Index Of Problems

Prefix Sum

Range Sum Query - Immutable

Given an integer array nums, handle multiple queries of the following type:

Calculate the sum of the elements of nums between indices left and right inclusive where left <= right.

Implement the NumArray class:

NumArray(int[] nums) Initializes the object with the integer array nums.
int sumRange(int left, int right) Returns the sum of the elements of nums between indices left and right inclusive (i.e. nums[left] + nums[left + 1] + ... + nums[right]).

Example 1:

Input
["NumArray", "sumRange", "sumRange", "sumRange"]
[[[-2, 0, 3, -5, 2, -1]], [0, 2], [2, 5], [0, 5]]
Output
[null, 1, -1, -3]

Explanation
NumArray numArray = new NumArray([-2, 0, 3, -5, 2, -1]);
numArray.sumRange(0, 2); // return (-2) + 0 + 3 = 1
numArray.sumRange(2, 5); // return 3 + (-5) + 2 + (-1) = -1
numArray.sumRange(0, 5); // return (-2) + 0 + 3 + (-5) + 2 + (-1) = -3

Constraints:

$1 <= nums.length <= 10^4$
$-10^5 <= nums[i] <= 10^5$
$0 <= left <= right < nums.length$
At most 10^4 calls will be made to sumRange.

Contiguous Array

Given a binary array nums, return the maximum length of a contiguous subarray with an equal number of 0 and 1.

Example 1:

Input: nums = [0,1]
Output: 2
Explanation: [0, 1] is the longest contiguous subarray with an equal number of 0 and 1.

Example 2:

Input: nums = [0,1,0]
Output: 2
Explanation: [0, 1] (or [1, 0]) is a longest contiguous subarray with equal number of 0 and 1.

Constraints:

1 <= nums.length <= 10^5
nums[i] is either 0 or 1.

Subarray Sum Equals K

Given an array of integers nums and an integer k, return the total number of subarrays whose sum equals to k.
A subarray is a contiguous non-empty sequence of elements within an array.

Example 1:
Input: nums = [1,1,1], k = 2
Output: 2
Example 2:
Input: nums = [1,2,3], k = 3
Output: 2

Constraints:

1 <= nums.length <= $2 * 10^4$
-1000 <= nums[i] <= 1000
-10^7 <= k <= 10^7

Two Pointers

Two Sum II - Input Array Is Sorted

Given a 1-indexed array of integers numbers that is already sorted in non-decreasing order, find two numbers such that they add up to a specific target number. Let these two numbers be numbers[index1] and numbers[index2] where 1 <= index1 < index2 <= numbers.length.
Return the indices of the two numbers, index1 and index2, added by one as an integer array [index1, index2] of length 2.
The tests are generated such that there is exactly one solution. You may not use the same element twice.
Your solution must use only constant extra space.

Example 1:

Input: numbers = [2,7,11,15], target = 9
Output: [1,2]
Explanation: The sum of 2 and 7 is 9. Therefore, index1 = 1, index2 = 2. We return [1, 2].

Example 2:

Input: numbers = [2,3,4], target = 6
Output: [1,3]
Explanation: The sum of 2 and 4 is 6. Therefore index1 = 1, index2 = 3. We return [1, 3].

Example 3:

Input: numbers = [-1,0], target = -1
Output: [1,2]
Explanation: The sum of -1 and 0 is -1. Therefore index1 = 1, index2 = 2. We return [1, 2].

Constraints:

$2 <= numbers.length <= 3 * 10^4$
$-1000 <= numbers[i] <= 1000$
numbers is sorted in non-decreasing order.
$-1000 <= target <= 1000$

3Sum

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.
Notice that the solution set must not contain duplicate triplets.

Example 1:

Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
Explanation:
nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.
nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.
nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.
The distinct triplets are [-1,0,1] and [-1,-1,2].
Notice that the order of the output and the order of the triplets does not matter.

Example 2:

Input: nums = [0,1,1]
Output: []
Explanation: The only possible triplet does not sum up to 0.

Example 3:

Input: nums = [0,0,0]
Output: [[0,0,0]]
Explanation: The only possible triplet sums up to 0.

Constraints:

3 <= nums.length <= 3000
-10^5 <= nums[i] <= 10^5

Container With Most Water

You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the i^th line are (i, 0) and (i, height[i]).
Find two lines that together with the x-axis form a container, such that the container contains the most water.
Return the maximum amount of water a container can store.
Notice that you may not slant the container.

