**5 Marks Coding Question**

**Problem Statement**

**You are given an array of strings representing a binary tree. Each string in the array corresponds to the value of a node in the binary tree. The position of each string in the array represents the level order traversal of the binary tree. If a node at a particular position in the array is null, it indicates that the corresponding node in the binary tree is null.**

**Write a function that takes the array of strings as input and returns the preorder traversal of the nodes' values in the binary tree.**

**Input Format:**

**4**

[“1”,”null”,”2”,”3”]

**Output Format:**

**1 2 3**

**Code Constraints:**

**1<=N<=10**

**Solution: (in required language)**

**#include <iostream>**

**#include <vector>**

**#include <queue>**

**// Definition for a binary tree node.**

**struct TreeNode {**

**int val;**

**TreeNode \*left;**

**TreeNode \*right;**

**TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}**

**};**

**// Function to convert array to binary tree**

**TreeNode\* arrayToTree(const std::vector<int>& arr) {**

**if (arr.empty()) {**

**return nullptr;**

**}**

**TreeNode\* root = new TreeNode(arr[0]);**

**std::queue<TreeNode\*> q;**

**q.push(root);**

**for (int i = 1; i < arr.size(); i += 2) {**

**TreeNode\* current = q.front();**

**q.pop();**

**if (arr[i] != NULL) {**

**current->left = new TreeNode(arr[i]);**

**q.push(current->left);**

**}**

**if (i + 1 < arr.size() && arr[i + 1] != NULL) {**

**current->right = new TreeNode(arr[i + 1]);**

**q.push(current->right);**

**}**

**}**

**return root;**

**}**

**// Function to perform a preorder traversal of the binary tree**

**void preorderTraversal(TreeNode\* root) {**

**if (root == nullptr) {**

**return;**

**}**

**std::cout << root->val << " ";**

**preorderTraversal(root->left);**

**preorderTraversal(root->right);**

**}**

**int main() {**

**// Input the array representation from the user**

**std::cout << "Enter the size of the array: ";**

**int size;**

**std::cin >> size;**

**std::vector<int> arr(size);**

**std::cout << "Enter the array elements (use 'null' for NULL): ";**

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

**std::string element;**

**std::cin >> element;**

**if (element == "null") {**

**arr[i] = NULL;**

**} else {**

**arr[i] = std::stoi(element);**

**}**

**}**

**// Convert array to binary tree**

**TreeNode\* root = arrayToTree(arr);**

**// Perform preorder traversal to verify the tree structure**

**std::cout << "Preorder Traversal: ";**

**preorderTraversal(root);**

**// Clean up allocated memory**

**// You can implement a function to delete the tree nodes**

**return 0;**

**}**

**Sample Testcase 1**

Input: 4

1 null 2 3

Output: 1 2 3

**Sample Testcase 2**

Input:1

1

Output: 1

**Hidden Testcase 1 - (Easy) Weightage 10%**

Input:4

1 2 2 3

Output: 1 2 3 2

**Hidden Testcase 2 - (Easy) Weightage 10%**

Input:7

1 2 2 3 4 4 5

Output: 1 2 3 4 2 4 5

**Hidden Testcase 3 - (Medium) Weightage 15%**

Input:4

1 null 2 3

Output: 1 2 3

**Hidden Testcase 4 - (Medium) Weightage 15%**

Input:7

1 2 2 3 null 9 5

Output: 1 2 3 2 9 5

**Hidden Testcase 5 - (Hard) Weightage 25%**

Input:7

1 2 2 3 null null 5

Output: 1 2 3 2 5

**Hidden Testcase 6 - (Hard) Weightage 25%**

Input: 7

1 2 2 10 null null 5

Output: 1 2 10 2 5

**5 Marks Coding Question**

**Problem Statement**

**You are given an array of strings representing a binary tree. Each string in the array corresponds to the value of a node in the binary tree. The position of each string in the array represents the level order traversal of the binary tree. If a node at a particular position in the array is null, it indicates that the corresponding node in the binary tree is null.**

**Write a function that takes the array of strings as input and returns the postorder traversal of the nodes' values in the binary tree.Input Format:**

