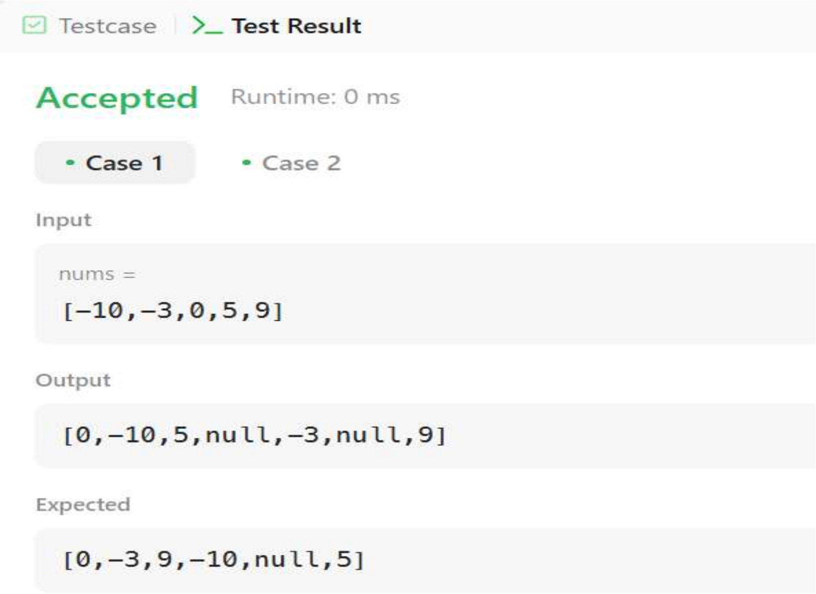
Name : Himanshu Rawat UID : 22BCS11411 Section : 605-B Assignment-6

Ques 1. Convert Sorted Array to Binary Search Tree.

Code:

```
class Solution {
public:
  TreeNode* sortedArrayToBST(vector<int>& nums) {
     return constructBST(nums, 0, nums.size() - 1);
  }
private:
  TreeNode* constructBST(vector<int>& nums, int left, int right) {
     if (left > right) return nullptr;
     int mid = left + (right - left) / 2; // Middle element as root
     TreeNode* root = new TreeNode(nums[mid]);
     root->left = constructBST(nums, left, mid - 1);   // Left subtree
     root->right = constructBST(nums, mid + 1, right); // Right subtree
     return root;
  }
};
```

Output:



