# Assignment 2

## Student Name: Anuj Kumar UID:22BCS11581

## Branch: BE-CSE (General) Section/Group: FL\_IOT-602 A

## Semester:6th Date of Performance: 04-02-25

## Subject Name: Advanced Programming Lab-2 Subject Code: 22CSP-351

# Aim: 1763. Longest Nice Substring

# Implementation/ Code:

# class Solution {

# public:

# string longestNiceSubstring(string s) {

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

# unordered\_set<char> charSet(s.begin(), s.end());

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

# if(charSet.count(tolower(s[i]))&&charSet.count(toupper(s[i]))){

# continue;

# }

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

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

# return left.size() >= right.size() ? left:right;

# }

# return s;

# }

# };

# Output:

# 

# Aim: 190. Reverse Bits

# Implementation/ Code:

# class Solution {

# public:

# uint32\_t reverseBits(uint32\_t n) {

# uint32\_t res=0;

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

# int last=n&1;

# int mask=last<<(31-i);

# res= res|mask;

# n=n>>1;

# }

# return res;

# }

# };

# Output:

# 

# Aim: 191. Number of 1 Bits

# Implementation/ Code:

# class Solution {

# public:

# int hammingWeight(int n) {

# int count=0;

# while(n){

# count+=(n&1);

# n>>=1;

# }

# return count;

# }

# };

# Output:

# 

# Aim: 53. Maximum Subarray

# Implementation/ Code:

# class Solution {

# public:

# int maxSubArray(vector<int>& nums) {

# int maxSum = INT\_MIN;

# int currentSum = 0;

# for (int num : nums) {

# currentSum += num;

# maxSum = max(maxSum, currentSum);

# if (currentSum < 0) currentSum = 0;

# }

# return maxSum;

# }

# };

# Output:

# 

# Aim: 240. Search a 2D Matrix II

# Implementation/ Code:

# class Solution {

# public:

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

# int row = 0, col = matrix[0].size() - 1;

# while (row < matrix.size() && col >= 0) {

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

# matrix[row][col] > target ? col-- : row++;

# }

# return false;

# }

# };

# Output:

# 

# Aim: 372. Super Pow

# Implementation/ Code:

# class Solution {

# public:

# int modPow(int a, int b, int mod) {

# int res = 1;

# a %= mod;

# while (b) {

# if (b % 2) res = (1LL \* res \* a) % mod;

# a = (1LL \* a \* a) % mod;

# b /= 2;

# }

# return res;

# }

# 

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

# int mod = 1337, res = 1;

# for (int d : b) res = modPow(res, 10, mod) \* modPow(a, d, mod) % mod;

# return res;

# }

# };

# Output:

# 

# Aim: 932. Beautiful Array

# Implementation/ Code:

# class Solution {

# public:

# vector<int> beautifulArray(int n) {

# vector<int> res = {1};

# while (res.size() < n) {

# vector<int> tmp;

# for (int x : res) if (2 \* x - 1 <= n) tmp.push\_back(2 \* x - 1);

# for (int x : res) if (2 \* x <= n) tmp.push\_back(2 \* x);

# res = tmp;

# }

# return res;

# }

# };

# Output:

# 

# Aim: 218. The Skyline Problem

# Implementation/ Code:

# class Solution {

# public:

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

# vector<pair<int, int>> events;

# multiset<int> heights = {0};

# vector<vector<int>> res;

# for (auto &b : buildings) {

# events.emplace\_back(b[0], -b[2]);

# events.emplace\_back(b[1], b[2]);

# }

# sort(events.begin(), events.end());

# int prevHeight = 0;

# for (auto &e : events) {

# if (e.second < 0) heights.insert(-e.second);

# else heights.erase(heights.find(e.second));

# 

# int curHeight = \*heights.rbegin();

# if (curHeight != prevHeight) {

# res.push\_back({e.first, curHeight});

# prevHeight = curHeight;

# }

# }

# return res;

# }

# };

# Output:

# 

# Aim: 493. Reverse Pairs

# Implementation/ Code:

# class Solution {

# public:

# int merge(vector<int>& nums, int left, int mid, int right) {

# int count = 0, j = mid + 1;

# for (int i = left; i <= mid; i++) {

# while (j <= right && nums[i] > 2LL \* nums[j]) j++;

# count += j - (mid + 1);

# }

# 

# inplace\_merge(nums.begin() + left, nums.begin() + mid + 1, nums.begin() + right + 1);

# return count;

# }

# int mergeSort(vector<int>& nums, int left, int right) {

# if (left >= right) return 0;

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

# int count = mergeSort(nums, left, mid) + mergeSort(nums, mid + 1, right);

# return count + merge(nums, left, mid, right);

# }

# int reversePairs(vector<int>& nums) {

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

# }

# };

# Output:

# 

# Aim: 2407. Longest Increasing Subsequence II

# Implementation/ Code:

# class SegmentTree {

# public:

# vector<int> tree;

# int size;

# SegmentTree(int n) {

# size = n;

# tree.resize(4 \* n, 0);

# }

# void update(int index, int value, int node = 1, int start = 0, int end = 100000) {

# if (start == end) {

# tree[node] = value;

# return;

# }

# int mid = (start + end) / 2;

# if (index <= mid) update(index, value, 2 \* node, start, mid);

# else update(index, value, 2 \* node + 1, mid + 1, end);

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

# }

# int query(int left, int right, int node = 1, int start = 0, int end = 100000) {

# if (left > end || right < start) return 0;

# if (left <= start && end <= right) return tree[node];

# int mid = (start + end) / 2;

# return max(query(left, right, 2 \* node, start, mid), query(left, right, 2 \* node + 1, mid + 1, end));

# }

# };

# class Solution {

# public:

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

