AP assignment 6

JATIN DHAKAR

Solution of problem 1-

#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);

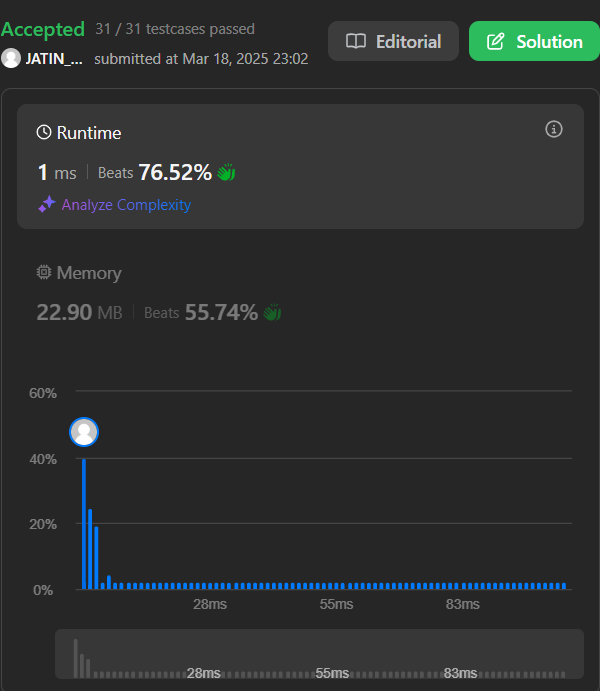
        root->right = helper(nums, mid + 1, right);

        return root;

    }

};

Output-



Solution of Problem 2-

class Solution {

public:

    int ans = 0;

    int hammingWeight(int n)

    {

        if(n == 0) return ans;

        if(n%2 == 1)ans++;

        hammingWeight(n/2);

        return ans;

    }

};

Output-

A screenshot of a computer

AI-generated content may be incorrect.

Solution of Problem 3-

class Solution {

public:

    void merge(vector<int>& nums,int s, int e){

        int m=(s+e)/2;

        vector<int>first(m-s+1),second(e-m);

        for (int i=0;i<first.size();i++){first[i]=nums[s+i];}

        for (int i=0;i<second.size();i++){second[i]=nums[m+1+i];}

        int i1=0,i2=0,maindex=s;

        while (i1<first.size() && i2<second.size()){

            if (first[i1]<second[i2]){nums[maindex++]=first[i1++];}

            else {nums[maindex++]=second[i2++];}

        }

        while (i1<first.size()){nums[maindex++]=first[i1++];}

        while (i2<second.size()){nums[maindex++]=second[i2++];}

    }

    void mergesort(vector<int>&nums, int s, int e){

        if (s>=e){return ;}

        int m=(s+e)/2;

        mergesort(nums,s,m);

        mergesort(nums,m+1,e);

        merge(nums,s,e);

    }

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

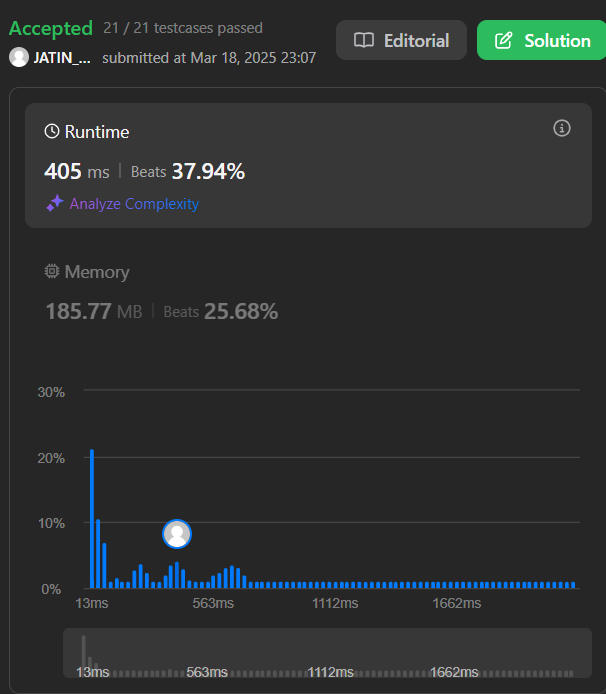
        mergesort(nums,0,nums.size()-1);

        return nums;

    }

};

Output-



Solution of Problem 4-

class Solution {

public:

    int maxSubArray(vector<int>& nums)

    {

        int res = nums[0];

        int cmax = 0;

        for(int val : nums)

        {

            cmax = max(val,cmax+val);

            res = max(res,cmax);

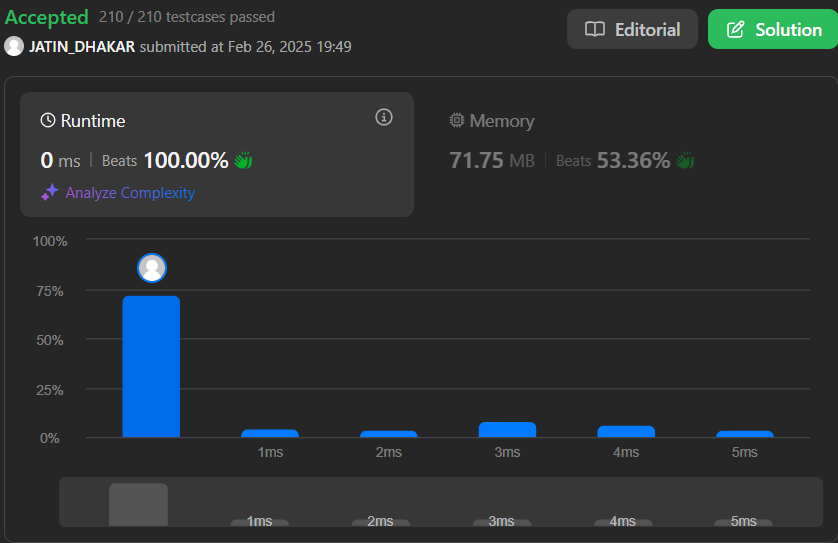
        }

        return res;

    }

};

Output-



Solution of Problem 5-

class Solution {

    const int base = 1337;

    int powmod(int a, int k) //a^k mod 1337 where 0 <= k <= 10

    {

        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-

A screenshot of a computer

AI-generated content may be incorrect.

Solution of Problem 6-

    vector<int> beautifulArray(int N) {

        vector<int> res = {1};

        while (res.size() < N) {

            vector<int> tmp;

            for (int i : res) if (i \* 2 - 1 <= N) tmp.push\_back(i \* 2 - 1);

            for (int i : res) if (i \* 2 <= N) tmp.push\_back(i \* 2);

            res = tmp;

        }

        return res;

    }

Output-

A screenshot of a computer

AI-generated content may be incorrect.

Solution of Problem 7-

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-

A screenshot of a black screen

AI-generated content may be incorrect.