**Advanced Programming**

**ASSIGNMENT 2**

1. **1763.Longest Nice Substring**

**Code :**

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;

    }

};

**Output:**

****

1. **190.Reverse Bits**

**Code :**

class Solution {

public:

    uint32\_t reverseBits(uint32\_t n) {

        n = ((n & 0xffff0000) >> 16) | ((n & 0x0000ffff) << 16);

        n = ((n & 0xff00ff00) >> 8) | ((n & 0x00ff00ff) << 8);

        n = ((n & 0xf0f0f0f0) >> 4) | ((n & 0x0f0f0f0f) << 4);

        n = ((n & 0xcccccccc) >> 2) | ((n & 0x33333333) << 2);

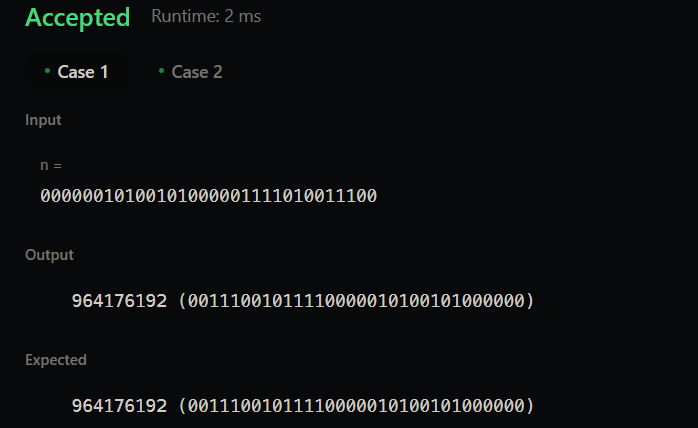
      n = ((n & 0xaaaaaaaa) >> 1) | ((n & 0x55555555) << 1);

        return n;

    }

};

**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. **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);

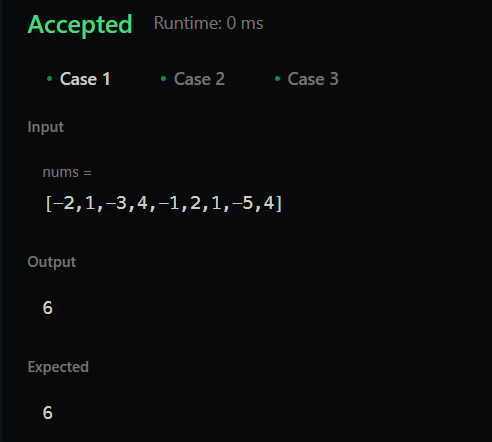
        }

        return res;

    }

};

**Output:**



1. **240.Search a 2D Matrix II**

**Code :**

class Solution {

public:

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

        int cols = matrix[0].size() - 1;

        int n = matrix.size() - 1;

        int rows = 0;

        while(rows <= n && cols >= 0){

            int toCompare = matrix[rows][cols];

            if(toCompare > target){

                cols--;

            }else if(toCompare < target){

                rows++;

            }else{

                return true;

            }

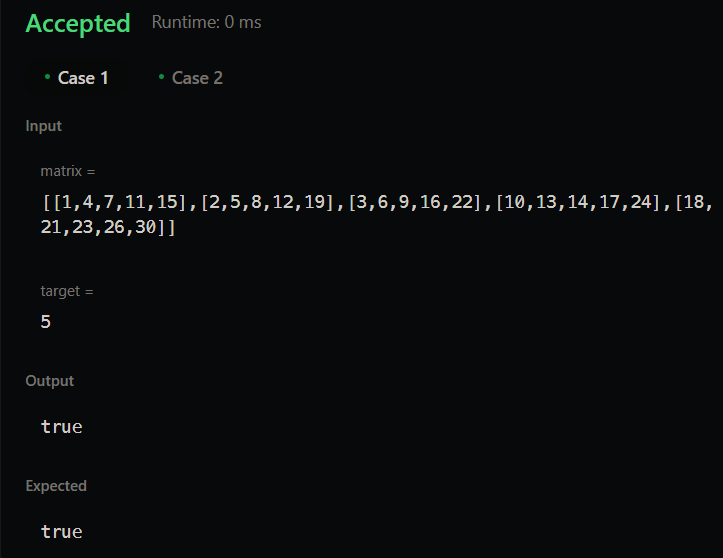
        }

        return false;

    }

};

**Output:**

****

1. **372.Super Pow**

**Code :**

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();

