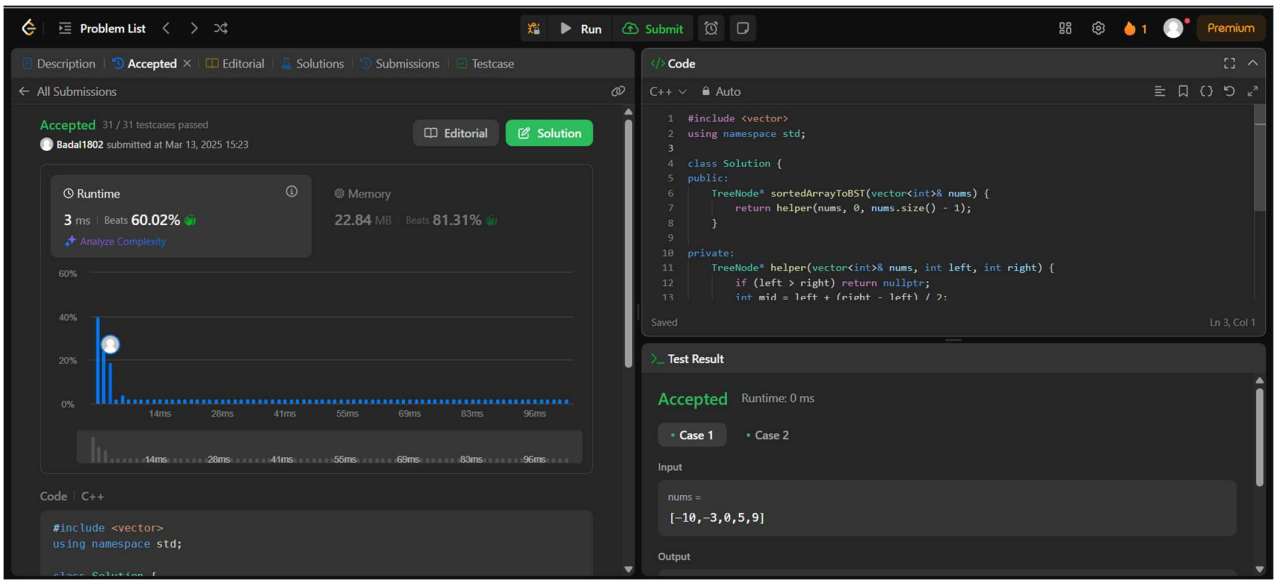
108.Convert Sorted Array to Binary Search Tree

CODE:

```
#include <vector>
using namespace std;
class Solution {
public:
    TreeNode* sortedArrayToBST(vector<int>& nums) {
        return helper(nums, 0, nums.size() - 1);
    }
private:
    TreeNode* helper(vector<int>& nums, int left, int right) {
        if (left > right) return nullptr;
        int mid = left + (right - left) / 2;
        TreeNode* root = new TreeNode(nums[mid]);
        root->left = helper(nums, left, mid - 1);
        root->right = helper(nums, mid + 1, right);
        return root;
    }
};
```

OUTPUT:

