# Question

Given an array of integers nums, calculate the **pivot index** of this array.

The **pivot index** is the index where the sum of all the numbers **strictly** to the left of the index is equal to the sum of all the numbers **strictly** to the index's right.

If the index is on the left edge of the array, then the left sum is 0 because there are no elements to the left. This also applies to the right edge of the array.

Return *the****leftmost pivot index***. If no such index exists, return -1.

**Example 1:**

**Input:** nums = [1,7,3,6,5,6]

**Output:** 3

**Explanation:**

The pivot index is 3.

Left sum = nums[0] + nums[1] + nums[2] = 1 + 7 + 3 = 11

Right sum = nums[4] + nums[5] = 5 + 6 = 11

**Example 2:**

**Input:** nums = [1,2,3]

**Output:** -1

**Explanation:**

There is no index that satisfies the conditions in the problem statement.

**Example 3:**

**Input:** nums = [2,1,-1]

**Output:** 0

**Explanation:**

The pivot index is 0.

Left sum = 0 (no elements to the left of index 0)

Right sum = nums[1] + nums[2] = 1 + -1 = 0

**Constraints:**

* 1 <= nums.length <= 104
* -1000 <= nums[i] <= 1000

# Solution

#### **Approach #1: Prefix Sum [Accepted]**

**Intuition and Algorithm**

We need to quickly compute the sum of values to the left and the right of every index.

Let's say we knew S as the sum of the numbers, and we are at index i. If we knew the sum of numbers leftsum that are to the left of index i, then the other sum to the right of the index would just be S - nums[i] - leftsum.

As such, we only need to know about leftsum to check whether an index is a pivot index in constant time. Let's do that: as we iterate through candidate indexes i, we will maintain the correct value of leftsum.

|  |
| --- |
| class Solution {  public int pivotIndex(int[] nums) {  int sum = 0, leftsum = 0;  for (int x: nums) sum += x;  for (int i = 0; i < nums.length; ++i) {  if (leftsum == sum - leftsum - nums[i]) return i;  leftsum += nums[i];  }  return -1;  }  } |

**Complexity Analysis**

* Time Complexity: O(N)*O*(*N*), where N*N* is the length of nums.
* Space Complexity: O(1)*O*(1), the space used by leftsum and S