# Assignment 6

## Student Name: Aryan Mehta UID:22BCS12209

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

## Semester:6th Date of Performance: 04-03-25

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

# Aim: 108. Convert Sorted Array to Binary Search Tree

# Implementation/ Code:

# class Solution {

# public:

# TreeNode\* buildBST(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 = buildBST(nums, left, mid - 1);

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

# return root;

# }

# TreeNode\* sortedArrayToBST(vector<int>& nums) {

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

# }

# void inorderTraversal(TreeNode\* root) {

# if (!root) return;

# inorderTraversal(root->left);

# cout << root->val << " ";

# inorderTraversal(root->right);

# }

# };

# Output:

# 

# Aim: 191. Number of 1 Bits

# Implementation/ Code:

# class Solution {

# public:

# int hammingWeight(int n) {

# int count = 0;

# while (n) {

# n &= (n - 1); // Removes the rightmost set bit

# count++;

# }

# return count;

# }};

# Output:

# 

# Aim: 912. Sort an Array

# Implementation/ Code:

# class Solution {

# public:

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

# int n1 = mid - left + 1;

# int n2 = right - mid;

# vector<int> leftArr(n1), rightArr(n2);

# for (int i = 0; i < n1; i++)

# leftArr[i] = nums[left + i];

# for (int i = 0; i < n2; i++)

# rightArr[i] = nums[mid + 1 + i];

# int i = 0, j = 0, k = left;

# while (i < n1 && j < n2) {

# if (leftArr[i] <= rightArr[j]) {

# nums[k] = leftArr[i];

# i++;

# } else {

# nums[k] = rightArr[j];

# j++;

# }

# k++;

# }

# 

# while (i < n1) {

# nums[k] = leftArr[i];

# i++;

# k++;

# }

# while (j < n2) {

# nums[k] = rightArr[j];

# j++;

# k++;

# }

# }

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

# if (left < right) {

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

# mergeSort(nums, left, mid); // Sort first half

# mergeSort(nums, mid + 1, right); // Sort second half

# merge(nums, left, mid, right); // Merge sorted halves

# }

# }

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

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

# return nums;

# }

# };

# Output:

# 

# Aim: 53. Maximum Subarray

# Implementation/ Code:

# class Solution {

# public:

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

# int maxSum = INT\_MIN, currentSum = 0;

# for (int num : nums) {

# currentSum = max(num, currentSum + num);

# maxSum = max(maxSum, currentSum);

# }

# return maxSum;

# }

# };

# Output:

# 

# Aim: 932. Beautiful Array

# Implementation/ Code:

# class Solution {

# public:

# vector<int> beautifulArray(int n) {

# if (n == 1) return {1}; // Base case

# vector<int> odd, even, result;

# for (int num : beautifulArray((n + 1) / 2))

# odd.push\_back(2 \* num - 1);

# for (int num : beautifulArray(n / 2))

# even.push\_back(2 \* num);

# result.insert(result.end(), odd.begin(), odd.end());

# result.insert(result.end(), even.begin(), even.end());

# return result;

# }

# };

# Output:

# 

# Aim: 372. Super Pow

# Implementation/ Code:

class Solution {

public:

const int MOD = 1337;

int modPower(int a, int b) {

int result = 1;

a %= MOD;

while (b > 0) {

if (b % 2 == 1) result = (result \* a) % MOD;

a = (a \* a) % MOD;

b /= 2;

}

return result;

}

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

if (b.empty()) return 1;

int lastDigit = b.back();

b.pop\_back();

int part1 = modPower(superPow(a, b), 10) % MOD; // Recursive part

int part2 = modPower(a, lastDigit) % MOD; // Direct part

return (part1 \* part2) % MOD;

}

# };

# Output:

# 

# Aim: 218. The Skyline Problem

# Implementation/ Code:

# class Solution {

# public:

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

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

# 

# for (auto& b : buildings) {

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

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

# }

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

# multiset<int> heights = {0};

# vector<vector<int>> result;

# int prevMax = 0;

# for (auto& e : events) {

# int x = e.first, h = e.second;

# if (h < 0) {

# heights.insert(-h);

# } else {

# heights.erase(heights.find(h));

# }

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

# if (currMax != prevMax) {

# result.push\_back({x, currMax});

# prevMax = currMax;

# }

# }

# return result;

# }

# };Output:

# 