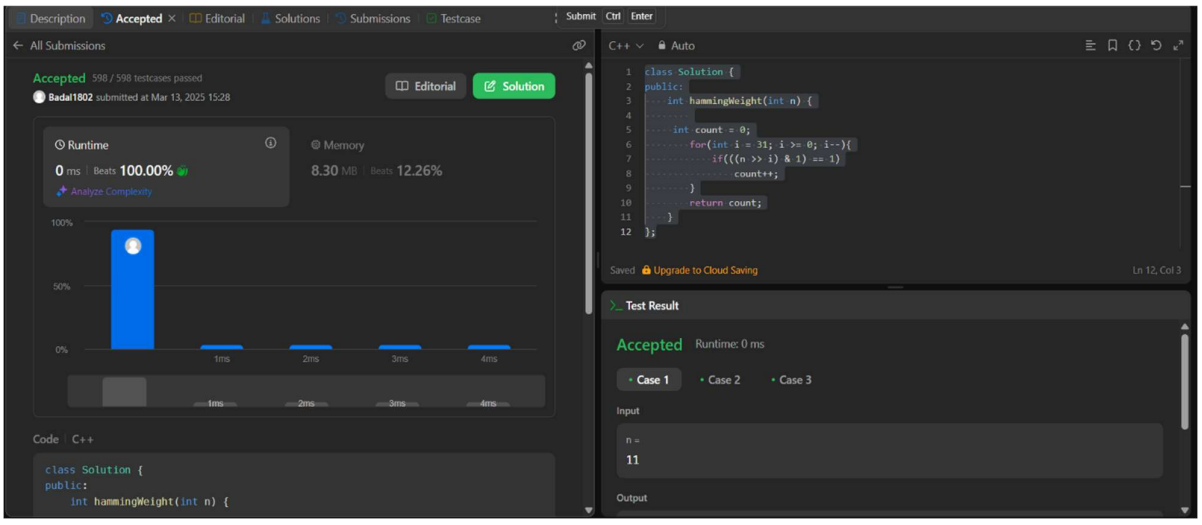


191.Number of 1 Bits

CODE:

```
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;
    }
};
```

OUTPUT:



912.Sort an Array

CODE:

```
class Solution {
public:
   void merge(vector<int>&nums,int start,int mid,int end)
   {
      vector<int>temp(end-start+1);
      int left =start,right = mid+1,index =0;
      while(left<=mid && right<=end)
      {
         if(nums[left]<=nums[right])
         {
            temp[index]=nums[left];
            index++,left++;
         }
         else
         {
            temp[index]=nums[right];
            index++,right++;
         }
      }
      //if left array remains
      while(left<=mid)
      {
         temp[index]=nums[left];
         index++,left++;
      }
      //if right array remains
      while(right<=end)
      {
         temp[index]=nums[right];
```

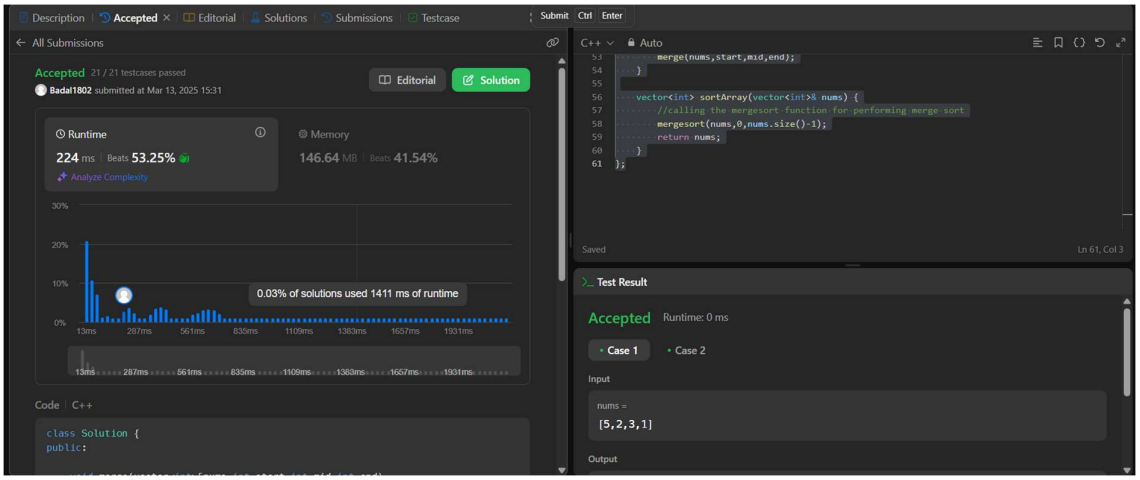
```
            index++,right++;
        }
        //put the values in the original array
        index = 0;
        while(start<=end)
        {
            nums[start] = temp[index];
            start++,index++;
        }
    }

    void mergesort(vector<int>&nums,int start,int end)
    {
        if(start==end)
        return;

        int mid = start+(end-start)/2;
        //leftside
        mergesort(nums,start,mid);
        //rightside
        mergesort(nums,mid+1,end);
        //merge
        merge(nums,start,mid,end);
    }

    vector<int> sortArray(vector<int>& nums) {
        //calling the mergesort function for performing merge sort
        mergesort(nums,0,nums.size()-1);
        return nums;
    }
};
```

OUTPUT:

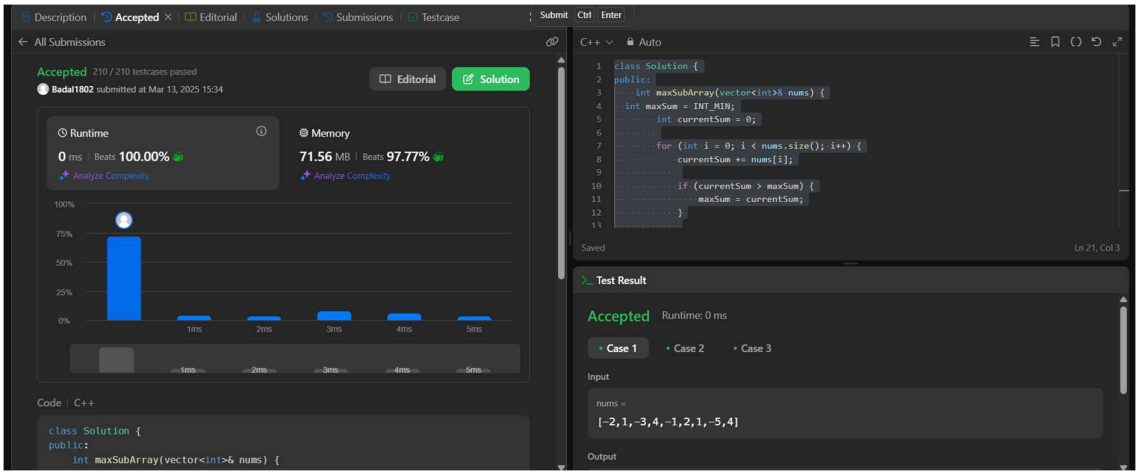


53.[Maximum Subarray]

CODE:

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
  int maxSum = INT_MIN;
        int currentSum = 0;
        for (int i = 0; i < nums.size(); i++) {
            currentSum += nums[i];
            if (currentSum > maxSum) {
                maxSum = currentSum;
            }
            if (currentSum < 0) {
                currentSum = 0;
            }
        }
        return maxSum;
    }
};
```

OUTPUT:



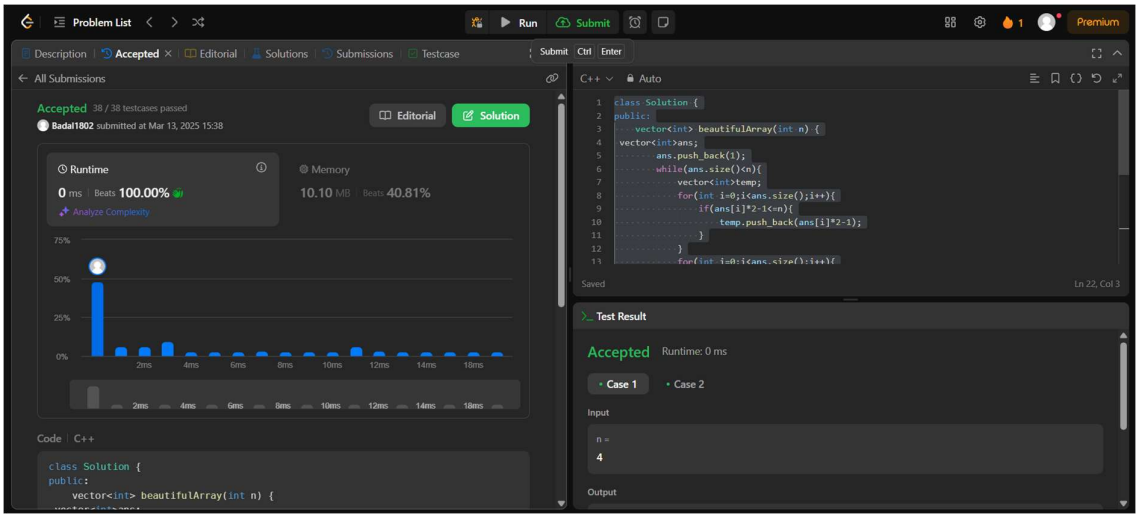
932.[Beautiful Array]

CODE:

