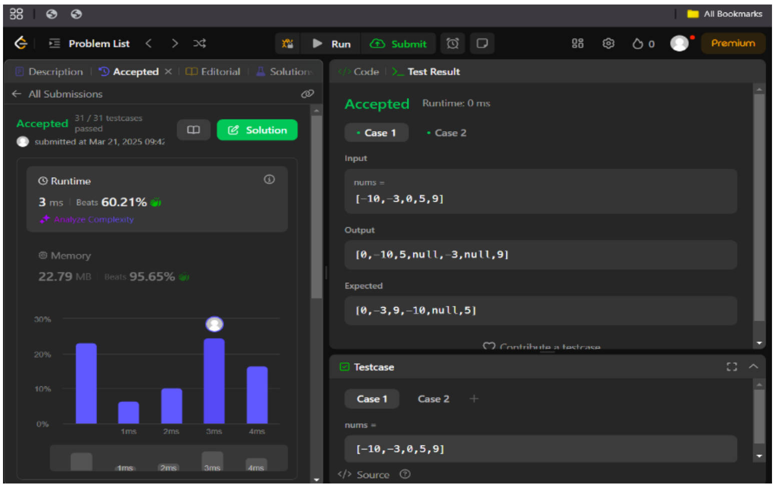
Name – Amitoj Singh UID – 22BCS13085 Section – IOT_605-B

Ques 1. 108. Convert Sorted Array to Binary Search Tree

Code:

```
#include <vector>

using namespace std;

class Solution {

public:

   TreeNode* sortedArrayToBST(vector<int>& nums) {

      return buildBST(nums, 0, nums.size() - 1);

   }

private:

   TreeNode* buildBST(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 = buildBST(nums, left, mid - 1);

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

      return root;

   }

};
```

Output:



Accepted 31 / 31 testcases passed

submitted at Mar 21, 2025 09:4

Runtime
3 ms | Beats 60.21%

Memory
22.79 MB | Beats 95.65%

Accepted Runtime: 0 ms

Case 1 Case 2

Input

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

Output

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

Expected

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

Testcase

Case 1 Case 2

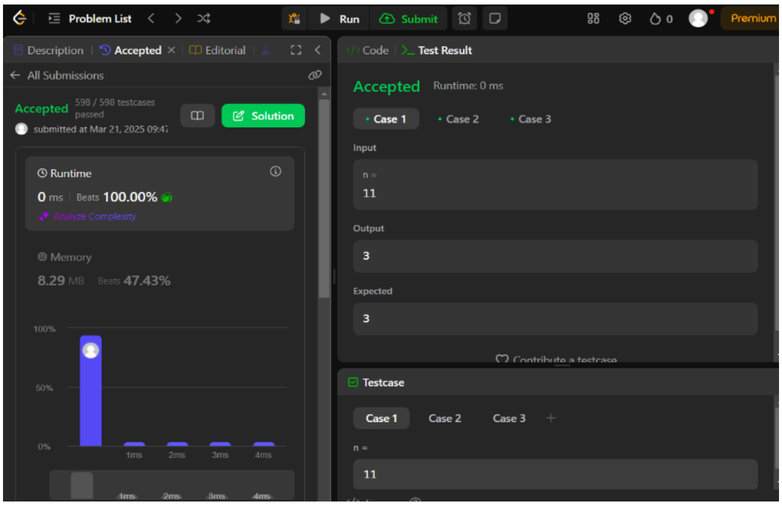
```
nums =
[-10,-3,0,5,9]
```

Ques 2. Number of 1 Bits

Code:

```
class Solution {
public:
    int hammingWeight(int n) {
        int count = 0;
        while (n) {
            n &= (n - 1); // Removes the rightmost '1' bit
            count++;
        }
        return count;
    }
};
```

Output:



