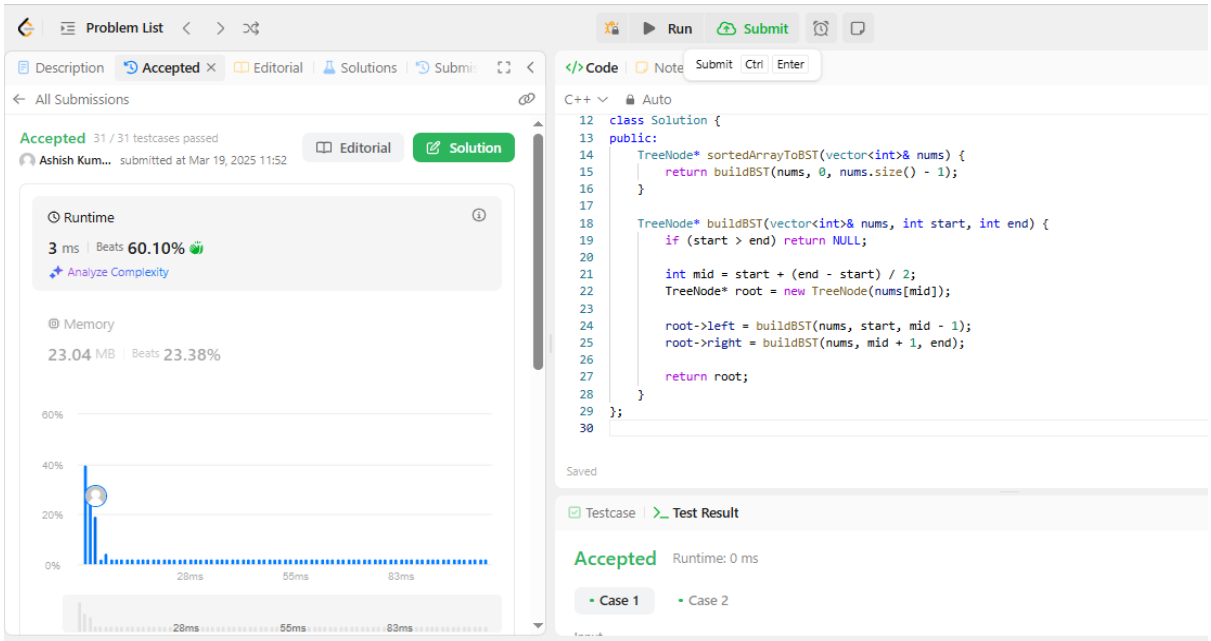
Ashish Kumar Saurav  22BCS16682

# Assignment 6

## 108.Convert Sorted Array to Binary Search Tree

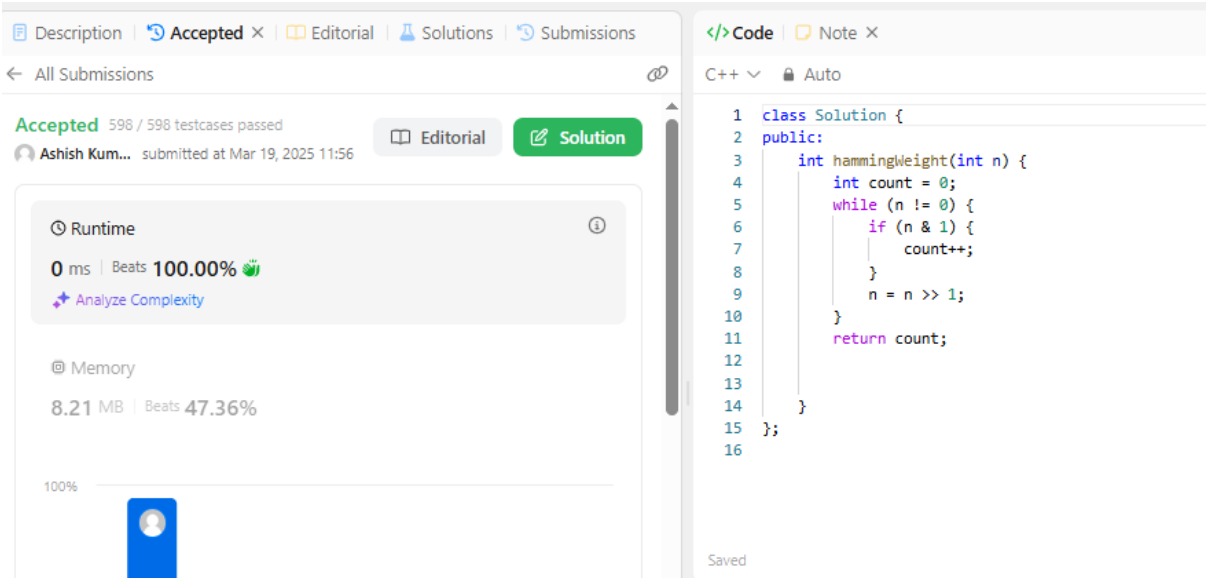
```cpp
class Solution {
public:
    TreeNode* sortedArrayToBST(vector<int>& nums) {
        return buildBST(nums, 0, nums.size() - 1);
    }

    TreeNode* buildBST(vector<int>& nums, int start, int end) {
        if (start > end) return NULL;

        int mid = start + (end - start) / 2;
        TreeNode* root = new TreeNode(nums[mid]);

        root->left = buildBST(nums, start, mid - 1);
        root->right = buildBST(nums, mid + 1, end);

        return root;
    }
};
```