```
class Solution {
public:
    vector<int> beautifulArray(int n) {
 vector<int>ans;
        ans.push_back(1);
        while(ans.size()<n){
            vector<int>temp;
            for(int i=0;i<ans.size();i++){
                if(ans[i]*2-1<=n){
                    temp.push_back(ans[i]*2-1);
                }
            }
            for(int i=0;i<ans.size();i++){
                if(ans[i]*2<=n){
                    temp.push_back(ans[i]*2);
                }
            }
```

```
            ans=temp;
        }
        return ans;
    }
};
```

OUTPUT:



372.[Super Pow]

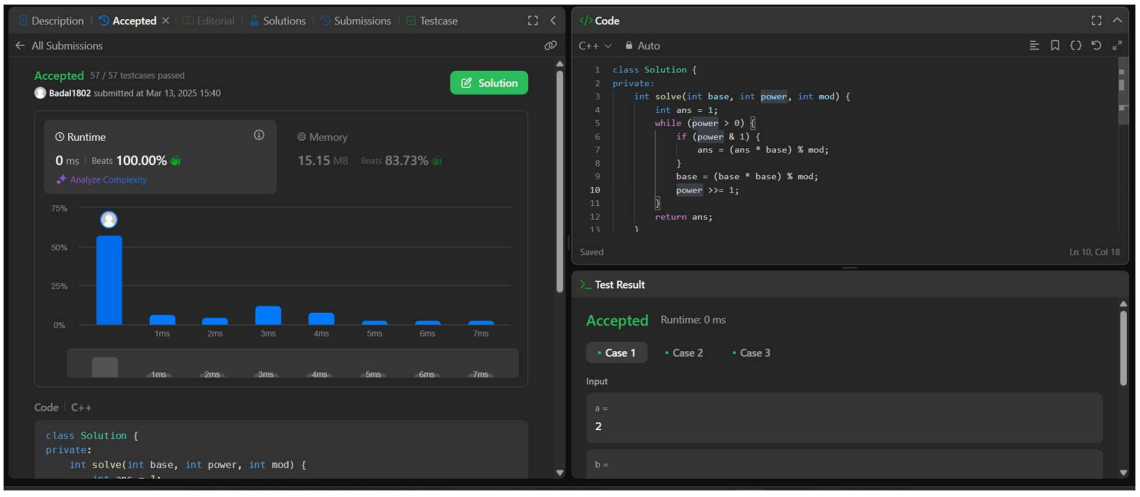
CODE:

```
class Solution {
private:
    int solve(int base, int power, int mod) {
        int ans = 1;
        while (power > 0) {
            if (power & 1) {
                ans = (ans * base) % mod;
            }
            base = (base * base) % mod;
            power >>= 1;
```

```
        }
        return ans;
    }

public:
    int superPow(int a, vector<int>& b) {
        a%=1337;
        int n = b.size();
        int m = 1140;
        int expi = 0;
        for(int i : b){
            expi = (expi*10+i)%m;
        }
        if (expi == 0) {
            expi = m;
        }
        return solve(a,expi,1337);
    }
};
```

OUTPUT:



218.The Skyline Problem

CODE:

```
class Solution {
public:
   vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
int edge_idx = 0;
      vector<pair<int, int>> edges;
      priority_queue<pair<int, int>> pq;
      vector<vector<int>> skyline;

      for (int i = 0; i < buildings.size(); ++i) {
         const auto &b = buildings[i];
         edges.emplace_back(b[0], i);
         edges.emplace_back(b[1], i);
      }

      std::sort(edges.begin(), edges.end());

      while (edge_idx < edges.size()) {
         int curr_height;
         const auto &[curr_x, _] = edges[edge_idx];
         while (edge_idx < edges.size() &&
                curr_x == edges[edge_idx].first) {
            const auto &[_, building_idx] = edges[edge_idx];
            const auto &b = buildings[building_idx];
            if (b[0] == curr_x)
               pq.emplace(b[2], b[1]);
            ++edge_idx;
         }
         while (!pq.empty() && pq.top().second <= curr_x)
            pq.pop();
```

```
            curr_height = pq.empty() ? 0 : pq.top().first;
            if (skyline.empty() || skyline.back()[1] != curr_height)
                skyline.push_back({curr_x, curr_height});
        }
        return skyline;
    }
};
```

OUTPUT: