Q-1: Given an array of integers, find the contiguous subarray with the largest sum.

Time complexity should be O(n).

Sample test case:

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

Solution:

#include <iostream>

#include <climits>

using namespace std;

int maxSubarraySum(int arr[], int n) {

int max\_so\_far = INT\_MIN;

int max\_ending\_here = 0;

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

max\_ending\_here = max\_ending\_here + arr[i];

if (max\_so\_far < max\_ending\_here)

max\_so\_far = max\_ending\_here;

if (max\_ending\_here < 0)

max\_ending\_here = 0;

}

return max\_so\_far;

}

int main() {

int arr[] = {-2, -3, 4, -1, -2, 1, 5, -3};

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

cout << "Maximum subarray sum: " << maxSubarraySum(arr, n);

return 0;

}

Q-2: Given an array of non-negative integers representing the maximum jump length from each position, determine the minimum number of jumps needed to reach the last index. The time complexity should be O(n).

Sample test case:

|  |
| --- |
| Input: arr={2, 3, 1, 1, 4}  Output: Minimum jumps required: 2 |

Solution:

#include <iostream>

using namespace std;

int minJumps(int arr[], int n) {

if (n <= 1)

return 0;

int maxReach = arr[0];

int steps = arr[0];

int jumps = 1;

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

if (i == n - 1)

return jumps;

maxReach = max(maxReach, i + arr[i]);

steps--;

if (steps == 0) {

jumps++;

steps = maxReach - i;

}

}

return -1; // If the end is not reachable

}

int main() {

int arr[] = {2, 3, 1, 1, 4};

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

int minJumpsRequired = minJumps(arr, n);

cout << "Minimum jumps required: " << minJumpsRequired;

return 0;

}

Q-3: Given an array of non-negative integers representing the heights of vertical lines, find two lines that, together with the x-axis, form a container that contains the most water.

The program should run with time complexity O(n).

Sample test case:

|  |
| --- |
| Input: arr[] = {3, 0, 2, 0, 4}  Output: 7 |
| Solution: |

#include <bits/stdc++.h>

using namespace std;

int findWater(int arr[], int n)

{

// left[i] contains height of tallest bar to the

// left of i'th bar including itself

int left[n];

// Right [i] contains height of tallest bar to

// the right of ith bar including itself

int right[n];

// Initialize result

int water = 0;

// Fill left array

left[0] = arr[0];

for (int i = 1; i < n; i++)

left[i] = max(left[i - 1], arr[i]);

// Fill right array

right[n - 1] = arr[n - 1];

for (int i = n - 2; i >= 0; i--)

right[i] = max(right[i + 1], arr[i]);

// Calculate the accumulated water element by element

// consider the amount of water on i'th bar, the

// amount of water accumulated on this particular

// bar will be equal to min(left[i], right[i]) - arr[i]

// .

for (int i = 1; i < n - 1; i++) {

int var = min(left[i - 1], right[i + 1]);

if (var > arr[i]) {

water += var - arr[i];

}

}

return water;

}

int main()

{

int arr[] = { 0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1 };

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

cout << findWater(arr, n);

return 0;

}

Q-4: Given an array of integers, find all elements that appear more than once in the array.

Time complexity should be O(n).

Sample test case:

|  |
| --- |
| Input: arr= {4, 3, 2, 7, 8, 2, 9, 1,9}  Output: 2, 9 |

Solution:

#include <iostream>

using namespace std;

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

const int MAX = 100000; // Assuming maximum value of array elements

int bitset[MAX] = {0};

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

if (bitset[arr[i]] == 1) {

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

}

bitset[arr[i]]++;

}

}

int main() {

int arr[] = {4, 3, 2, 7, 8, 2, 9, 1};

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

cout << "Duplicate elements: ";

findDuplicates(arr, n);

return 0;

}