## 191.Number of 1 Bits

```cpp
class Solution {
public:
```

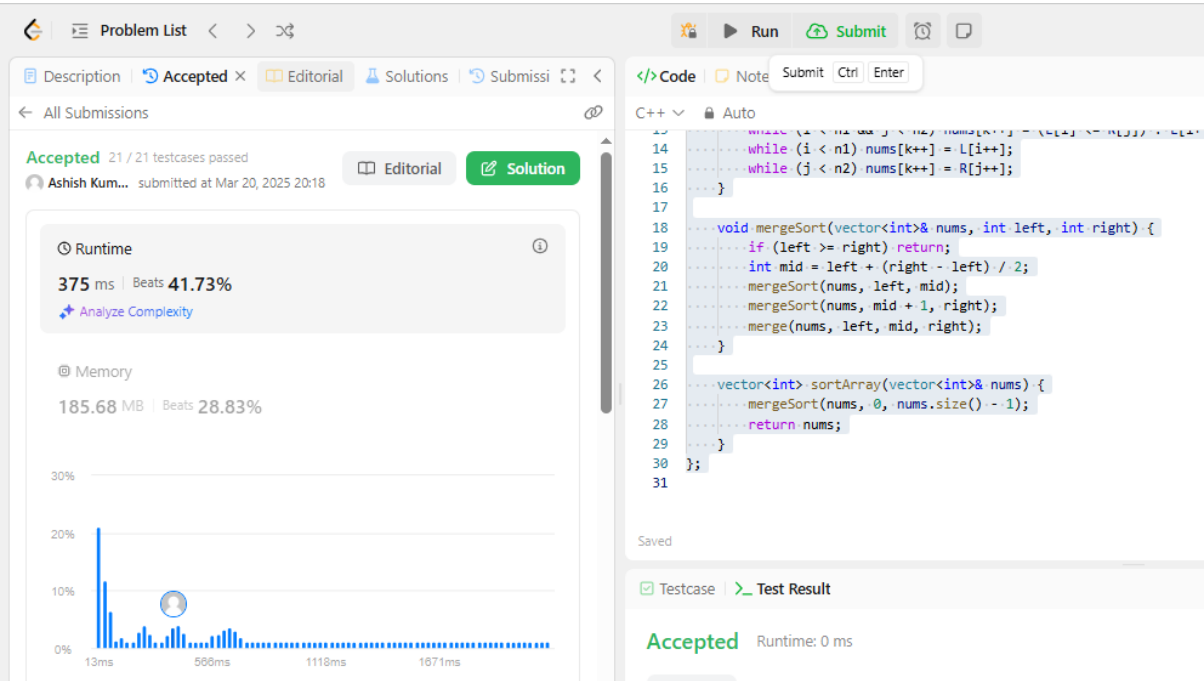
```
    int hammingWeight(int n) {
        int count = 0;
        while (n != 0) {
            if (n & 1) {
                count++;
            }
            n = n >> 1;
        }
        return count;


    }
};
```




## 912. Sort an Array

```
#include <vector>

using namespace std;

class Solution {
public:
    void merge(vector<int>& nums, int left, int mid, int right) {
        int n1 = mid - left + 1, n2 = right - mid;
        vector<int> L(n1), R(n2);
        for (int i = 0; i < n1; i++) L[i] = nums[left + i];
        for (int i = 0; i < n2; i++) R[i] = nums[mid + 1 + i];
```