**4**

[“1”,”null”,”2”,”3”]

**Output Format:**

3 2 1

**Code Constraints:**

**1<=N<=10**

**Solution: (in required language)**

**#include <iostream>**

**#include <vector>**

**#include <queue>**

**// Definition for a binary tree node.**

**struct TreeNode {**

**int val;**

**TreeNode \*left;**

**TreeNode \*right;**

**TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}**

**};**

**// Function to convert array to binary tree**

**TreeNode\* arrayToTree(const std::vector<int>& arr) {**

**if (arr.empty()) {**

**return nullptr;**

**}**

**TreeNode\* root = new TreeNode(arr[0]);**

**std::queue<TreeNode\*> q;**

**q.push(root);**

**for (int i = 1; i < arr.size(); i += 2) {**

**TreeNode\* current = q.front();**

**q.pop();**

**if (arr[i] != NULL) {**

**current->left = new TreeNode(arr[i]);**

**q.push(current->left);**

**}**

**if (i + 1 < arr.size() && arr[i + 1] != NULL) {**

**current->right = new TreeNode(arr[i + 1]);**

**q.push(current->right);**

**}**

**}**

**return root;**

**}**

**// Function to perform a postorder traversal of the binary tree**

**void postorderTraversal(TreeNode\* root) {**

**if (root == nullptr) {**

**return;**

**}**

**postorderTraversal(root->left);**

**postorderTraversal(root->right);**

**std::cout << root->val << " ";**

**}**

**int main() {**

**// Input the array representation from the user**

**std::cout << "Enter the size of the array: ";**

**int size;**

**std::cin >> size;**

**std::vector<int> arr(size);**

**std::cout << "Enter the array elements (use 'null' for NULL): ";**

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

**std::string element;**

**std::cin >> element;**

**if (element == "null") {**

**arr[i] = NULL;**

**} else {**

**arr[i] = std::stoi(element);**

**}**

**}**

**// Convert array to binary tree**

**TreeNode\* root = arrayToTree(arr);**

**// Perform postorder traversal**

**std::cout << "Postorder Traversal: ";**

**postorderTraversal(root);**

**std::cout << std::endl;**

**// Clean up allocated memory**

**// You can implement a function to delete the tree nodes**

**return 0;**

**}**

**Sample Testcase 1**

Input: 4

1 null 2 3

Output: 3 2 1

**Sample Testcase 2**

Input:1

1

Output: 1

**Hidden Testcase 1 - (Easy) Weightage 10%**

Input:4

1 null 4 3

Output: 3 4 1

**Hidden Testcase 2 - (Easy) Weightage 10%**

Input:6

1 null 4 3 null 5

Output: 5 3 4 1

**Hidden Testcase 3 - (Medium) Weightage 15%**

Input: 6

1 2 4 3 null 5

Output: 3 2 5 4 1

**Hidden Testcase 4 - (Medium) Weightage 15%**

Input: 6

1 2 4 3 5 6

Output: 3 6 2 5 4 1

**Hidden Testcase 5 - (Hard) Weightage 25%**

Input:7

1 2 4 3 6 5 7

Output: 3 6 2 5 7 4 1

**Hidden Testcase 6 - (Hard) Weightage 25%**

Input: 4

1 null 2 3

Output: 3 2 1

**5 Marks Coding Question**

**Problem Statement**

**You are given an array of strings representing a binary tree. Each string in the array corresponds to the value of a node in the binary tree. The position of each string in the array represents the level order traversal of the binary tree. If a node at a particular position in the array is null, it indicates that the corresponding node in the binary tree is null.**

**Write a function that takes the array of strings as input and returns the inorder traversal of the nodes' values in the binary tree.Input Format:**

**4**

[“1”,”null”,”2”,”3”]

**Output Format:**

1 3 2

**Code Constraints:**

**1<=N<=10**

**Solution: (in required language)**

**#include <iostream>**

**#include <vector>**

**#include <queue>**

**// Definition for a binary tree node.**

**struct TreeNode {**

**int val;**

**TreeNode \*left;**

**TreeNode \*right;**

**TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}**

**};**

**// Function to convert array to binary tree**

**TreeNode\* arrayToTree(const std::vector<int>& arr) {**

**if (arr.empty()) {**

**return nullptr;**

**}**

**TreeNode\* root = new TreeNode(arr[0]);**

**std::queue<TreeNode\*> q;**

**q.push(root);**

**for (int i = 1; i < arr.size(); i += 2) {**

**TreeNode\* current = q.front();**

**q.pop();**

**if (arr[i] != NULL) {**

**current->left = new TreeNode(arr[i]);**

**q.push(current->left);**

**}**

**if (i + 1 < arr.size() && arr[i + 1] != NULL) {**

**current->right = new TreeNode(arr[i + 1]);**

**q.push(current->right);**

**}**

**}**

**return root;**

**}**

**// Function to perform an inorder traversal of the binary tree**

**void inorderTraversal(TreeNode\* root) {**

**if (root == nullptr) {**

**return;**

**}**

**inorderTraversal(root->left);**

**std::cout << root->val << " ";**

**inorderTraversal(root->right);**

**}**

**int main() {**

**// Input the array representation from the user**

**std::cout << "Enter the size of the array: ";**

**int size;**

**std::cin >> size;**

**std::vector<int> arr(size);**

**std::cout << "Enter the array elements (use 'null' for NULL): ";**

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

**std::string element;**

**std::cin >> element;**

**if (element == "null") {**

**arr[i] = NULL;**

**} else {**

**arr[i] = std::stoi(element);**

**}**

**}**

**// Convert array to binary tree**

**TreeNode\* root = arrayToTree(arr);**

**// Perform inorder traversal**

**std::cout << "Inorder Traversal: ";**

**inorderTraversal(root);**

**std::cout << std::endl;**

**// Clean up allocated memory**

**// You can implement a function to delete the tree nodes**

**return 0;**

**}**

**Sample Testcase 1**

Input: **4**

**1 null 2 3**

Output: 1 3 2

**Sample Testcase 2**

Input:5

1 null 2 3 5

Output: 1 3 2 5

**Hidden Testcase 1 - (Easy) Weightage 10%**

Input: 6

1 null 2 3 5 7

Output: 1 7 3 2 5

**Hidden Testcase 2 - (Easy) Weightage 10%**

Input: 6

1 2 2 3 5 7

Output: 3 2 5 1 7 2

**Hidden Testcase 3 - (Medium) Weightage 15%**

Input: 6

1 2 2 3 9 7

Output: 3 2 9 1 7 2

**Hidden Testcase 4 - (Medium) Weightage 15%**

Input: 7

1 2 2 3 9 7 8

Output: 3 2 9 1 7 2 8

**Hidden Testcase 5 - (Hard) Weightage 25%**

Input:7

1 5 2 3 9 7 8

Output: 3 5 9 1 7 2 8

**Hidden Testcase 6 - (Hard) Weightage 25%**

Input: 7

1 5 2 3 null 7 8

Output: 3 5 1 7 2 8

**5 Marks Coding Question**

**Problem Statement**

**You are given an array of strings representing a binary tree. Each string in the array corresponds to the value of a node in the binary tree. The position of each string in the array represents the level order traversal of the binary tree. If a node at a particular position in the array is null, it indicates that the corresponding node in the binary tree is null.**

**Write a function that takes the array of strings as input and returns the level order traversal of the nodes' values in the binary tree.**

**Input Format:4**

1 null 2 3

**Output Format:**

1 2 3

**Code Constraints:**

**1<=N<=10**

**Solution: (in required language)**

**#include <iostream>**

**#include <vector>**

**#include <queue>**

**// Definition for a binary tree node.**

**struct TreeNode {**

**int val;**

**TreeNode \*left;**

**TreeNode \*right;**

**TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}**

**};**

**// Function to convert array to binary tree**

**TreeNode\* arrayToTree(const std::vector<int>& arr) {**

**if (arr.empty()) {**

**return nullptr;**

**}**

**TreeNode\* root = new TreeNode(arr[0]);**

**std::queue<TreeNode\*> q;**

**q.push(root);**

**for (int i = 1; i < arr.size(); i += 2) {**

**TreeNode\* current = q.front();**

**q.pop();**

**if (arr[i] != NULL) {**

**current->left = new TreeNode(arr[i]);**

**q.push(current->left);**

**}**

**if (i + 1 < arr.size() && arr[i + 1] != NULL) {**

**current->right = new TreeNode(arr[i + 1]);**

**q.push(current->right);**

**}**

**}**

**return root;**

**}**

**// Function to perform a level order traversal of the binary tree**

**void levelOrderTraversal(TreeNode\* root) {**

**if (root == nullptr) {**

**return;**

**}**

**std::queue<TreeNode\*> q;**

**q.push(root);**

**while (!q.empty()) {**

**TreeNode\* current = q.front();**

**q.pop();**

**std::cout << current->val << " ";**

**if (current->left != nullptr) {**

**q.push(current->left);**

**}**

**if (current->right != nullptr) {**

**q.push(current->right);**

**}**

**}**

**}**

**int main() {**

**// Input the array representation from the user**

**std::cout << "Enter the size of the array: ";**

**int size;**

**std::cin >> size;**

**std::vector<int> arr(size);**

**std::cout << "Enter the array elements (use 'null' for NULL): ";**

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

**std::string element;**

**std::cin >> element;**

**if (element == "null") {**

**arr[i] = NULL;**

**} else {**

**arr[i] = std::stoi(element);**

**}**

**}**

**// Convert array to binary tree**

**TreeNode\* root = arrayToTree(arr);**

**// Perform level order traversal**

**std::cout << "Level Order Traversal: ";**

**levelOrderTraversal(root);**

**std::cout << std::endl;**

**// Clean up allocated memory**

**// You can implement a function to delete the tree nodes**

**return 0;**

**}**

**Sample Testcase 1**

Input: 7

3 9 20 null null 15 7

Output:

3 9 20 15 7

**Sample Testcase 2**

Input:1

1

Output: 1

**Hidden Testcase 1 - (Easy) Weightage 10%**

Input:4

1 null 2 3

Output: 1 2 3

**Hidden Testcase 2 - (Easy) Weightage 10%**

Input:4

1 2 2 3

Output: 1 2 2 3

**Hidden Testcase 3 - (Medium) Weightage 15%**

Input: 1 2 null 3

Output: 1 2 3

**Hidden Testcase 4 - (Medium) Weightage 15%**

Input: 4

1 2 5 3

Output: 1 2 5 3

**Hidden Testcase 5 - (Hard) Weightage 25%**

Input:5

1 2 5 3 6

Output: 1 5 3 6

**Hidden Testcase 6 - (Hard) Weightage 25%**

Input: 6

1 2 5 3 6 9

Output: 1 2 5 3 6 9

**5 Marks Coding Question**

**Problem Statement**

Given an array nums of size n, print the majority element.

The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

**Input Format:**

N=3

nums = [3,2,3]