Example 1:

Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

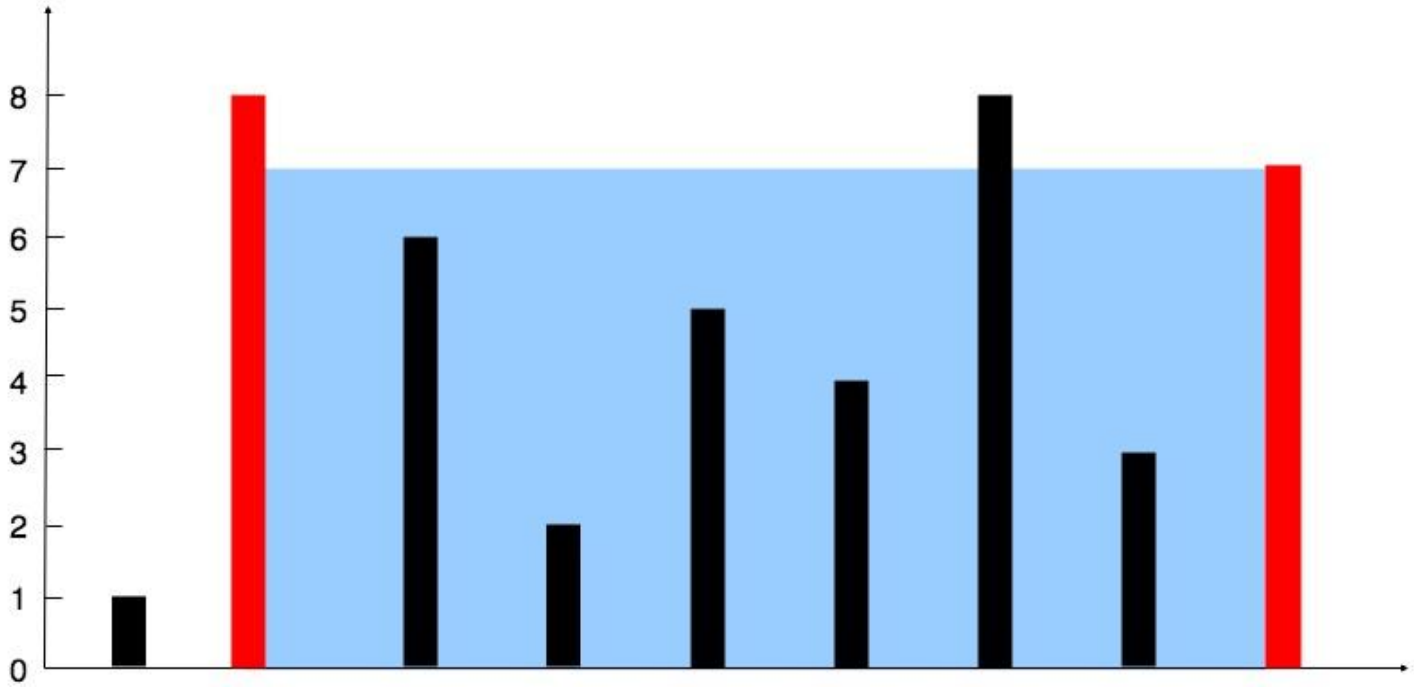
Example 2:

Input: height = [1,1]
Output: 1

Constraints:

n == height.length
$2 <= n <= 10^5$
$0 <= height[i] <= 10^4$

Images for Container With Most Water:



Sliding Window

Maximum Average Subarray I

You are given an integer array nums consisting of n elements, and an integer k.
Find a contiguous subarray whose length is equal to k that has the maximum average value and return this value. Any answer with a calculation error less than 10□⁵ will be accepted.

Example 1:

Input: nums = [1,12,-5,-6,50,3], k = 4
Output: 12.75000
Explanation: Maximum average is (12 - 5 - 6 + 50) / 4 = 51 / 4 = 12.75

Example 2:

Input: nums = [5], k = 1
Output: 5.00000

Constraints:

n == nums.length
1 <= k <= n <= 10^5
-10^4 <= nums[i] <= 10^4

Longest Substring Without Repeating Characters

Given a string s, find the length of the longest substring without repeating characters.

Example 1:

Input: s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.

Example 2:

Input: s = "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.

Example 3:

Input: s = "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.

Constraints:

$0 <= s.length <= 5 * 10^4$
s consists of English letters, digits, symbols and spaces.

Minimum Window Substring

Given two strings s and t of lengths m and n respectively, return the minimum window substring of s such that every character in t (including duplicates) is included in the window. If there is no such substring, return the empty string "".
The testcases will be generated such that the answer is unique.

Example 1:

Input: s = "ADOBECODEBANC", t = "ABC"
Output: "BANC"
Explanation: The minimum window substring "BANC" includes 'A', 'B', and 'C' from string t.

Example 2:

Input: s = "a", t = "a"
Output: "a"
Explanation: The entire string s is the minimum window.

Example 3:

Input: s = "a", t = "aa"
Output: ""
Explanation: Both 'a's from t must be included in the window.
Since the largest window of s only has one 'a', return empty string.

Constraints:

m == s.length
n == t.length
$1 <= m, n <= 10^5$
s and t consist of uppercase and lowercase English letters.

Follow up: Could you find an algorithm that runs in O(m + n) time?