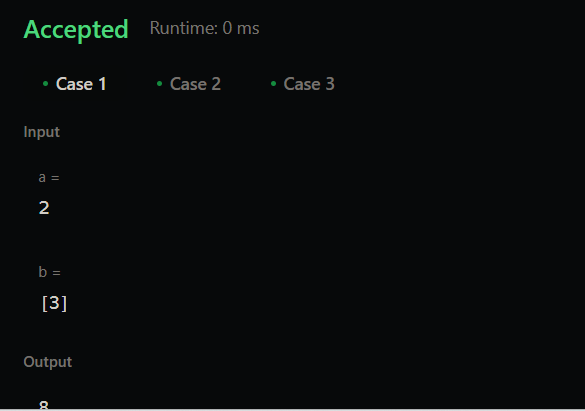
        b.pop\_back();

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

    }

};

**Output:**

****

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. **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);

        }

        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});

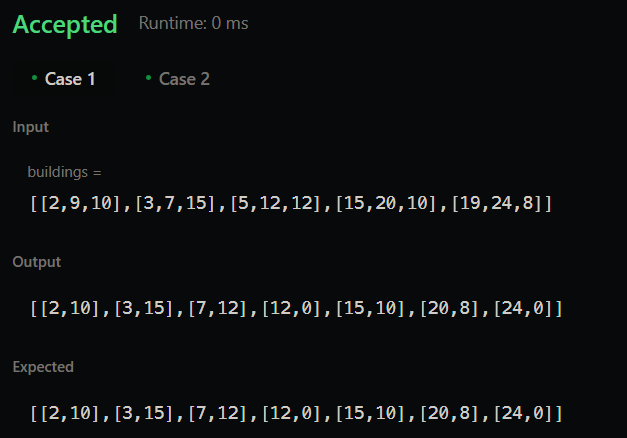
        }

        return skyline;

    }

};

**Output:**

****

1. **493.Reverse Pairs**

**Code :**

class Solution { //OPTIMAL APPROACH,,,T-O(NLOGN \* 2), S-O(N)

public:

    void merge(vector<int>&arr, int low, int mid, int high){

        int left = low, right = mid+1; vector<int>temp;

        while(left <= mid && right <= high){

            if(arr[left] <= arr[right]){

                temp.push\_back(arr[left++]);

            }

            else{

                temp.push\_back(arr[right++]);

            }

        }

        //if left array left

        while(left <= mid){

            temp.push\_back(arr[left++]);

        }

        while(right <= high){

            temp.push\_back(arr[right++]);

        }

        //now pushing back the temp array elmenets to original array

        for(int i=low; i<=high;i++){

            arr[i] = temp[i - low];

        }

    }

    int countPairs(vector<int>&arr, int low, int mid, int high){

        int count=0, right=mid+1;

        for(int i = low;i <= mid; i++){

            while (right <= high && (long long)arr[i] > 2LL \* arr[right]) right++;

            count += (right - (mid+1));

        }

        return count;

    }

    int mergeSort(vector<int>&arr, int low, int high){

        int count=0;

        if(low >= high) return count;

        int mid = (low + high)/2;

        count += mergeSort(arr, low, mid);

        count += mergeSort(arr, mid+1, high);

        count += countPairs(arr, low, mid, high);

        merge(arr, low, mid, high);

        return count;

    }

    int reversePairs(vector<int>& nums) {

        int n = nums.size();

        return mergeSort(nums, 0, n-1);

    }

};

**Output:**

****

1. **2407.Longest Increasing Subsequence II**

**Code :**

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);

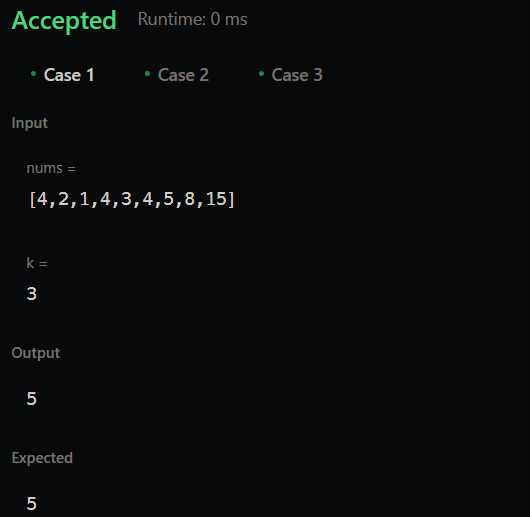
    }

    return tree.max\_value();

  }

};

**Output:**

****

1. **88.Merge Sorted Array**

**Code :**

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(i>=0 && j>=0){

            if(nums1[i]>nums2[j]){

                nums1[k]=nums1[i];

                k--;

                i--;

            }else{

                nums1[k]=nums2[j];

                k--;

                j--;

            }

        }

        while(j>=0){

            nums1[k]=nums2[j];

            k--;