Accepted Runtime: 0 ms

Input

```
nums =
[-10,-3,0,5,9]
```

Output

```
[0,-10,5,null,-3,null,9]
```

Expected

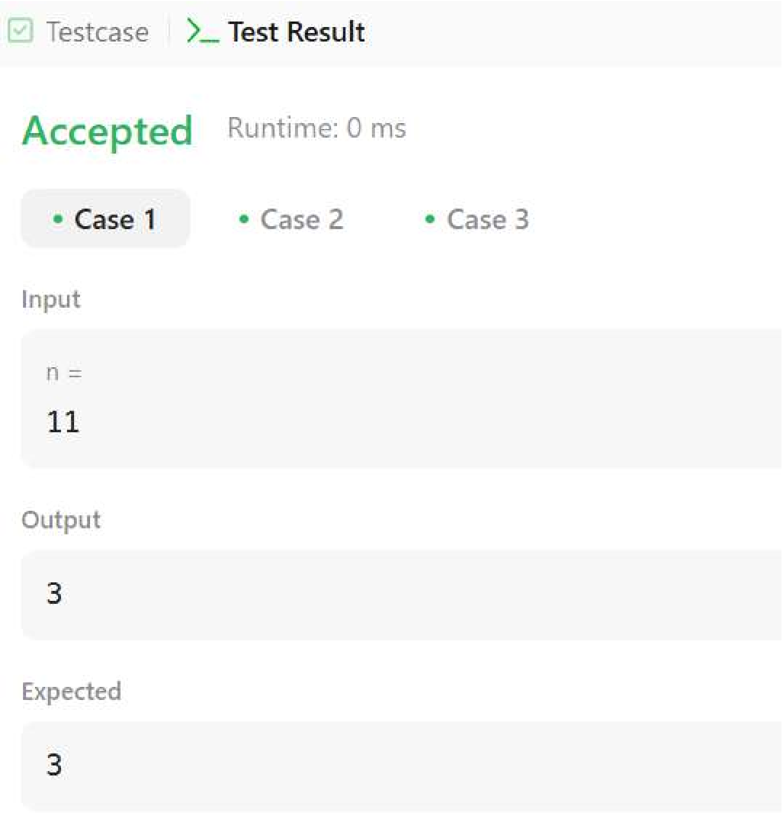
```
[0,-3,9,-10,null,5]
```

Ques 2. Number of 1 Bits.

Code:

```
class Solution {
public:
    int hammingWeight(int n) {
        int count = 0;
        while (n) {
            count += (n & 1);
            n >>= 1;
        }
        return count;
    }
};
```

Output:



Accepted Runtime: 0 ms

Input

n =

11

Output

3

Expected

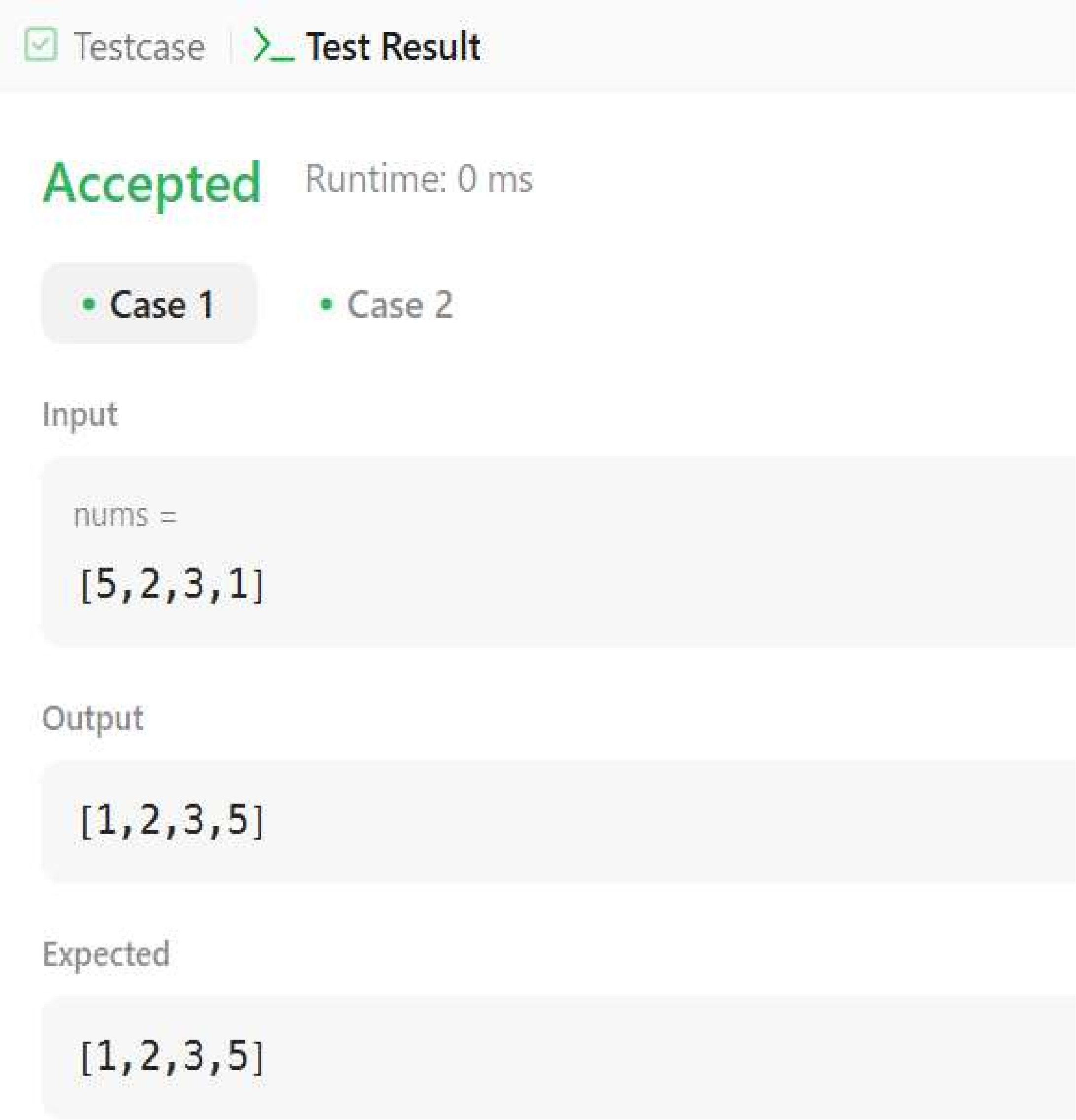
3

Ques 3. Sort an Array.

Code:

```
class Solution {
public:
    vector<int> sortArray(vector<int>& nums) {
        quickSort(nums, 0, nums.size() - 1);
        return nums;
    }

private:
    void quickSort(vector<int>& nums, int left, int right) {
        if (left >= right) return;

        int pivot = partition(nums, left, right);
        quickSort(nums, left, pivot - 1);
        quickSort(nums, pivot + 1, right);
    }

    int partition(vector<int>& nums, int left, int right) {
        int pivot = nums[right];
        int i = left - 1;

        for (int j = left; j < right; j++) {
            if (nums[j] < pivot) {
                swap(nums[++i], nums[j]);
            }
        }
        swap(nums[i + 1], nums[right]);
        return i + 1;
    }
};
```

Output:



Accepted Runtime: 0 ms

Input

nums =

```
[5,2,3,1]
```

Output

```
[1,2,3,5]
```

Expected

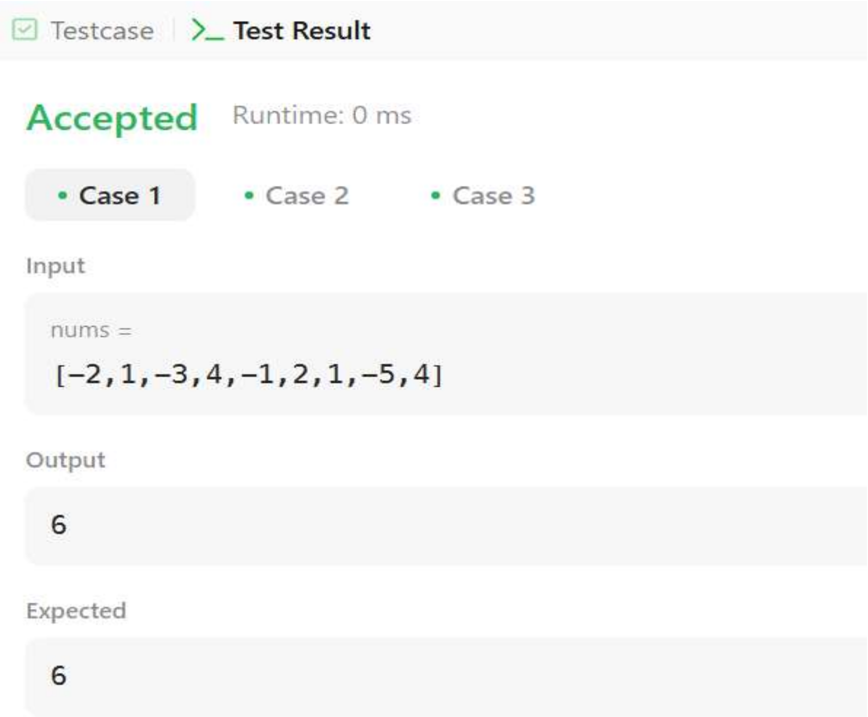
```
[1,2,3,5]
```

Ques 4. Maximum Subarray.

Code:

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

        for (int i = 1; i < nums.size(); i++) {
            currentSum = max(nums[i], currentSum + nums[i]);
            maxSum = max(maxSum, currentSum);
        }

        return maxSum;
    }
};
```

Output:



Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

```
nums =
[-2,1,-3,4,-1,2,1,-5,4]
```

Output

```
6
```

Expected

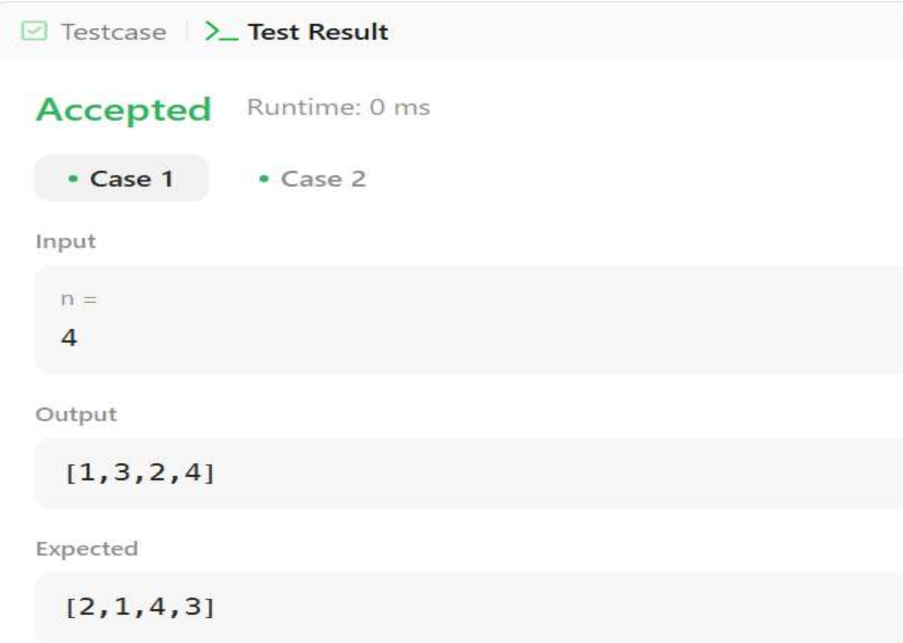
```
6
```

Ques 5. Beautiful Array.

Code:

```
class Solution {
public:
    vector<int> beautifulArray(int n) {
        if (n == 1) return {1};

        vector<int> odd = beautifulArray((n + 1) / 2);
        vector<int> even = beautifulArray(n / 2);

        vector<int> result;
        for (int num : odd) result.push_back(num * 2 - 1);
        for (int num : even) result.push_back(num * 2);

        return result;
    }
};
```

Output:



Accepted Runtime: 0 ms

Input

n =
4

Output