**Output Format:**

3

**Code Constraints:**

n == nums.length

1 <= n <= 5 \* 104

-109 <= nums[i] <= 109

**Solution: (in required language)**

**#include <iostream>**

**#include <vector>**

**#include <unordered\_map>**

**int findMajorityElement(const std::vector<int>& nums) {**

**std::unordered\_map<int, int> countMap;**

**int majorityElement = -1;**

**int majorityCount = 0;**

**for (int num : nums) {**

**countMap[num]++;**

**if (countMap[num] > majorityCount) {**

**majorityElement = num;**

**majorityCount = countMap[num];**

**}**

**}**

**return majorityElement;**

**}**

**int main() {**

**// Input size of the array**

**int size;**

**std::cout << "Enter the size of the array: ";**

**std::cin >> size;**

**// Input array elements**

**std::vector<int> nums(size);**

**std::cout << "Enter the elements of the array: ";**

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

**std::cin >> nums[i];**

**}**

**// Find the majority element**

**int majorityElement = findMajorityElement(nums);**

**// Print the majority element**

**std::cout << "The majority element is: " << majorityElement << std::endl;**

**return 0;**

**}**

**Sample Testcase 1**

Input: 3

3 2 3

Output: 3

**Sample Testcase 2**

Input: 3

2 3 2

Output: 2

**Hidden Testcase 1 - (Easy) Weightage 10%**

Input: 1

1

Output: 1

**Hidden Testcase 2 - (Easy) Weightage 10%**

Input: 3

1 1 2

Output: 1

**Hidden Testcase 3 - (Medium) Weightage 15%**

Input: 5

1 1 1 6 7

Output: 1

**Hidden Testcase 4 - (Medium) Weightage 15%**

Input: 5

2 2 5 6 2

Output: 2

**Hidden Testcase 5 - (Hard) Weightage 25%**

Input:3

1 2 2

Output:2

**Hidden Testcase 6 - (Hard) Weightage 25%**

Input: 5

6 6 6 8 6

Output: 6

**5 Marks Coding Question**

**Problem Statement**

Given an integer array nums, print yes if any value appears at least twice in the array, and print no if every element is distinct.

**Input Format:**

N=4

Nums =[1,2,3,1]

**Output Format:**

yes

**Code Constraints:**

1 <= nums.length <= 105

-109 <= nums[i] <= 109

**Solution: (in required language)**

#include <iostream>

#include <vector>

#include <map>

using namespace std;

string checkDuplicates(const vector<int>& nums) {

map<int, int> countMap;

for (int num : nums) {

countMap[num]++;

if (countMap[num] > 1) {

return "yes";

}

}

return "no";

}

int main() {

// Input size of the array

int size;

cout << "Enter the size of the array: ";

cin >> size;

// Input array elements

vector<int> nums(size);

cout << "Enter the elements of the array: ";

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

cin >> nums[i];

}

// Check for duplicates

string result = checkDuplicates(nums);

// Print the result

cout << result << endl;

return 0;

}

**Sample Testcase 1**

Input: 5

1 2 3 4 5

Output: no

**Sample Testcase 2**

Input:5

1 2 3 3 2

Output: yes

**Hidden Testcase 1 - (Easy) Weightage 10%**

Input: 1

1

Output: no

**Hidden Testcase 2 - (Easy) Weightage 10%**

Input: 1

2

Output: no

**Hidden Testcase 3 - (Medium) Weightage 15%**

Input: 2

2 2

Output: yes

**Hidden Testcase 4 - (Medium) Weightage 15%**

Input: 3

1 2 3

Output: no

**Hidden Testcase 5 - (Hard) Weightage 25%**

Input:2

1 2

Output: no

**Hidden Testcase 6 - (Hard) Weightage 25%**

Input: 5

1 2 3 3 5

Output: yes

**5 Marks Coding Question**

**Problem Statement**

take input of two strings from user s and t

String t is generated by random shuffling string s and then add one more letter at a random position.

print the letter that was added to t.

**Input Format:**

s = "abcd"

t = "abcde"

**Output Format:**

e

**Code Constraints:**

0 <= s.length <= 1000

t.length == s.length + 1

s and t consist of lowercase English letters.

**Solution: (in required language)**

#include <iostream>

#include <string>

#include <unordered\_map>

char findAddedLetter(const std::string& s, const std::string& t) {

std::unordered\_map<char, int> countMap;

// Count characters in string t

for (char ch : t) {

countMap[ch]++;

}

// Decrement counts based on characters in string s

for (char ch : s) {

countMap[ch]--;

}

// Find the character with non-zero count

for (const auto& entry : countMap) {

if (entry.second > 0) {

return entry.first;

}

}

return '\0'; // Return null character if no character found

}

int main() {

// Input strings s and t

std::string s, t;

std::cout << "Enter string s: ";

std::cin >> s;

std::cout << "Enter string t: ";

std::cin >> t;

// Find the added letter

char addedLetter = findAddedLetter(s, t);

// Print the added letter

if (addedLetter != '\0') {

std::cout << "The added letter in string t is: " << addedLetter << std::endl;

