NAME: NATHAN SIKALO
REG NO: SCT212-0189/2022
UNIT NAME: DATA STRUCTURES AND ALGORITHM

Quiz 1 Array
1. QUESTION ONE.

```
#include <iostream>
#include <vector>

int removeDuplicates(std::vector<int>& nums) {
    if (nums.empty()) {
        return 0;
    }

    int uniqueIndex = 0;

    for (int i = 1; i < nums.size(); ++i) {
        if (nums[i] != nums[uniqueIndex]) {
            ++uniqueIndex;
            nums[uniqueIndex] = nums[i];
        }
    }

    return uniqueIndex + 1;
}

int main() {
    // Example usage
    std::vector<int> nums = {1, 1, 2, 2, 3, 4, 4, 5};
    int newLength = removeDuplicates(nums);

    std::cout << "New Length: " << newLength << std::endl;

    for (int i = 0; i < newLength; ++i) {
        std::cout << nums[i] << " ";
    }

    return 0;
}
```

2. QUESTION TWO
This program defines a function **rotateArray** that takes a vector of integers and an integer **k** as input and rotates the array to the right by **k** steps. The main function demonstrates the usage with an example array and the specified number of steps.

```
#include <iostream>
#include <vector>

void rotateArray(std::vector<int>& nums, int k) {
    int n = nums.size();
```

```
    k = k % n;  // In case k is greater than the array size

    // Create a temporary array to store the rotated elements
    std::vector<int> temp(nums.begin() + n - k, nums.end());

    // Shift the elements to the right by k steps
    for (int i = n - k - 1; i >= 0; --i) {
        nums[i + k] = nums[i];
    }

    // Copy the temporary array to the beginning of the original array
    for (int i = 0; i < k; ++i) {
        nums[i] = temp[i];
    }
}

int main() {
    // Example usage
    std::vector<int> nums = {1, 2, 3, 4, 5};
    int k = 2;

    rotateArray(nums, k);

    for (int num : nums) {
        std::cout << num << " ";
    }

    return 0;
}
```

3. QUESTION THREE

This program defines a function **containsDuplicate** that takes a vector of integers as input and returns **true** if there are any duplicates, and **false** otherwise. It uses an **unordered_set** to keep track of unique elements encountered while iterating through the array. If a duplicate is found, the function returns **true**; otherwise, it returns **false**. The main function demonstrates the usage with an example array.

```
#include <iostream>
#include <vector>
#include <unordered_set>

bool containsDuplicate(std::vector<int>& nums) {
    std::unordered_set<int> numSet;

    for (int num : nums) {
        // If the number is already in the set, it's a duplicate
        if (numSet.find(num) != numSet.end()) {
            return true;
        }
```

```
        // Otherwise, add the number to the set
        numSet.insert(num);
    }

    // If the loop completes, no duplicates were found
    return false;
}

int main() {
    // Example usage
    std::vector<int> nums = {1, 2, 3, 4, 5, 2};

    if (containsDuplicate(nums)) {
        std::cout << "Contains duplicates." << std::endl;
    } else {
        std::cout << "No duplicates found." << std::endl;
    }

    return 0;
}
```

4. QUESTION FOUR

This program defines a function `singleNumber` that takes a vector of integers as input and uses bitwise XOR to find the single number. The XOR operation has the property that XOR-ing a number with itself results in 0, so when you XOR all elements in the array, the duplicates cancel each other out, and you are left with the single number. The main function demonstrates the usage with an example array.

```cpp
#include <iostream>
#include <vector>

int singleNumber(std::vector<int>& nums) {
    int result = 0;

    // XOR all elements in the array
    for (int num : nums) {
        result ^= num;
    }

    return result;
}

int main() {
    // Example usage
    std::vector<int> nums = {1, 2, 3, 4, 3, 2, 1};

    int single = singleNumber(nums);

    std::cout << "The single number is: " << single << std::endl;

    return 0;
}
```