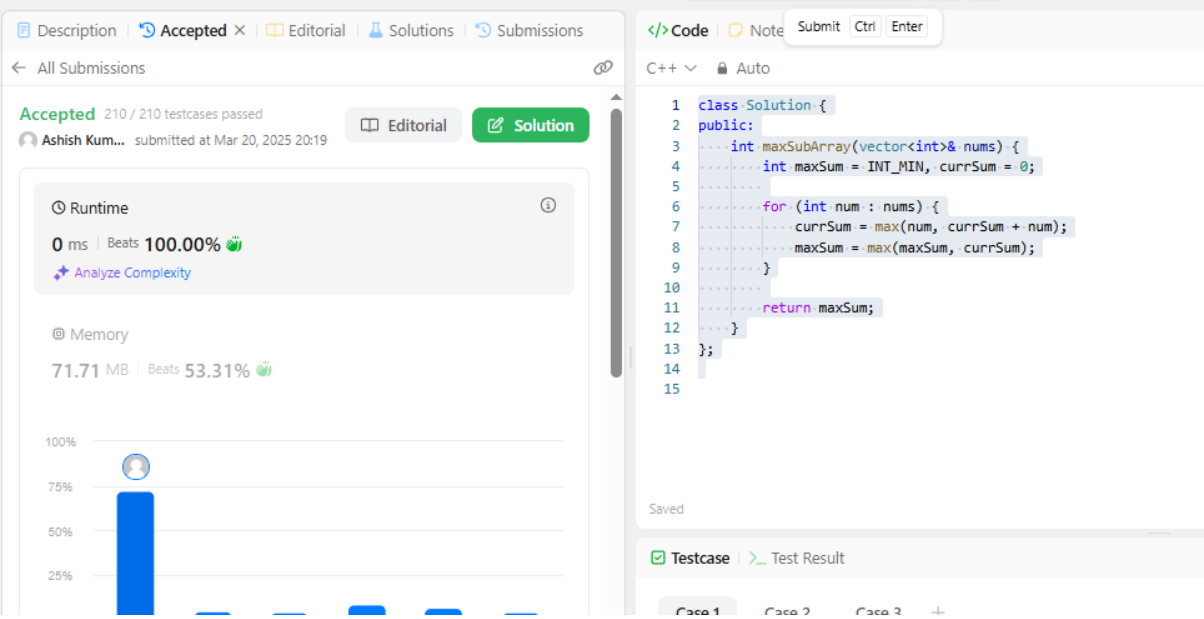
```
        int i = 0, j = 0, k = left;
        while (i < n1 && j < n2) nums[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];
        while (i < n1) nums[k++] = L[i++];
        while (j < n2) nums[k++] = R[j++];
    }

    void mergeSort(vector<int>& nums, int left, int right) {
        if (left >= right) return;
        int mid = left + (right - left) / 2;
        mergeSort(nums, left, mid);
        mergeSort(nums, mid + 1, right);
        merge(nums, left, mid, right);
    }

    vector<int> sortArray(vector<int>& nums) {
        mergeSort(nums, 0, nums.size() - 1);
        return nums;
    }
};
```




## 53. Maximum Subarray

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int maxSum = INT_MIN, currSum = 0;

        for (int num : nums) {
            currSum = max(num, currSum + num);
```

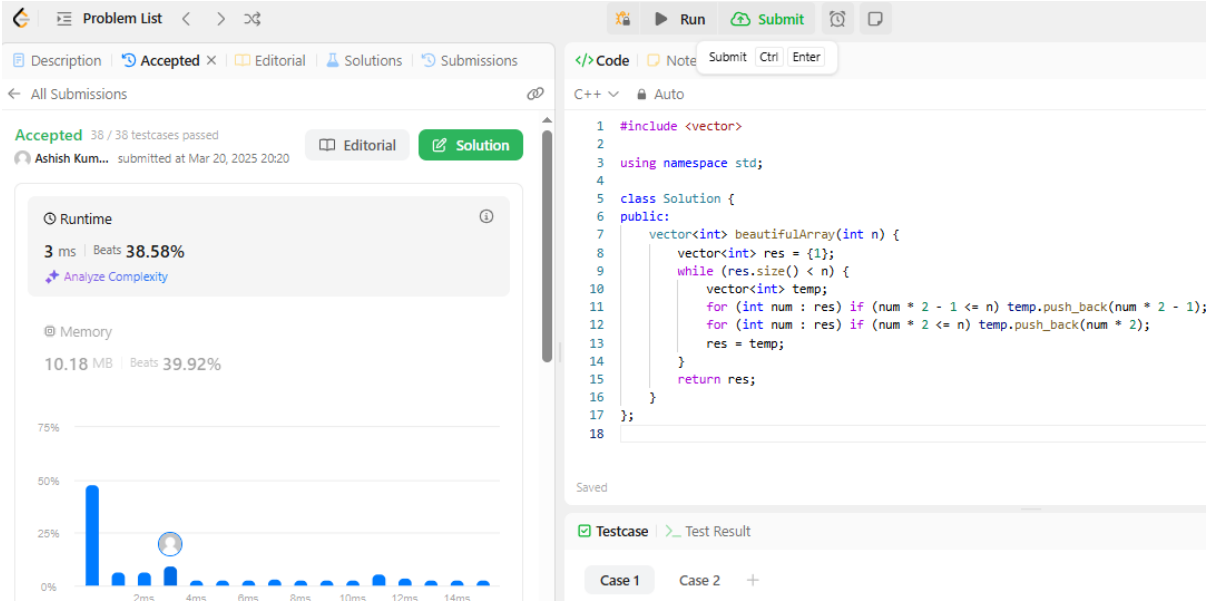
```
            maxSum = max(maxSum, currSum);
        }

