**QUESTION 1**

Use two pointers, one to iterate over the array and one to keep track of the unique elements. Whenever you encounter a duplicate element, you skip it and move the first pointer forward. Whenever you encounter a non-duplicate element, you copy it to the second pointer and move both pointers forward. This way, you can modify the array in-place and return the new length as the position of the second pointer.

#include <iostream>

#include <vector>

using namespace std;

int removeDuplicates(vector<int>& nums) {

// Check if the array is empty

if (nums.empty()) {

return 0;

}

// Initialize two pointers

int i = 0; // The first pointer to iterate over the array

int j = 0; // The second pointer to keep track of the unique elements

// Loop through the array

while (i < nums.size()) {

// If the current element is not a duplicate

if (nums[i] != nums[j]) {

// Copy it to the second pointer

j++;

nums[j] = nums[i];

}

// Move the first pointer forward

i++;

}

// Return the new length as the position of the second pointer

return j + 1;

}

int main() {

// Test the function with an example array

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

int len = removeDuplicates(nums);

cout << "The new length is: " << len << endl;

cout << "The modified array is: ";

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

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

}

cout << endl;

return 0;

}

**QUESTION 2**

Use a reverse function that reverses a part of the array from a given start index to a given end index. Then, you can apply the reverse function three times: first, to the whole array; second, to the first k elements; and third, to the remaining n - k elements. This way, you can rotate the array to the right by k steps in-place.

#include <iostream>

#include <vector>

using namespace std;

// Define a function to reverse a part of the array

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

// Loop until the start and end pointers meet or cross

while (start < end) {

// Swap the elements at the start and end pointers

int temp = nums[start];

nums[start] = nums[end];

nums[end] = temp;

// Move the pointers towards the middle

start++;

end--;

}

}

// Define a function to rotate the array to the right by k steps

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

// Check if the array is empty or k is zero

if (nums.empty() || k == 0) {

return;

}

// Adjust k to be within the range of the array length

k = k % nums.size();

// Reverse the whole array

reverse(nums, 0, nums.size() - 1);

// Reverse the first k elements

reverse(nums, 0, k - 1);

// Reverse the remaining n - k elements

reverse(nums, k, nums.size() - 1);

}

int main() {

// Test the function with an example array

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

int k = 3;

rotate(nums, k);

cout << "The rotated array is: ";

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

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

}

cout << endl;

return 0;

}

**QUESTION 3**

Use a hash set to store the elements of the array and check if any element is already present in the set. If yes, then the array contains duplicates and the function returns true. If no, then the array does not contain duplicates and the function returns false.

#include <iostream>

#include <unordered\_set>

#include <vector>

using namespace std;

bool containsDuplicate(vector<int>& nums) {

// Check if the array is empty

if (nums.empty()) {

return false;

}

// Create a hash set to store the elements of the array

unordered\_set<int> seen;

// Loop through the array

for (int num : nums) {

// If the element is already in the set, return true

if (seen.count(num) > 0) {

return true;

}

// Otherwise, add the element to the set

seen.insert(num);

}

// If the loop ends without returning true, return false

return false;

}

int main() {

// Test the function with an example array

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

bool result = containsDuplicate(nums);

cout << "The result is: " << result << endl;

return 0;

}

**QUESTION 4**

One possible solution is to use the XOR operation, which has the following properties:

XOR of a number with itself is 0.

XOR of a number with 0 is the number itself.

XOR is commutative and associative.

Therefore, if we XOR all the elements of the array, the result will be the single element that appears only once.

#include <stdio.h>

int singleNumber(int nums[], int n) {

// Initialize the result to 0

int result = 0;

// Loop through the array

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

// XOR the current element with the result

result = result ^ nums[i];

}

// Return the result

return result;

}

int main() {

// Test the function with an example array

int nums[] = {2, 2, 1, 4, 1};

int n = sizeof(nums) / sizeof(nums[0]);

int single = singleNumber(nums, n);

printf("The single element is: %d\n", single);

return 0;

}