# SegmentTree segTree(100001);

# int res = 1;

# for (int num : nums) {

# int best = segTree.query(max(0, num - k), num - 1);

# segTree.update(num, best + 1);

# res = max(res, best + 1);

# }

# return res;

# }

# };

# Output:

# 

# Aim: 88. Merge Sorted Array

# Implementation/ Code:

# class Solution {

# public:

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

# int i = m - 1, j = n - 1, k = m + n - 1;

# while (j >= 0) {

# nums1[k--] = (i >= 0 && nums1[i] > nums2[j]) ? nums1[i--] : nums2[j--];

# }

# }

# };

# Output:

# 

# Aim: 278. First Bad Version

# Implementation/ Code:

# class Solution {

# public:

# int firstBadVersion(int n) {

# int left = 1, right = n;

# while (left < right) {

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

# if (isBadVersion(mid)) right = mid;

# else left = mid + 1;

# }

# return left;

# }

# };

# Output:

# 

# Aim: 75. Sort Colors

# Implementation/ Code:

# class Solution {

# public:

# void sortColors(vector<int>& nums) {

# int low = 0, mid = 0, high = nums.size() - 1;

# while (mid <= high) {

# if (nums[mid] == 0) swap(nums[low++], nums[mid++]);

# else if (nums[mid] == 1) mid++;

# else swap(nums[mid], nums[high--]);

# }

# }

# };

# Output:

# 

# Aim: 347. Top K Frequent Elements

# Implementation/ Code:

# class Solution {

# public:

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

# unordered\_map<int, int> freq;

# for (int num : nums) freq[num]++;

# 

# priority\_queue<pair<int, int>> pq;

# for (auto [num, count] : freq) pq.emplace(count, num);

# 

# vector<int> res;

# while (k--) {

# res.push\_back(pq.top().second);

# pq.pop();

# }

# return res;

# }

# };

# Output:

# 

# Aim: 215. Kth Largest Element in an Array

# Implementation/ Code:

# class Solution {

# public:

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

# priority\_queue<int, vector<int>, greater<int>> pq;

# for (int num : nums) {

# pq.push(num);

# if (pq.size() > k) pq.pop();

# }

# return pq.top();

# }

# };

# Output:

# 

# Aim: 162. Find Peak Element

# Implementation/ Code:

# class Solution {

# public:

# int findPeakElement(vector<int>& nums) {

# int left = 0, right = nums.size() - 1;

# while (left < right) {

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

# if (nums[mid] > nums[mid + 1]) right = mid;

# else left = mid + 1;

# }

# return left;

# }

# };

# Output:

# 

# Aim: 56. Merge Intervals

# Implementation/ Code:

# class Solution {

# public:

# vector<vector<int>> merge(vector<vector<int>>& intervals) {

# sort(intervals.begin(), intervals.end());

# vector<vector<int>> res;

# for (auto &interval : intervals) {

# if (res.empty() || res.back()[1] < interval[0]) res.push\_back(interval);

# else res.back()[1] = max(res.back()[1], interval[1]);

# }

# return res;

# }

# };

# Output:

# 

# Aim: 33.Search in Rotated Sorted Array

# Implementation/ Code:

# class Solution {

# public:

# int search(vector<int>& nums, int target) {

# int left = 0, right = nums.size() - 1;

# while (left <= right) {

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

# if (nums[mid] == target) return mid;

# 

# if (nums[left] <= nums[mid]) {

# if (nums[left] <= target && target < nums[mid]) right = mid - 1;

# else left = mid + 1;

# } else {

# if (nums[mid] < target && target <= nums[right]) left = mid + 1;

# else right = mid - 1;

# }

# }

# return -1;

# }

# };

# Output:

# 

# Aim: 324.Wiggle Sort II

# Implementation/ Code:

# class Solution {

# public:

# void wiggleSort(vector<int>& nums) {

# vector<int> sorted = nums;

# sort(sorted.begin(), sorted.end());

# int n = nums.size(), mid = (n - 1) / 2, end = n - 1;

# 

# for (int i = 0; i < n; i++) {

# nums[i] = (i % 2 == 0) ? sorted[mid--] : sorted[end--];

# }

# }

# };

# Output:

# 

# Aim: 378.Kth Smallest Element in a Sorted Matrix

# Implementation/ Code:

# class Solution {

# public:

# int kthSmallest(vector<vector<int>>& matrix, int k) {

# priority\_queue<int> pq;

# for (auto &row : matrix) {

# for (int num : row) {

# pq.push(num);

# if (pq.size() > k) pq.pop();

# }

# }

# return pq.top();

# }

# };

# Output:

# 

# Aim: 4.Median of Two Sorted Arrays

# Implementation/ Code:

# class Solution {

# public:

# double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {

# if (nums1.size() > nums2.size()) return findMedianSortedArrays(nums2, nums1);

# 

# int x = nums1.size(), y = nums2.size();

# int low = 0, high = x;

# 

# while (low <= high) {

# int partitionX = (low + high) / 2;

# int partitionY = (x + y + 1) / 2 - partitionX;

# 

# int maxX = (partitionX == 0) ? INT\_MIN : nums1[partitionX - 1];

# int minX = (partitionX == x) ? INT\_MAX : nums1[partitionX];

# int maxY = (partitionY == 0) ? INT\_MIN : nums2[partitionY - 1];

# int minY = (partitionY == y) ? INT\_MAX : nums2[partitionY];

# 

# if (maxX <= minY && maxY <= minX) {

# if ((x + y) % 2 == 0) return (max(maxX, maxY) + min(minX, minY)) / 2.0;

# else return max(maxX, maxY);

# } else if (maxX > minY) {

# high = partitionX - 1;

# } else {

# low = partitionX + 1;

# }

# }

# return -1;

# }

# };

# Output:

# 