Accepted 598 / 598 testcases passed

submitted at Mar 21, 2025 09:47

Runtime

0 ms | Beats 100.00%

Memory

8.29 MB | Beats 47.43%

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

n =

11

Output

3

Expected

3

Testcase

Case 1 Case 2 Case 3

n =

11

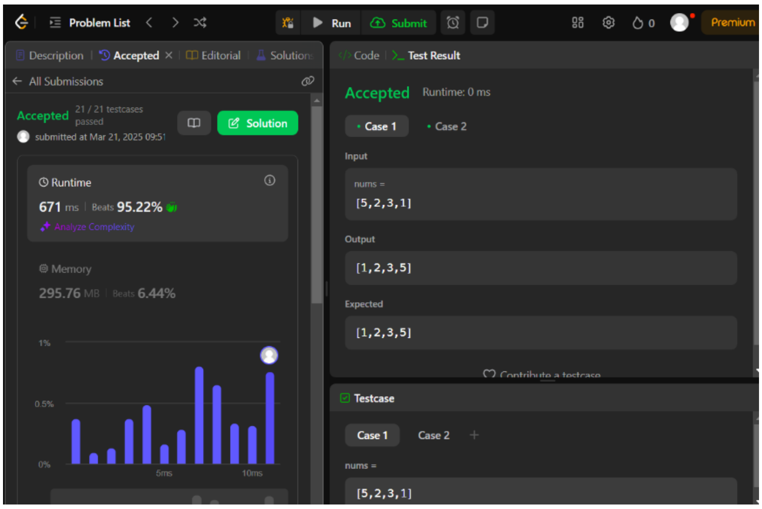
Ques 3. Sort an Array

Code:

```
class Solution {
public:
 int hammingWeight(int n) {
  int count = 0;
  while (n) {
   count += (n & 1); // Add 1 if the last bit is set
   n >>= 1; // Right shift n to check the next bit
  }
  return count;
 }
};class Solution {
public:
 vector<int> sortArray(vector<int>& nums) {
  mergeSort(nums, 0, nums.size() - 1);
  return nums; // Ensure function returns the sorted vector
 }
private:
 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);
 }
 void merge(vector<int>& nums, int left, int mid, int right) {
  vector<int> temp;
  int i = left, j = mid + 1;
  while (i <= mid && j <= right) {
   if (nums[i] < nums[j]) temp.push_back(nums[i++]);
   else temp.push_back(nums[j++]);
  }
  while (i <= mid) temp.push_back(nums[i++]);
  while (j <= right) temp.push_back(nums[j++]);
  for (int k = 0; k < temp.size(); ++k) {
```

```
            nums[left + k] = temp[k];
        }
    }
};
```

Output:



Accepted 21 / 21 testcases passed

submitted at Mar 21, 2025 09:51

Runtime

671 ms | Beats 95.22%

Memory

295.76 MB | Beats 6.44%

Accepted Runtime: 0 ms

Case 1 Case 2

Input

nums =

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

Output

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

Expected

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

nums =

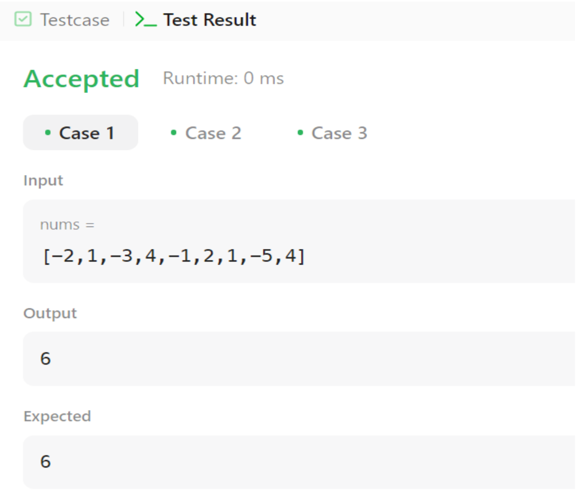
```
[5,2,3,1]
```

Ques 4. <u>Maximum Subarray.</u>

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:



Accepted Runtime: 0 ms

Input

nums =

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

Output

```
6
```

Expected

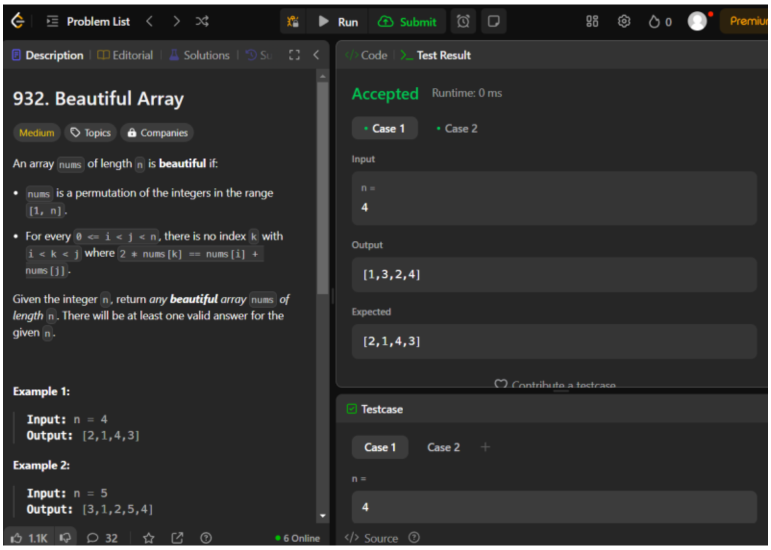
```
6
```

Ques 5. Beautiful Array

Code:

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

Output:



932. Beautiful Array

An array `nums` of length `n` is **beautiful** if:

- `nums` is a permutation of the integers in the range `[1, n]`.
- For every `0 <= i < j < n`, there is no index `k` with `i < k < j` where `2 * nums[k] == nums[i] + nums[j]`.

Given the integer `n`, return *any* ***beautiful*** *array* `nums` *of length* `n`. There will be at least one valid answer for the given `n`.