} else {

std::cout << "No letter was added to string t." << std::endl;

}

return 0;

}

**Sample Testcase 1**

Input: abcd

abcde

Output: e

**Sample Testcase 2**

Input :abc

abce

Output: e

**Hidden Testcase 1 - (Easy) Weightage 10%**

Input: asd

asdf

Output: f

**Hidden Testcase 2 - (Easy) Weightage 10%**

Input: asdfg

asdfgh

Output: h

**Hidden Testcase 3 - (Medium) Weightage 15%**

Input: qwerty

qwertyu

Output: u

**Hidden Testcase 4 - (Medium) Weightage 15%**

Input: zxc

zxcv

Output: v

**Hidden Testcase 5 - (Hard) Weightage 25%**

Input:ghj

ghjk

Output: k

**Hidden Testcase 6 - (Hard) Weightage 25%**

Input: vgy

vgyu

Output: u

**5 Marks Coding Question**

**Problem Statement**

take size and array input from user, for each nums[i] find out how many numbers in the array are smaller than it. That is, for each nums[i] you have to count the number of valid j's such that j != i and nums[j] < nums[i].

**print the answer.**

**Input Format:**

N=4

nums = [8,1,2,2,3]

**Output Format:**

4 0 1 1 3

**Code Constraints:**

**1<=N<=10**

**Solution: (in required language)**

**#include <iostream>**

**#include <vector>**

**#include <map>**

**using namespace std;**

**vector<int> countSmallerNumbers(const vector<int>& nums) {**

**map<int, int> countMap;**

**for (int num : nums) {**

**countMap[num]++;**

**}**

**vector<int> counts;**

**int smallerCount = 0;**

**for (int num : nums) {**

**counts.push\_back(smallerCount);**

**smallerCount += countMap[num];**

**}**

**return counts;**

**}**

**int main() {**

**// Input size of the array**

**int size;**

**cout << "Enter the size of the array: ";**

**cin >> size;**

**// Input array elements**

**vector<int> nums(size);**

**cout << "Enter the elements of the array: ";**

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

**cin >> nums[i];**

**}**

**// Count smaller numbers for each element**

**vector<int> counts = countSmallerNumbers(nums);**

**// Print the counts**

**cout << "Number of smaller elements for each element in the array: ";**

**for (int count : counts) {**

**cout << count << " ";**

**}**

**cout << endl;**

**return 0;**

**}**

**Sample Testcase 1**

Input: 5

5 2 6 1 7

Output: 2 1 3 0 4

**Sample Testcase 2**

Input:4

3 3 3 3

Output: 0 0 0 0

**Hidden Testcase 1 - (Easy) Weightage 10%**

Input: 4

1 1 1 1

Output:0 0 0 0

**Hidden Testcase 2 - (Easy) Weightage 10%**

Input: 5

1 2 3 4 5

Output: 0 1 2 3 4

**Hidden Testcase 3 - (Medium) Weightage 15%**

Input: 5

5 5 5 5 5

Output: 0 0 0 0 0

**Hidden Testcase 4 - (Medium) Weightage 15%**

Input: 5

5 4 3 2 1

Output: 4 3 2 1 0

**Hidden Testcase 5 - (Hard) Weightage 25%**

Input:3

1 1 1

Output:0 0 0

**Hidden Testcase 6 - (Hard) Weightage 25%**

Input: 5

0 0 0 0 0

Output: 0 0 0 0 0

**5 Marks Coding Question**

**Problem Statement**

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

**Input Format:**

Sixe = 4

nums = [2,7,11,15]

Target = 9

**Output Format:**

0 1 (Because nums[0] + nums[1] == 9, we return [0, 1].)