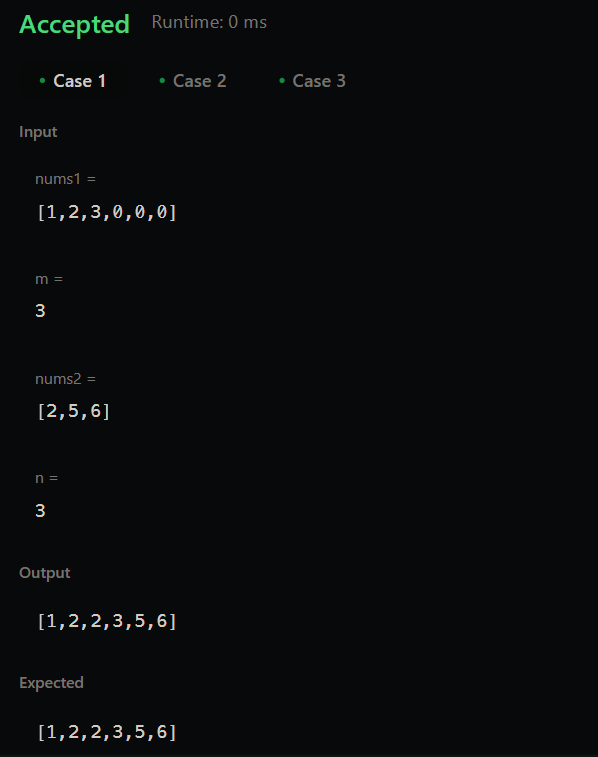
            j--;

        }

    }

};

**Output:**

****

1. **278.First Bad Version**

**Code :**

// The API isBadVersion is defined for you.

 bool isBadVersion(int version);

class Solution {

public:

    int firstBadVersion(int n) {

        int low=1;

        int high=n;

        while(low<=high){

            int mid=low+(high-low)/2;

            int version=isBadVersion(mid);

            if(version==true){

                high=mid-1;

            }

            else{

                low=mid+1;

            }

        }

        return low;

    }

};

**Output:**

****

1. **75.Sort Colors**

**Code :**

class Solution {

public:

    void sortColors(vector<int>& nums) {

        int l=nums.size();

        int count0=0;

        int count1=0;

        int count2=0;

        for(int i=0;i<l;i++){

            if(nums[i]==0) count0++;

            else if(nums[i]==1) count1++;

            else count2++;

        }

        count1=count0+count1;

        count2=count0+count1+count2;

        for(int i=0;i<l;i++){

            if(i<count0) nums[i]=0;

            else if(i<count1) nums[i]=1;

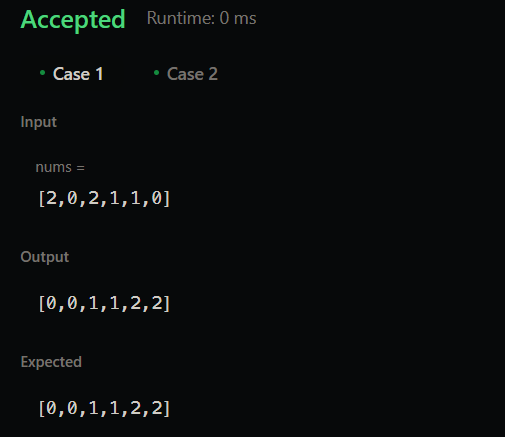
            else nums[i]=2;

        }

    }

}

**Output:**

****

1. **347.Top K Frequent Elements**

**Code :**

class Solution {

public:

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

        int n = nums.size();

        unordered\_map<int, int> map;

        vector<int> ans;

        for (int &x : nums) map[x]++;

        vector<vector<int>> arr(n + 1);

        for (auto [a, b] : map) arr[b].push\_back(a);

        for (int i = n; i > 0; i--) {

            for (int &x : arr[i]) {

                if (ans.size() == k) return ans;

                ans.push\_back(x);

            }

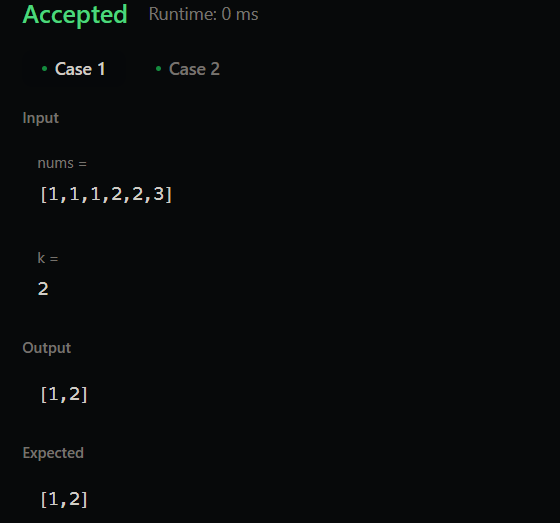
        }

        return ans;

    }

};

**Output:**

****

1. **215.Kth Largest Element in an Array**

**Code :**

class Solution {

public:

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

        nth\_element(nums.begin(), nums.begin() + k - 1, nums.end(), greater<int>());

        return nums[k - 1];

    }

};

**Output:**

