**Advanced Programming 2**

**Assignment 6**

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

**Code:**

/\*\*

 \* Definition for a binary tree node.

 \* struct TreeNode {

 \*     int val;

 \*     TreeNode \*left;

 \*     TreeNode \*right;

 \*     TreeNode() : val(0), left(nullptr), right(nullptr) {}

 \*     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

 \*     TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}

 \* };

 \*/

class Solution {

public:

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

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

    }

private:

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

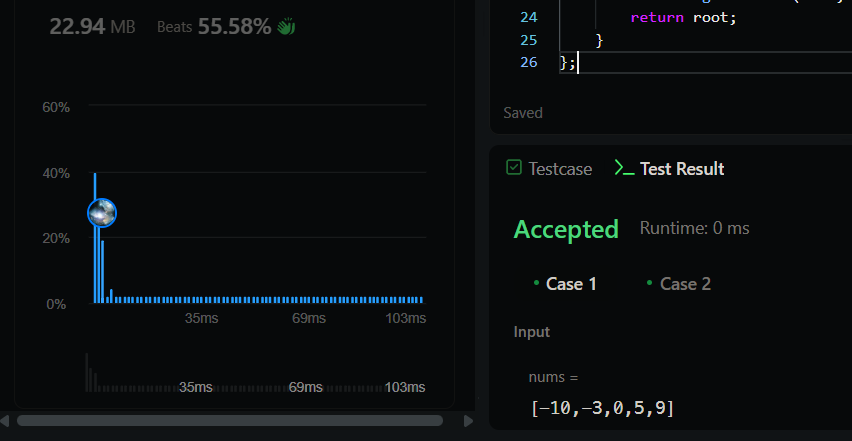
        root->right = Tree(nums,mid+1,right);

        return root;

    }

};

**Output:**

****

1. **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++;

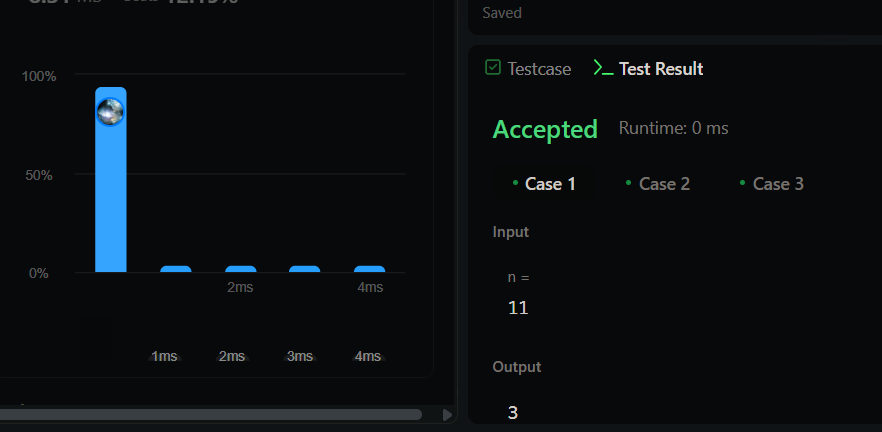
        }

        return count;

    }

};

**Output:**

****

1. **912.Sort an Array**

**Code:**

class Solution {

public:

vector<int> sortArray(vector<int>& nums) {

mergeSort(nums, 0, nums.size() - 1);

return nums;

}

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++) {

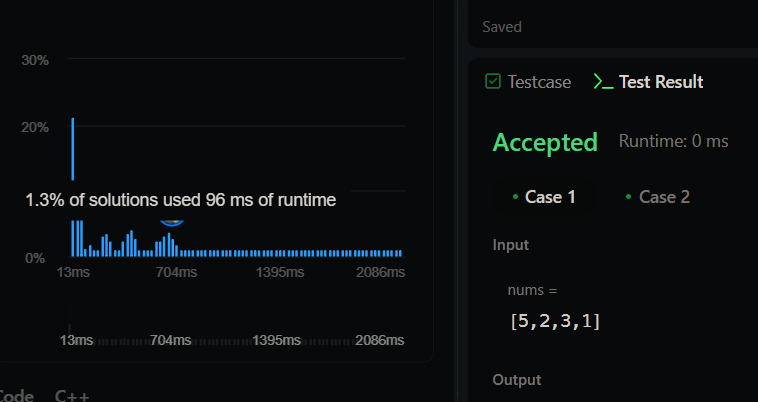
nums[left + k] = temp[k];

}

}

};

**Output:**

****

1. **53.Maximum Subarray**

**Code:**

class Solution {

public:

    int maxSubArray(vector<int>& nums) {

        int res = nums[0];

        int total = 0;

        for (int n : nums) {

            if (total < 0) {

                total = 0;

            }

            total += n;

            res = max(res, total);

        }

        return res;

    }

};

**Output:**

****

1. **932.Beautiful Array**

**Code:**

class Solution {

public:

    vector<int> beautifulArray(int n) {

        if(n==1)

            return {1};

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

        vector<int> odd = beautifulArray(n-(n/2));

        vector<int>ans;

        for(auto e:even)

            ans.push\_back(2\*e);

        for(auto e:odd)

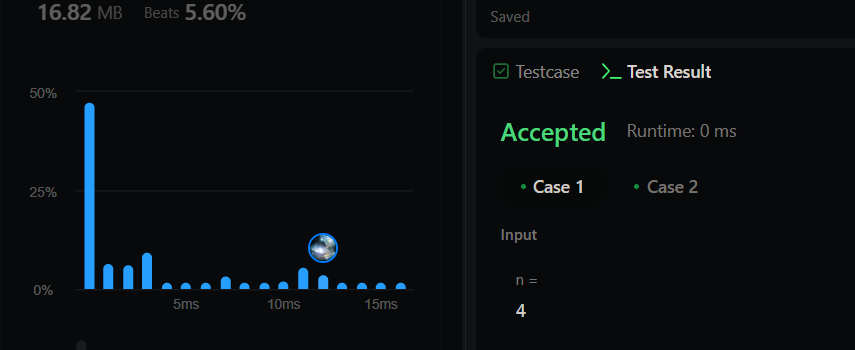
            ans.push\_back((2\*e)-1);

        return ans;

    }

};

**Output:**

****

1. **372.Super Pow**

**Code:**

class Solution {

    const int base = 1337;

    int powmod(int a, int k)

    {

        a %= base;

        int result = 1;

        for (int i = 0; i < k; ++i)

            result = (result \* a) % base;

        return result;

    }

public:

    int superPow(int a, vector<int>& b) {

        if (b.empty()) return 1;

        int last\_digit = b.back();

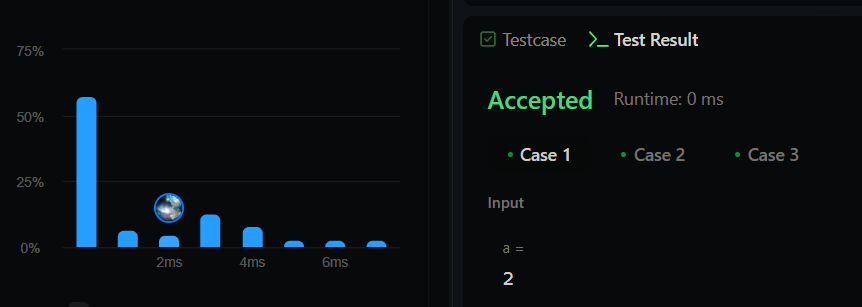
        b.pop\_back();

        return powmod(superPow(a, b), 10) \* powmod(a, last\_digit) % base;

    }

};

**Output:**

****

1. **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);

        }

        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:**

****