Example 1:

```
Input: n = 4
Output: [2,1,4,3]
```

Example 2:

```
Input: n = 5
Output: [3,1,2,5,4]
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

```
n =
4
```

Output

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

Expected

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

Case 1 Case 2 +

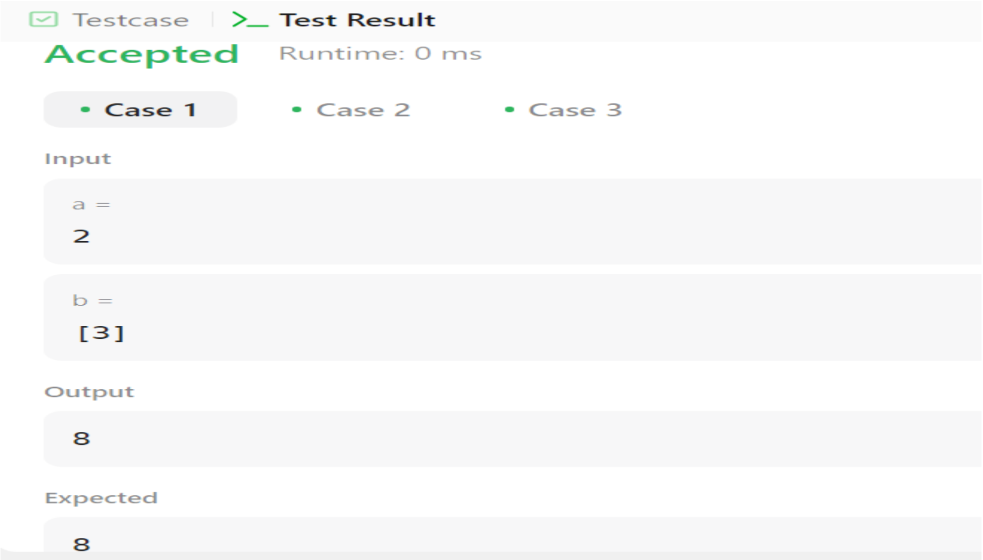
```
n =
4
```

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) { 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:



Testcase | >_ Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

```
a =
2
```

```
b =
[3]
```

Output

```
8
```

Expected

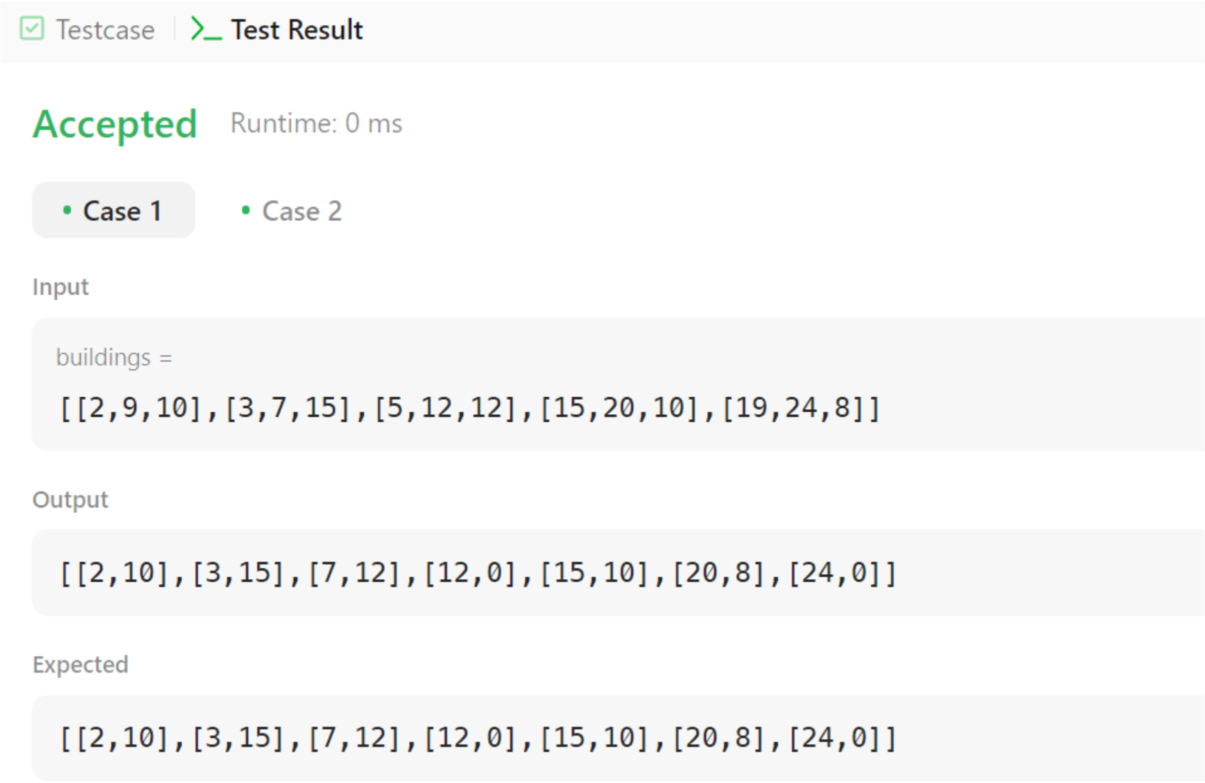
```
8
```

Ques 8. 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]}); // Left edge, add height
            events.push_back({b[1], b[2]});  // Right edge, remove height
        }
        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 the current height is different from the previous height, it's a key point
            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]]
```