**Code Constraints:**

2 <= nums.length <= 104

-109 <= nums[i] <= 109

-109 <= target <= 109

**Solution: (in required language)**

#include <iostream>

#include <vector>

#include <unordered\_map>

using namespace std;

vector<int> twoSum(const vector<int>& nums, int target) {

unordered\_map<int, int> numMap;

vector<int> indices;

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

int complement = target - nums[i];

if (numMap.find(complement) != numMap.end()) {

indices.push\_back(numMap[complement]);

indices.push\_back(i);

return indices;

}

numMap[nums[i]] = i;

}

return indices;

}

int main() {

// Input size of the array

int size;

cout << "Enter the size of the array: ";

cin >> size;

// Input array elements

vector<int> nums(size);

cout << "Enter the elements of the array: ";

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

cin >> nums[i];

}

// Input target value

int target;

cout << "Enter the target value: ";

cin >> target;

// Find indices

vector<int> result = twoSum(nums, target);

// Print indices if found

if (!result.empty()) {

cout << "Indices of the two numbers that add up to " << target << " are: ";

cout << result[0] << " and " << result[1] << endl;

} else {

cout << "No two numbers found that add up to the target." << endl;

}

return 0;

}

**Sample Testcase 1**

Input: 4

2 7 11 15

9

Output: 0 1

**Sample Testcase 2**

Input:3

3 2 4

6

Output: 1 2

**Hidden Testcase 1 - (Easy) Weightage 10%**

Input: 2

3 3

6

Output:0 1

**Hidden Testcase 2 - (Easy) Weightage 10%**

Input:3

3 2 4

6

Output: 1 2

**Hidden Testcase 3 - (Medium) Weightage 15%**

Input: 4

2 7 11 15

6

Output: 0 1

**Hidden Testcase 4 - (Medium) Weightage 15%**

Input: 4

2 9 7 15

9

Output: 0 1

**Hidden Testcase 5 - (Hard) Weightage 25%**

Input:4

1 7 9 15

8

Output: 0 1

**Hidden Testcase 6 - (Hard) Weightage 25%**

Input: 4

1 7 9 8

8

Output:0 1

**5 Marks Coding Question**

**Problem Statement**

Given two integer arrays nums1 and nums2, return an array of their intersection. Each element in the result must be unique and you may return the result

**Input Format:**

Size1=4

Size2=2

nums1 = [1,2,3,4]

nums2 = [2,6]

**Output Format:**

2

**Code Constraints:**

1 <= nums1.length, nums2.length <= 1000

0 <= nums1[i], nums2[i] <= 1000

**Solution: (in required language)**

#include <iostream>

#include <vector>

#include <unordered\_map>

using namespace std;

void printCommonElements(const vector<int>& nums1, const vector<int>& nums2) {

unordered\_map<int, int> countMap;

for (int num : nums1) {

countMap[num]++;

}

cout << "Common elements in both arrays are: ";

for (int num : nums2) {

if (countMap[num] > 0) {

cout << num << " ";

countMap[num]--;

}

}

cout << endl;

}

int main() {

// Input size of the first array

int size1;

cout << "Enter the size of the first array: ";

cin >> size1;

// Input elements of the first array

vector<int> nums1(size1);

cout << "Enter the elements of the first array: ";

for (int i = 0; i < size1; ++i) {

cin >> nums1[i];

}

// Input size of the second array

int size2;

cout << "Enter the size of the second array: ";

cin >> size2;

// Input elements of the second array

vector<int> nums2(size2);

cout << "Enter the elements of the second array: ";

for (int i = 0; i < size2; ++i) {

cin >> nums2[i];

}

// Print common elements

printCommonElements(nums1, nums2);

return 0;

}

**Sample Testcase 1**

Input: 4

4

1 2 3 4

4 5 6 7

Output: 4

**Sample Testcase 2**

Input:2

2

1 2

2 3

Output: 2

**Hidden Testcase 1 - (Easy) Weightage 10%**

Input: 3

3

1 2 3

2 3 4

Output: 2 3

**Hidden Testcase 2 - (Easy) Weightage 10%**

Input: 3

3

1 2 4

4 5 6

Output: 4

**Hidden Testcase 3 - (Medium) Weightage 15%**

Input: 5

5

1 2 3 4 5

3 4 5 6 7

Output: 3 4 5

**Hidden Testcase 4 - (Medium) Weightage 15%**

Input:

