Experiment – 2 (AP)

Q1. 1763.[Longest Nice Substring](https://leetcode.com/problems/longest-nice-substring/description/)

class Solution {

public:

string longestNiceSubstring(string s) {

if (s.size() < 2) return "";

unordered\_set<char> st(begin(s), end(s));

for (int i = 0; i < s.size(); i++) {

if (st.find((char) toupper(s[i])) == end(st) || st.find((char)tolower(s[i])) == end(st)) {

string s1 = longestNiceSubstring(s.substr(0, i));

string s2 = longestNiceSubstring(s.substr(i + 1));

return s1.size() >= s2.size() ? s1 : s2;

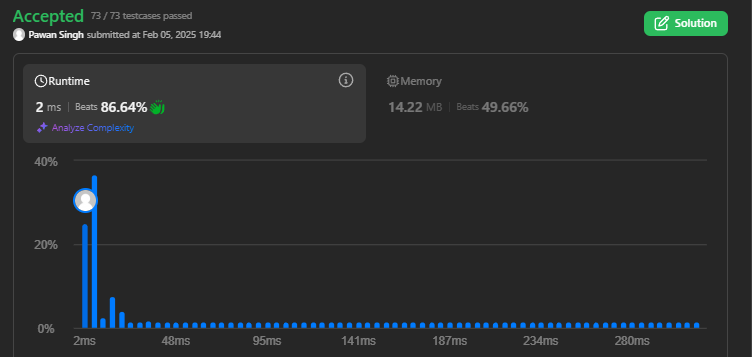
}

}

return s;

}

};



Q2 190.[Reverse Bits](https://leetcode.com/problems/reverse-bits/description/)

Class Solution {

public:

    uint32\_t reverseBits(uint32\_t n) {

        uint32\_t result = 0;

        for (int i = 0; i < 32; i++) {

            result = (result << 1) | (n & 1);

            n >>= 1;

        }

        return result;

    }

};



Q3 191.[Number of 1 Bits](https://leetcode.com/problems/number-of-1-bits/description/)

class Solution {

public:

int hammingWeight(int n) {

int res = 0;

for (int i = 0; i < 32; i++) {

if ((n >> i) & 1) {

res += 1;

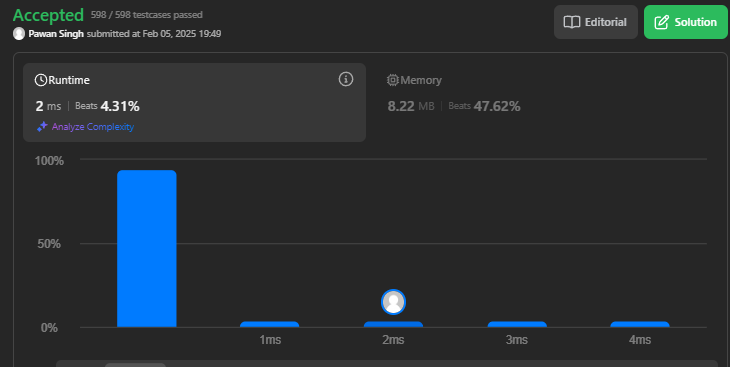
}

}

return res;

}

};



Q4 53.[Maximum Subarray](https://leetcode.com/problems/maximum-subarray/description/)

class Solution {

public:

int maxSubArray(vector<int>& nums) {

int maxSum = INT\_MIN;

int currentSum = 0;

for (int i = 0; i < nums.size(); i++) {

currentSum += nums[i];

if (currentSum > maxSum) {

maxSum = currentSum;

}

if (currentSum < 0) {

currentSum = 0;

}

}

return maxSum;

}

};



Q5 240.[Search a 2D Matrix II](https://leetcode.com/problems/search-a-2d-matrix-ii/description/)

class Solution {

public:

bool searchMatrix(vector<vector<int>>& matrix, int target) {

int n = matrix.size(), m = matrix[0].size();

int row = 0, col = m - 1;

while (row < n && col >= 0) {

if (matrix[row][col] == target) return true;

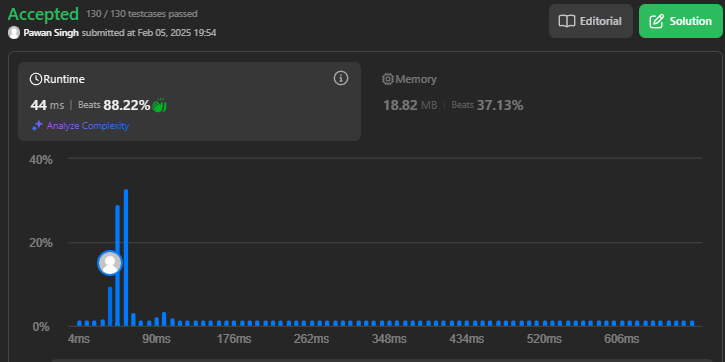
else if (matrix[row][col] < target) row++;

else col--;