```
[1,3,2,4]
```

Expected

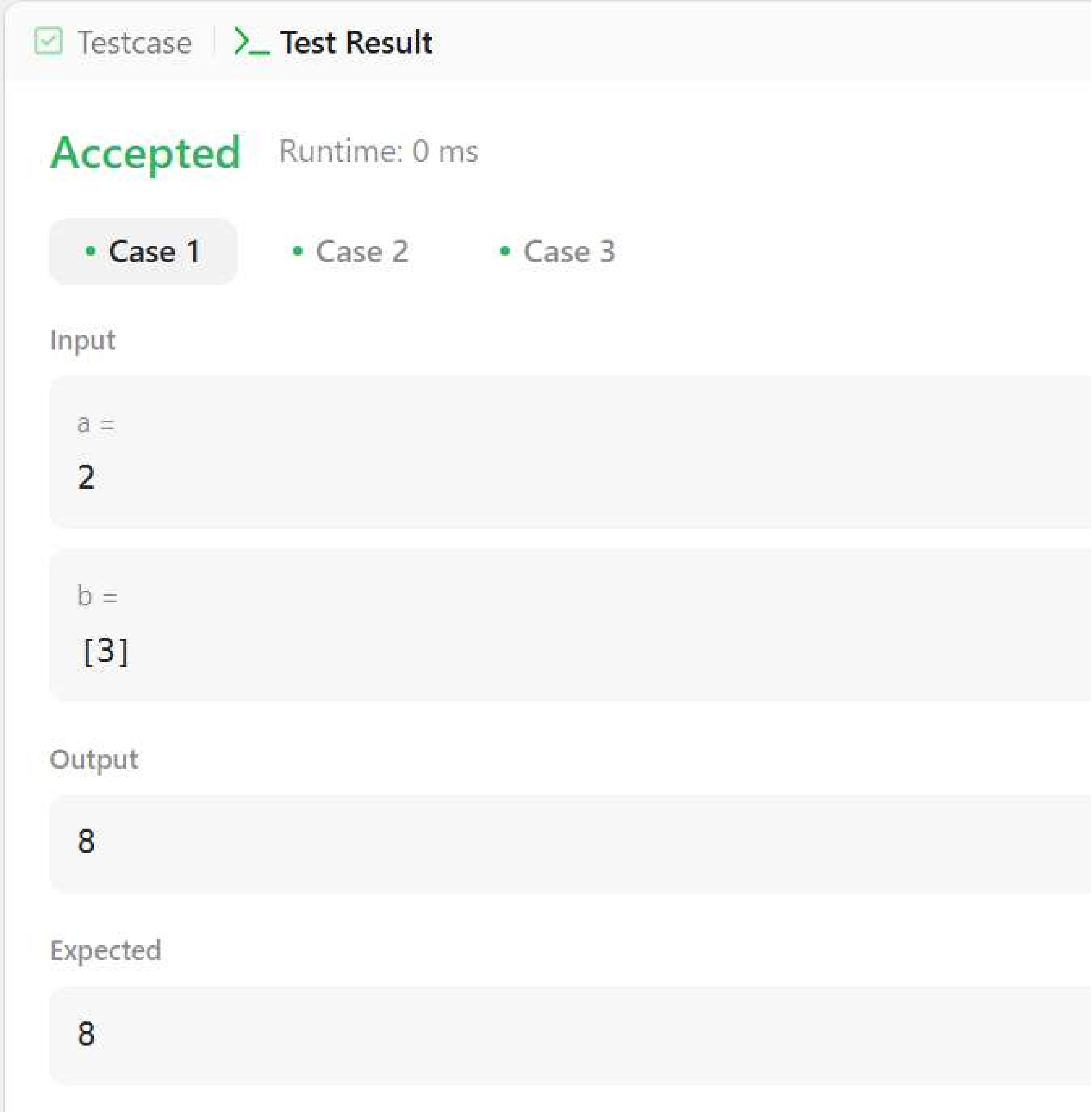
```
[2,1,4,3]
```

Ques 6. Super Pow.

Code:

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

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

    int superPow(int a, vector<int>& b) {
        a %= MOD;
        int result = 1;

        for (int digit : b) {
            result = (modPow(result, 10) * modPow(a, digit)) % MOD;
        }

        return result;
    }
};
```

Output:



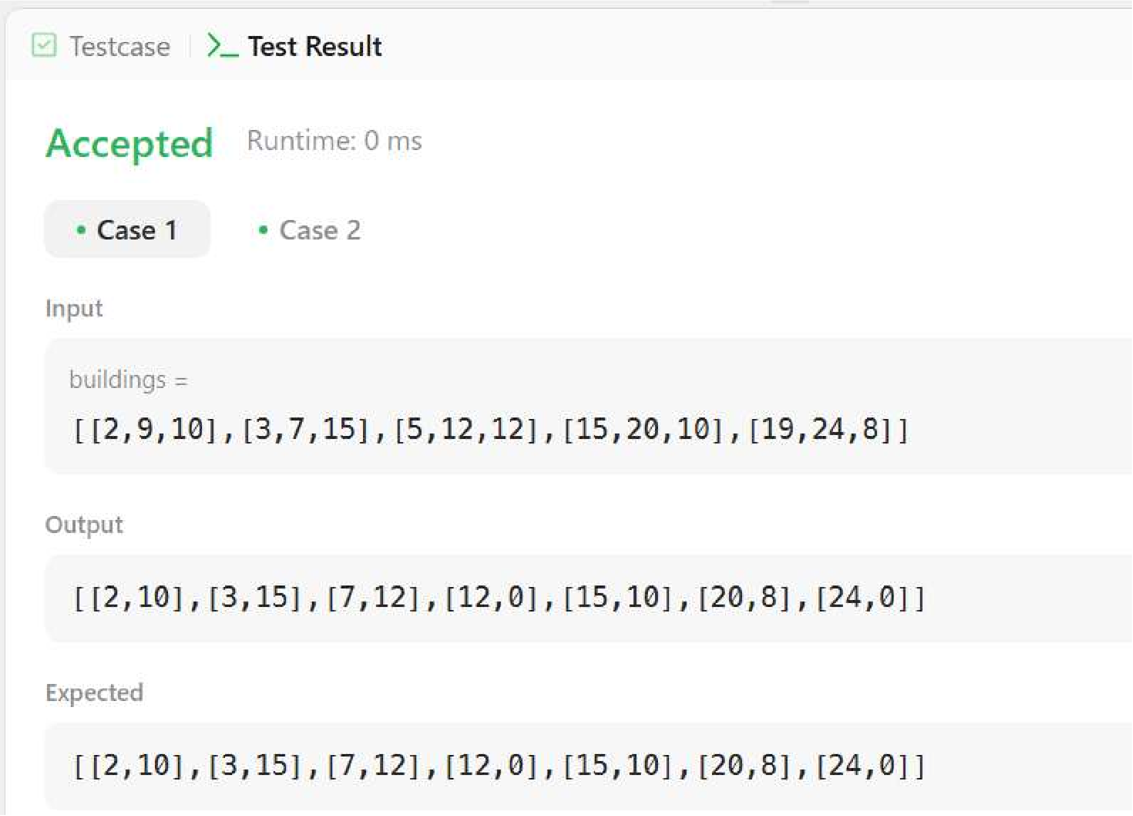
```
Accepted   Runtime: 0 ms

Input
a =
2
b =
[3]

Output
8

Expected
8
```

Ques.7 The Skyline Problem.

Code:

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<vector<int>> result;
        vector<pair<int, int>> events;
        for (const auto& b : buildings) {
            events.push_back({b[0], -b[2]});
            events.push_back({b[1], b[2]});
        }
        sort(events.begin(), events.end(), [](const pair<int, int>& a, const pair<int, int>& b) {
            if (a.first == b.first) {
                return a.second < b.second;
            }
            return a.first < b.first;
        });
        multiset<int> heights;
        heights.insert(0);
        int prevHeight = 0;
        for (const auto& event : events) {
            int x = event.first;
            int h = event.second;
            if (h < 0) {
                heights.insert(-h);
            } else {
                heights.erase(heights.find(h));
            }
            int currentHeight = *heights.rbegin();
            if (currentHeight != prevHeight) {
                result.push_back({x, currentHeight});
                prevHeight = currentHeight;
            }
        }
        return result;
    }
};
```

Output:



Accepted Runtime: 0 ms

Input

```
buildings =
[[2,9,10],[3,7,15],[5,12,12],[15,20,10],[19,24,8]]
```

Output

```
[[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]]
```

Expected

```
[[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]]
```