# NAME- Abir Chakraborty| UID- 22BCS14321 | SECTION- 601/A 1 CONVERT SORTED ARRAY TO BINARY SEARCH TREE

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;

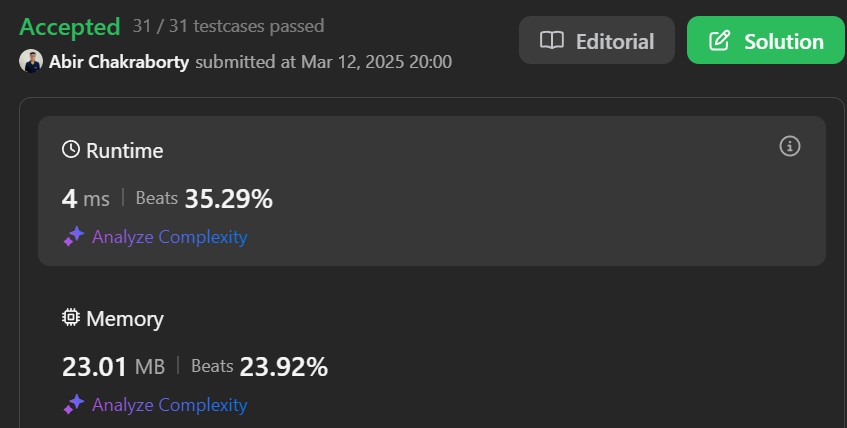
int mid = left + (right - left) / 2;

TreeNode\* root = new TreeNode(nums[mid]); root->left = helper(nums, left, mid - 1);

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

}

};



# NUMBER OF 1 BITS

class Solution { public:

int hammingWeight(uint32\_t n) { int res = 0;

for (int i = 0; i < 32; i++) { if ((n >> i) & 1) {

res += 1;

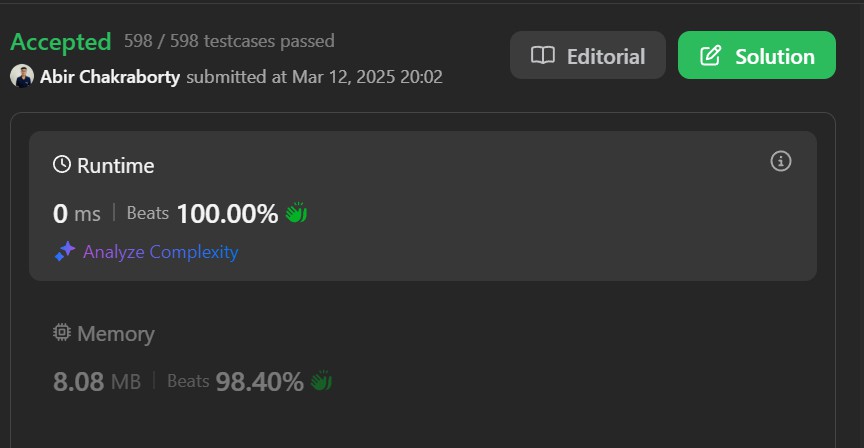
}

}

return res;

}

};



# SORT AN ARRAY

class Solution { public:

vector<int> sortArray(vector<int>& nums) { vector<int> counting(2 \* 50000 + 1, 0); for (int num : nums) {

counting[num + 50000]++;

}

int write\_ind = 0;

for (int number\_ind = 0; number\_ind < counting.size(); ++number\_ind) { int freq = counting[number\_ind];

while (freq != 0) {

nums[write\_ind] = number\_ind - 50000; write\_ind++;

freq--;

