];
        }
    return ans;
    }
```



6. Max Number of Tasks You Can Assign

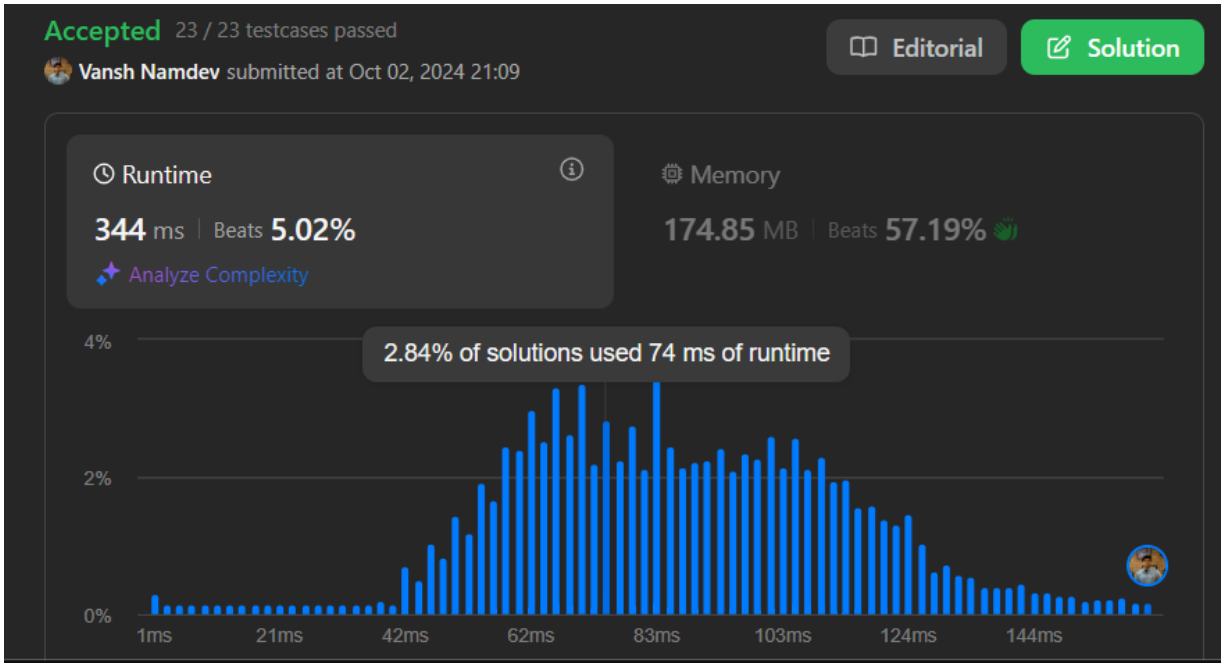
```
class Solution {
public:
    int check(vector<int> &tasks, int take, map<int,int> count, int pills, int power) {
        while (take >= 1 && count.size()) {
            auto it = count.end(); --it;

            if (tasks[take - 1] <= it->first) {}
            else if (pills) {
                it = count.lower_bound(tasks[take - 1] - power);
                if (it == count.end()) return 0;
                --pills;
            }
            else return 0;

            --take;
            (it->second)--;
            if (it->second == 0)
                count.erase(it);
        }
        return take == 0;
    }

    int maxTaskAssign(vector<int>& t, vector<int>& w, int p, int s) {
        int n = t.size();
        int m = w.size();
        sort(t.begin(), t.end());
        map<int,int> Count;
        for (auto &strength : w) Count[strength]++;

        int l = 0, r = n, ans = 0;

        while (l <= r) {
            int mid = l + (r - l) / 2;
            int chk = check(t, mid, Count, p, s);
            if (chk) {
                ans = mid;
                l = mid + 1;
            }
            else {
                r = mid - 1;
            }
        }
        return ans;
    }
};
```



7. LRU Cache

```
class Node{
public:
    int key;
    int val;
    Node* prev = nullptr;
    Node* next = nullptr;

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

class LRUCache {
public:
    int cap;
    unordered_map<int, Node*> cache;
    Node* left;
    Node* right;

    LRUCache(int capacity) {
        cap = capacity;
        left = new Node(0, 0);  // Dummy node for the head
        right = new Node(0, 0); // Dummy node for the tail
        left->next = right;
        right->prev = left;
```

```
    }

    // Function to remove a node from the linked list
    void remove(Node* node) {
        Node* pre = node->prev;
        Node* nex = node->next;
        pre->next = nex;
        nex->prev = pre;
    }

    // Function to insert a node at the right end (most recent position)
    void insert(Node* node) {
        Node* pre = right->prev;
        Node* nex = right;
        pre->next = node;
        node->prev = pre;
        node->next = nex;
        nex->prev = node;
    }

    // Function to get the value of a key from the cache
    int get(int key) {
        if (cache.find(key) != cache.end()) {
            // Move the accessed node to the most recent position
            remove(cache[key]);
            insert(cache[key]);
            return cache[key]->val;
        }
        return -1; // Key not found
    }

    // Function to add a key-value pair to the cache
    void put(int key, int value) {
        if (cache.find(key) != cache.end()) {
            // Remove the old node if it already exists
            remove(cache[key]);
        }
        cache[key] = new Node(key, value);
        insert(cache[key]);

        // Check if the cache exceeded its capacity
        if (cache.size() > cap) {
            Node* LRU = left->next; // Least Recently Used is at the front (left)
            remove(LRU);
            cache.erase(LRU->key);
            delete LRU;
        }
    }
};

/**
 * Your LRUCache object will be instantiated and called as such:
 * LRUCache* obj = new LRUCache(capacity);
 * int param_1 = obj->get(key);
 * obj->put(key, value);
                                                                */
```


Accepted 23 / 23 testcases passed
Vansh Namdev submitted at Oct 02, 2024 21:09
Editorial
Solution
Runtime
344 ms | Beats 5.02%
Analyze Complexity
Memory
174.85 MB | Beats 57.19%
2.84% of solutions used 74 ms of runtime
4%
2%
0%
1ms
21ms
42ms
62ms
83ms
103ms
124ms
144ms