}

return false;

}

}; 

Q6 372.[Super Pow](https://leetcode.com/problems/super-pow/description/)

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;

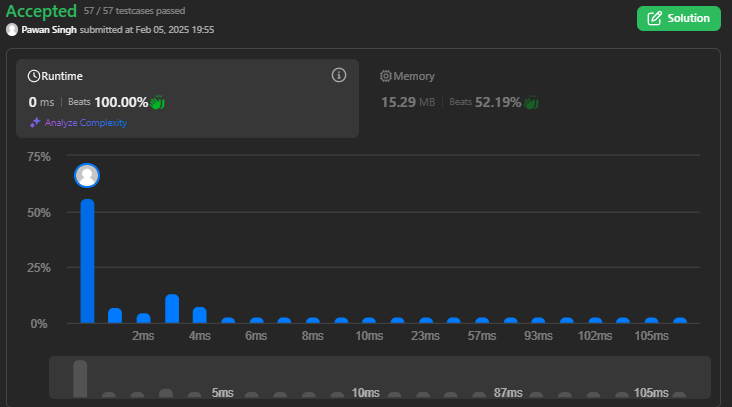
int last\_digit = b.back();

b.pop\_back();

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

}

};



Q7 932.[Beautiful Array](https://leetcode.com/problems/beautiful-array/description/)

class Solution {

public:

int partition(vector<int> &v, int start, int end, int mask)

{

int j = start;

for(int i = start; i <= end; i++)

{

if((v[i] & mask) != 0)

{

swap(v[i], v[j]);

j++;

}

}

return j;

}

void sort(vector<int> & v, int start, int end, int mask)

{

if(start >= end) return;

int mid = partition(v, start, end, mask);

sort(v, start, mid - 1, mask << 1);

sort(v, mid, end, mask << 1);

}

vector<int> beautifulArray(int N) {

vector<int> ans;

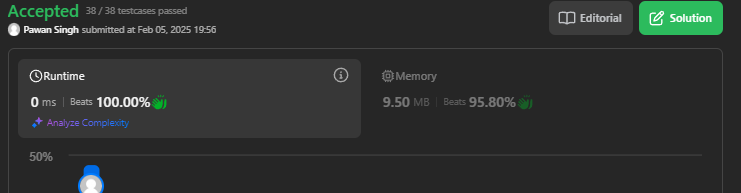
for(int i = 0; i < N; i++) ans.push\_back(i + 1);

sort(ans, 0, N - 1, 1);

return ans;

}

};



Q8 218.[The Skyline Problem](https://leetcode.com/problems/the-skyline-problem/description/)

class Solution {

public:

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

vector<vector<int>> ans;

multiset<int> pq{0};

vector<pair<int, int>> points;

for(auto b: buildings){

points.push\_back({b[0], -b[2]});

points.push\_back({b[1], b[2]});

}

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

int ongoingHeight = 0;

for(int i = 0; i < points.size(); i++){

int currentPoint = points[i].first;

int heightAtCurrentPoint = points[i].second;

if(heightAtCurrentPoint < 0){

pq.insert(-heightAtCurrentPoint);

} else {

pq.erase(pq.find(heightAtCurrentPoint));

}

auto pqTop = \*pq.rbegin();

if(ongoingHeight != pqTop){

ongoingHeight = pqTop;

ans.push\_back({currentPoint, ongoingHeight});

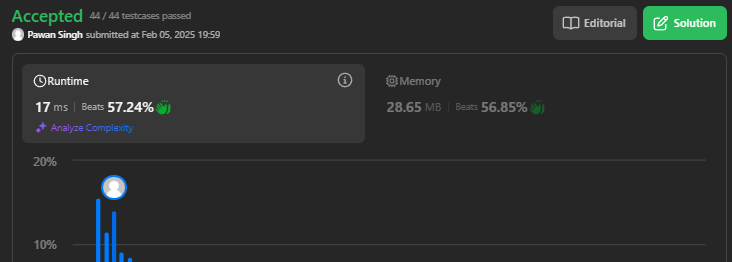
}

}

return ans;

}

};



Q9 493.[Reverse Pairs](https://leetcode.com/problems/reverse-pairs/description/)

class SegTree {

private:

    int tree\_size;

    vector<int> tree;

    void update(int lx, int rx, int ni, int idx) {

        if (rx - lx == 1) {

            tree[ni]++;

            return;

        }

        int m = (lx + rx) / 2;

        if (idx < m)

            update(lx, m, ni \* 2 + 1, idx);

        else

            update(m, rx, ni \* 2 + 2, idx);

        tree[ni] = tree[ni \* 2 + 1] + tree[ni \* 2 + 2];