        return maxSum;
    }
};
```




## 932. Beautiful Array

```
#include <vector>


using namespace std;


class Solution {
public:
    vector<int> beautifulArray(int n) {
        vector<int> res = {1};
        while (res.size() < n) {
            vector<int> temp;
            for (int num : res) if (num * 2 - 1 <= n) temp.push_back(num * 2 - 1);
            for (int num : res) if (num * 2 <= n) temp.push_back(num * 2);
            res = temp;
```

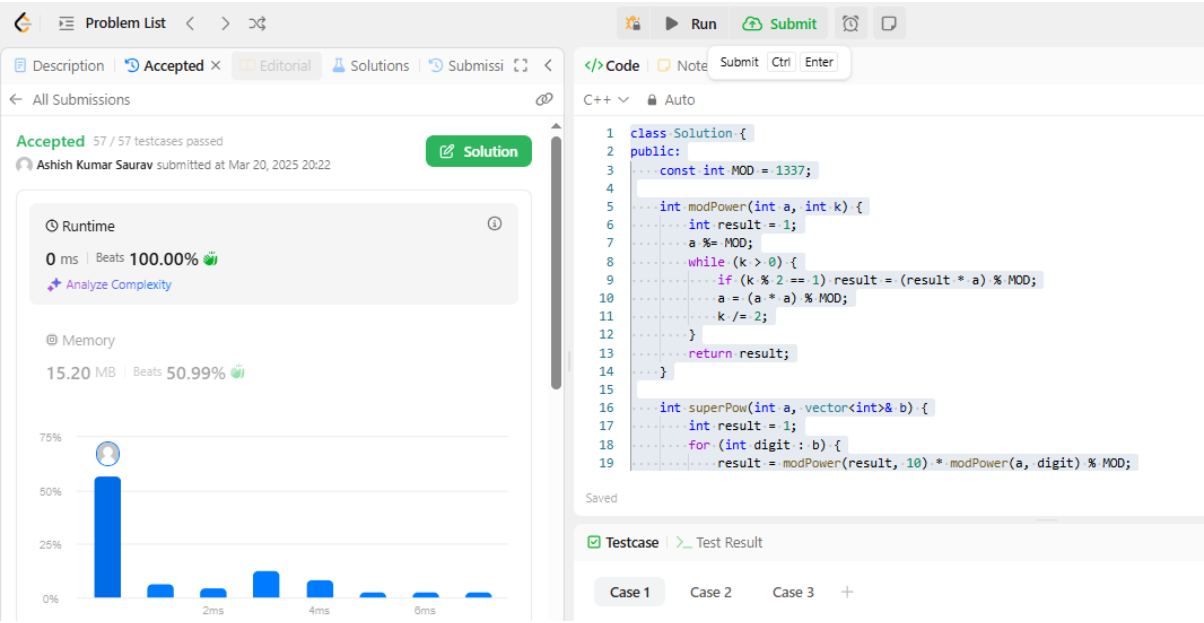
```
        }
        return res;
    }
};
```




## 372. Super Pow

```
class Solution {
public:
    const int MOD = 1337;

    int modPower(int a, int k) {
        int result = 1;
        a %= MOD;
        while (k > 0) {
            if (k % 2 == 1) result = (result * a) % MOD;
            a = (a * a) % MOD;
            k /= 2;
        }
        return result;
    }

    int superPow(int a, vector<int>& b) {
        int result = 1;
        for (int digit : b) {
            result = modPower(result, 10) * modPower(a, digit) % MOD;
        }
        return result;
    }
};
```

Ashish Kumar Saurav  22BCS16682




## 218. The Skyline Problem

```
#include <vector>

#include <set>

#include <map>


using namespace std;


class Solution {

public:

   vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {

      vector<pair<int, int>> events;

      for (auto& b : buildings) {

         events.emplace_back(b[0], -b[2]);

         events.emplace_back(b[1], b[2]);

      }


      sort(events.begin(), events.end());

      multiset<int> heights = {0};

      vector<vector<int>> result;
```

Ashish Kumar Saurav  22BCS16682

```
        int prevHeight = 0;

        for (auto& [x, h] : events) {
            if (h < 0) heights.insert(-h);
            else heights.erase(heights.find(h));

            int currHeight = *heights.rbegin();
            if (currHeight != prevHeight) {
                result.push_back({x, currHeight});
                prevHeight = currHeight;
            }
        }

        return result;
    }
};
```