2

2

1 3

3 5

Output: 3

**Hidden Testcase 5 - (Hard) Weightage 25%**

Input:4

4

1 2 3 5

4 5 6 7

Output: 5

**Hidden Testcase 6 - (Hard) Weightage 25%**

Input: 1

1

1

1

Output: 1

**5 Marks Coding Question**

**Problem Statement**

take input of a string from a user and print its concentration respectively

**Input Format:**

S=hello

**Output Format:**

1 1 2 2 1

**Code Constraints:**

**1<=s.length<=10**

**Solution: (in required language)**

#include <iostream>

#include <string>

#include <unordered\_map>

void printConcentration(const std::string& str) {

std::unordered\_map<char, int> frequency;

// Count frequency of each character in the string

for (char ch : str) {

frequency[ch]++;

}

// Print concentration of characters

for (char ch : str) {

std::cout << frequency[ch] << " ";

}

std::cout << std::endl;

}

int main() {

// Input string from the user

std::string input;

std::cout << "Enter a string: ";

std::cin >> input;

// Print concentration of characters

std::cout << "Concentration of characters in the string: ";

printConcentration(input);

return 0;

}

**Sample Testcase 1**

Input: hello

Output: 1 1 2 2 1

**Sample Testcase 2**

Input: aab

Output: 2 2 1

**Hidden Testcase 1 - (Easy) Weightage 10%**

Input: aaa

Output: 3 3 3

**Hidden Testcase 2 - (Easy) Weightage 10%**

Input: fghjj

Output: 1 1 1 2 2

**Hidden Testcase 3 - (Medium) Weightage 15%**

Input: ggg

Output: 3 3 3

**Hidden Testcase 4 - (Medium) Weightage 15%**

Input: lklk

Output: 2 2 2 2

**Hidden Testcase 5 - (Hard) Weightage 25%**

Input:dfggfd

Output: 2 2 2 2 2 2

**Hidden Testcase 6 - (Hard) Weightage 25%**

Input: aa

Output: 2 2

**5 Marks Coding Question**

**Problem Statement**

Given an array nums of size n, print the majority element.

The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

**Input Format:**

N=3

nums = [3,2,3]

**Output Format:**

3

**Code Constraints:**

n == nums.length

1 <= n <= 5 \* 104

-109 <= nums[i] <= 109

**Solution: (in required language)**

**#include <iostream>**

**#include <vector>**

**#include <unordered\_map>**

**int findMajorityElement(const std::vector<int>& nums) {**

**std::unordered\_map<int, int> countMap;**

**int majorityElement = -1;**

**int majorityCount = 0;**

**for (int num : nums) {**

**countMap[num]++;**

**if (countMap[num] > majorityCount) {**

**majorityElement = num;**

**majorityCount = countMap[num];**

**}**

**}**

**return majorityElement;**

**}**

**int main() {**

**// Input size of the array**

**int size;**

**std::cout << "Enter the size of the array: ";**

**std::cin >> size;**

**// Input array elements**

**std::vector<int> nums(size);**

**std::cout << "Enter the elements of the array: ";**

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

**std::cin >> nums[i];**

**}**

**// Find the majority element**

**int majorityElement = findMajorityElement(nums);**

**// Print the majority element**

**std::cout << "The majority element is: " << majorityElement << std::endl;**

**return 0;**

**}**

**Sample Testcase 1**

Input: 3

3 2 3

Output: 3

**Sample Testcase 2**

Input: 3

2 3 2

Output: 2

**Hidden Testcase 1 - (Easy) Weightage 10%**

Input: 1

1

Output: 1

**Hidden Testcase 2 - (Easy) Weightage 10%**

Input: 3

1 1 2

Output: 1

**Hidden Testcase 3 - (Medium) Weightage 15%**

Input: 5

1 1 1 6 7

Output: 1

**Hidden Testcase 4 - (Medium) Weightage 15%**

Input: 5

2 2 5 6 2

Output: 2

**Hidden Testcase 5 - (Hard) Weightage 25%**

Input:3

1 2 2

Output:2

**Hidden Testcase 6 - (Hard) Weightage 25%**

Input: 5

6 6 6 8 6

Output: 6