Q-1: Given an array containing only 0s, 1s, and 2s, sort the array in ascending order using the Dutch National Flag Algorithm. The algorithm should rearrange the elements in the array so that all 0s come before 1s, and all 1s come before 2s.

Sample test case:

|  |
| --- |
| Input: arr=[-2, -3, 4, -1, -2, 1, 5, -3]  Output: Maximum subarray sum: 7 |

Solution:

#include <iostream>

using namespace std;

// Dutch National Flag Algorithm

// This function takes an array 'arr' and its size 'n' as input.

// It sorts the array in-place in ascending order, where the elements are colors represented as integers (0, 1, or 2).

// The algorithm rearranges the elements so that all 0s come before 1s, and all 1s come before 2s.

void sortColors(int arr[], int n) {

int low = 0, mid = 0, high = n - 1;

int temp;

// The algorithm uses three pointers: 'low', 'mid', and 'high'.

// 'low' points to the boundary between 0s and 1s.

// 'mid' points to the boundary between 1s and unprocessed elements.

// 'high' points to the boundary between unprocessed elements and 2s.

while (mid <= high) {

switch (arr[mid]) {

case 0: {

// If the element at 'mid' is 0, swap it with the element at 'low'.

// Move 'low' and 'mid' one step forward.

temp = arr[low];

arr[low] = arr[mid];

arr[mid] = temp;

low++;

mid++;

break;

}

case 1: {

// If the element at 'mid' is 1, move 'mid' one step forward.

mid++;

break;

}

case 2: {

// If the element at 'mid' is 2, swap it with the element at 'high'.

// Move 'high' one step backward.

temp = arr[mid];

arr[mid] = arr[high];

arr[high] = temp;

high--;

break;

}

}

}

}

int main() {

// Sample array of colors represented as integers (0, 1, or 2)

int arr[] = {0, 1, 2, 1, 2, 0};

// Calculate the size of the array

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

// Print the unsorted array

cout << "Unsorted array: ";

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

cout << arr[i] << " ";

// Call the sortColors function to sort the array in-place

sortColors(arr, n);

// Print the sorted array

cout << "\nSorted array: ";

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

cout << arr[i] << " ";

return 0;

}

Q-2: Given two integer arrays A and B, find the maximum length of a subarray that appears in both arrays.

Sample test case:

|  |
| --- |
| Input: A = {1, 2, 3, 2, 1}, B = {3, 2, 1, 4, 7}  Output: Maximum length of a common subarray: 3 |

Solution:

#include <iostream>

#include <vector>

using namespace std;

int findMaxLengthOfCommonSubarray(vector<int>& A, vector<int>& B) {

int m = A.size();

int n = B.size();

// Initialize the dp table with all elements set to zero

vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));

int maxLength = 0;

// Iterate through the arrays A and B

for (int i = 1; i <= m; i++) {

for (int j = 1; j <= n; j++) {

// Check if elements are equal

if (A[i - 1] == B[j - 1]) {

// Extend the common subarray

dp[i][j] = dp[i - 1][j - 1] + 1;

maxLength = max(maxLength, dp[i][j]);

}

// If elements are not equal, dp[i][j] remains zero

}

}

return maxLength;

}

int main() {

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

vector<int> B = {3, 2, 1, 4, 7};

int maxLength = findMaxLengthOfCommonSubarray(A, B);

cout << "Maximum length of a common subarray: " << maxLength << endl;

return 0;

}

Q-3: Given an array and an integer K, find the subarray of exactly K elements with the maximum average.

Sample test case:

|  |
| --- |
| Input: nums = {1, 12, -5, -6, 50, 3}  K = 4  Output: Subarray with maximum average: 12 -5 -6 50 |
| Solution: |

#include <iostream>

#include <vector>

std::vector<int> findMaxAverageSubarray(const std::vector<int>& nums, int K) {

int n = nums.size();

// Check if the input array is valid

if (n < K || K <= 0) {

std::cout << "Invalid input: The array is smaller than K or K is non-positive." << std::endl;

return {};

}

int sum = 0;

int maxSum = 0;

int startIndex = 0;

// Calculate the sum of the first K elements to initialize maxSum

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

sum += nums[i];

}

maxSum = sum;

// Find the maximum sum of subarrays with exactly K elements

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

sum += nums[i] - nums[i - K]; // Moving window technique

if (sum > maxSum) {

maxSum = sum;

startIndex = i - K + 1;

}

}

// Create the result subarray with K elements that has the maximum average

std::vector<int> result;

for (int i = startIndex; i < startIndex + K; ++i) {

result.push\_back(nums[i]);

}

return result;

}

int main() {

std::vector<int> nums = {1, 12, -5, -6, 50, 3};

int K = 4;

std::vector<int> maxAverageSubarray = findMaxAverageSubarray(nums, K);

// Print the subarray with maximum average

std::cout << "Subarray with maximum average: ";

for (int num : maxAverageSubarray) {

std::cout << num << " ";

}

std::cout << std::endl;

return 0;

}

Q-4: Given a rotated sorted array and a target value, determine if the target is present in the array. If it is present, return its index; otherwise, return -1.

Sample test case:

|  |
| --- |
| Input: arr = {4, 5, 6, 7, 0, 1, 2}  target= 0  Output: Target found at index: 4 |

Solution:

#include <iostream>

using namespace std;

int searchRotatedArray(int arr[], int n, int target) {

int left = 0, right = n - 1;

while (left <= right) {

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

if (arr[mid] == target) {

return mid;

}

if (arr[left] <= arr[mid]) {

if (target >= arr[left] && target < arr[mid]) {

right = mid - 1;

} else {

left = mid + 1;

}

} else {

if (target > arr[mid] && target <= arr[right]) {

left = mid + 1;

} else {

right = mid - 1;

}

}

}

return -1;

}

int main() {

int arr[] = {4, 5, 6, 7, 0, 1, 2};

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

int target = 0;

int index = searchRotatedArray(arr, n, target);

if (index != -1) {

cout << "Target found at index: " << index;

} else {

cout << "Target not found in the array.";

}

return 0;

}