    }

    int query(int l, int r, int lx, int rx, int ni) {

        if (l >= rx || r <= lx)

            return 0;

        if (l <= lx && r >= rx)

            return tree[ni];

        int m = (lx + rx) / 2;

        return query(l, r, lx, m, ni \* 2 + 1) + query(l, r, m, rx, ni \* 2 + 2);

    }

public:

    SegTree(int n) {

        tree\_size = 1;

        while (tree\_size < n)

            tree\_size <<= 1;

        tree.assign(tree\_size \* 2, 0);

    }

    void update(int idx) {

        update(0, tree\_size, 0, idx);

    }

    int query(int l, int r) {

        return query(l, r, 0, tree\_size, 0);

    }

};

class Solution {

public:

    int reversePairs(vector<int>& nums) {

        int n = nums.size();

        set<long long> values;

        for (const auto& num : nums) {

            values.insert(num);

            values.insert(2LL \* num);

        }

        int last\_index = 0;

        unordered\_map<long long, int> values\_indices;

        for (const auto& val : values)

            values\_indices[val] = last\_index++;

        SegTree seg\_tree(last\_index);

        int ans = 0;

        for (int i = 0; i < n; ++i) {

            ans += seg\_tree.query(values\_indices[2LL \* nums[i]] + 1, last\_index);

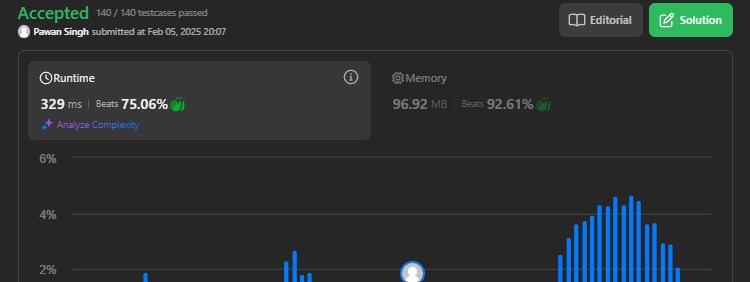
            seg\_tree.update(values\_indices[nums[i]]);

        }

        return ans;

    }

};



Q10 2407.[Longest Increasing Subsequence II](https://leetcode.com/problems/longest-increasing-subsequence-ii/description/)

class MaxSegmentTree {

public:

int n;

vector<int> tree;

MaxSegmentTree(int n\_) : n(n\_) {

int size = (int)(ceil(log2(n)));

size = (2 \* pow(2, size)) - 1;

tree = vector<int>(size);

}

int max\_value() { return tree[0]; }

int query(int l, int r) { return query\_util(0, l, r, 0, n - 1); }

int query\_util(int i, int qL, int qR, int l, int r) {

if (l >= qL && r <= qR) return tree[i];

if (l > qR || r < qL) return INT\_MIN;

int m = (l + r) / 2;

return max(query\_util(2 \* i + 1, qL, qR, l, m), query\_util(2 \* i + 2, qL, qR, m + 1, r));

}

void update(int i, int val) { update\_util(0, 0, n - 1, i, val); }

void update\_util(int i, int l, int r, int pos, int val) {

if (pos < l || pos > r) return;

if (l == r) {

tree[i] = max(val, tree[i]);

return;

}

int m = (l + r) / 2;

update\_util(2 \* i + 1, l, m, pos, val);

update\_util(2 \* i + 2, m + 1, r, pos, val);

tree[i] = max(tree[2 \* i + 1], tree[2 \* i + 2]);

}

};

class Solution {

public:

int lengthOfLIS(vector<int>& nums, int k) {

MaxSegmentTree tree(1e5 + 1);

for (int i : nums) {

int lower = max(0, i - k);

int cur = 1 + tree.query(lower, i - 1);

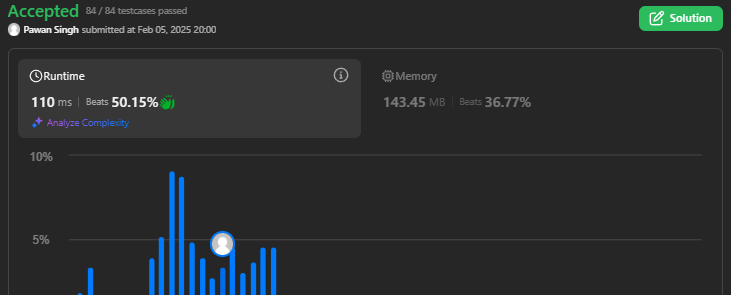
tree.update(i, cur);

}

return tree.max\_value();

}

};



Q11 88.[Merge Sorted Array](https://leetcode.com/problems/merge-sorted-array/description/)

class Solution {

public:

void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {

int i = m - 1;

int j = n - 1;

int k = m + n - 1;

while (j >= 0) {

if (i >= 0 && nums1[i] > nums2[j]) {

nums1[k--] = nums1[i--];

} else {

nums1[k--] = nums2[j--];

}

}

}

};

