**SCT212-0193/2022: SHARON NYABOKE**

**DSA ASSIGNMENT: QUIZ 1**

**QUESTION 1**

Remove duplicates from sorted arrays:

#include <iostream>

#include <vector>

using namespace std;

int removeDuplicates(vector<int>& nums)

{

if (nums.empty())

return 0;

int slow = 0;

for (int fast = 1; fast < nums.size(); ++fast)

{

if (nums[fast] != nums[slow]) {

++slow;

nums[slow] = nums[fast];

}

}

return slow + 1;

}

int main() {

vector<int> nums = {0, 0, 1, 1, 1, 2, 2, 3, 3, 4};

int newLength = removeDuplicates(nums);

cout << "New length: " << newLength << endl;

cout << "Array after removing duplicates: ";

for (int i = 0; i < newLength; ++i)

cout << nums[i] << " ";

cout << endl;

return 0;

}

Explanation:

We define a function **removeDuplicates** that takes a vector of integers as input, modifies it in-place to remove duplicates, and returns the new length of the modified array.

**QUESTION 2**

Rotate array:

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

void reverseArray(vector<int>& nums, int start, int end) {

while (start < end) {

swap(nums[start], nums[end]);

start++;

end--;

}

}

void rotate(vector<int>& nums, int k) {

int n = nums.size();

k = k % n; // Normalize k in case it's greater than the size of the array

// Reverse the whole array

reverseArray(nums, 0, n - 1);

// Reverse the first k elements

reverseArray(nums, 0, k - 1);

// Reverse the remaining elements after k

reverseArray(nums, k, n - 1);

}

int main() {

vector<int> nums = {1, 2, 3, 4, 5, 6, 7};

int k = 3;

rotate(nums, k);

cout << "Array after rotating by " << k << " steps to the right: ";

for (int num : nums) {

cout << num << " ";

}

cout << endl;

return 0;

}

Explanation:

The code defines a function ‘rotate’ that takes a vector of integers ‘nums’ and an integer ‘k’ as input and rotates the array to the right by ‘k’ steps. The ‘main’ function shows how to use this function with an example array

**QUESTION 3**

Contains duplicates:

#include <iostream>

#include <vector>

#include <unordered\_set>

using namespace std;

bool containsDuplicate(vector<int>& nums) {

unordered\_set<int> seen;

for (int num : nums) {

if (seen.find(num) != seen.end()) {

return true; // Found a duplicate

}

seen.insert(num);

}

return false; // No duplicates found

}

int main() {

vector<int> nums1 = {1, 2, 3, 4, 5};

vector<int> nums2 = {1, 2, 3, 4, 1};

cout << "nums1 contains duplicates: " << (containsDuplicate(nums1) ? "true" : "false") << endl;

cout << "nums2 contains duplicates: " << (containsDuplicate(nums2) ? "true" : "false") << endl;

return 0;

}

Explanation:

We iterate through the array, and for each element, we check if its already in the set. If it is, we’ve found a duplicate then we’ll retuorn ‘true’ . If we finish iterating through the entire array without finding any duplicates we return ‘false’.

**QUESTION 4**

Single number:

#include <iostream>

#include <vector>

using namespace std;

int singleNumber(vector<int>& nums) {

int result = 0;

for (int num : nums) {

result ^= num;

}

return result;

}

int main() {

vector<int> nums = {4, 1, 2, 1, 2};

cout << "The single number is: " << singleNumber(nums) << endl;

return 0;

}

Explanation:

We initialize **result** to 0. Then, we iterate through the array and apply XOR operation on each element with the **result**. At the end of the loop, **result** will hold the value of the single number that appears only once.