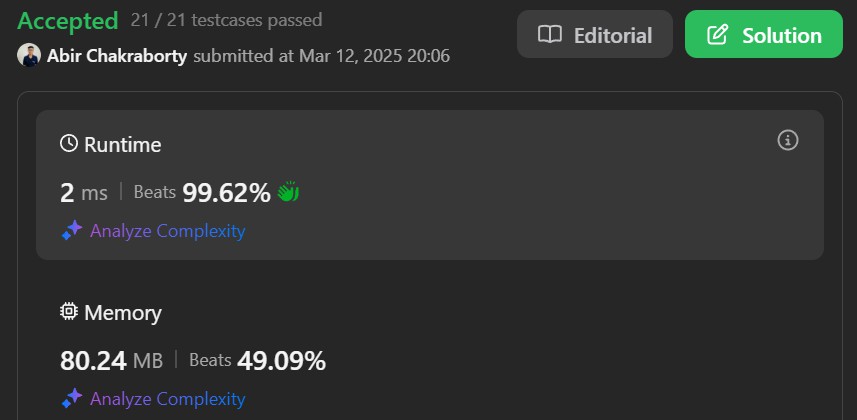
}

}

return nums;

}

};



# MAXIMUM SUBARRAY

class Solution { public:

int maxSubArray(vector<int>& arr) { int n = arr.size();

long long maxi = LONG\_MIN; long long sum = 0;

for (int i = 0; i < n; i++) { sum += arr[i];

if (sum > maxi) { maxi = sum;

}

if (sum < 0) { sum = 0;

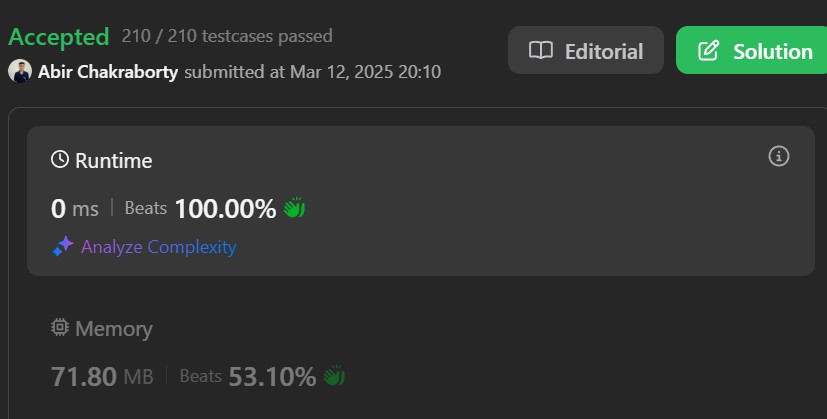
}

}

return maxi;

}

};



# BEAUTIFUL ARRAY

class Solution { public:

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);

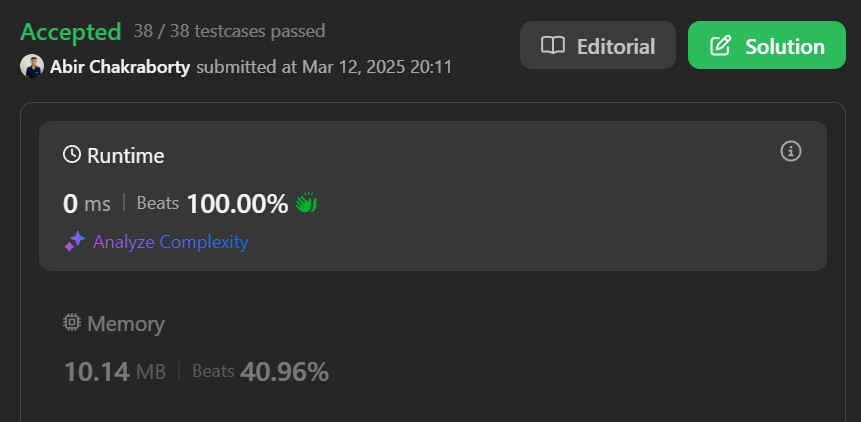
res = tmp;

}

return res;

}

};



## SUPER POW

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;

}

};



## SKYLINE PROBLEM

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;

}

};

