# Problem 724: Find Pivot Index

## Problem Information

**Difficulty:** Easy
**Acceptance Rate:** 0.00%
**Paid Only:** No

## Problem Description

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 <= 10

4

-1000 <= nums[i] <= 1000

Note:

This question is the same as 1991:

https://leetcode.com/problems/find-the-middle-index-in-array/

## Code Snippets

**C++:**

```
class Solution {
public:
int pivotIndex(vector<int>& nums) {

}
};
```

**Java:**

```
class Solution {
public int pivotIndex(int[] nums) {

}
}
```

**Python3:**

```
class Solution:
def pivotIndex(self, nums: List[int]) -> int:
```

**Python:**

```
class Solution(object):
def pivotIndex(self, nums):
"""
:type nums: List[int]
:rtype: int
"""
```

**JavaScript:**

```
/**
* @param {number[]} nums
* @return {number}
*/
var pivotIndex = function(nums) {

};
```

**TypeScript:**

```
function pivotIndex(nums: number[]): number {
```

```
};
```

**C#:**

```
public class Solution {
public int PivotIndex(int[] nums) {

}
}
```

**C:**

```
int pivotIndex(int* nums, int numsSize) {

}
```

**Go:**

```
func pivotIndex(nums []int) int {

}
```

**Kotlin:**

```
class Solution {
fun pivotIndex(nums: IntArray): Int {

}
}
```

**Swift:**

```
class Solution {
func pivotIndex(_ nums: [Int]) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn pivot_index(nums: Vec<i32>) -> i32 {
```

```
}
}
```

**Ruby:**

```
# @param {Integer[]} nums
# @return {Integer}
def pivot_index(nums)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function pivotIndex($nums) {

}
}
```

**Dart:**

```
class Solution {
int pivotIndex(List<int> nums) {

}
}
```

**Scala:**

```
object Solution {
def pivotIndex(nums: Array[Int]): Int = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec pivot_index(nums :: [integer]) :: integer
def pivot_index(nums) do

end
end
```

**Erlang:**

```
-spec pivot_index(Nums :: [integer()]) -> integer().
pivot_index(Nums) ->
.
```

**Racket:**

```
(define/contract (pivot-index nums)
(-> (listof exact-integer?) exact-integer?)
)
```

## Solutions

**C++ Solution:**

```
/*
* Problem: Find Pivot Index
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int pivotIndex(vector<int>& nums) {

}
};
```

**Java Solution:**

```
/**
* Problem: Find Pivot Index
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int pivotIndex(int[] nums) {

}
}
```

### Python3 Solution:

```
"""
Problem: Find Pivot Index
Difficulty: Easy
Tags: array

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def pivotIndex(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def pivotIndex(self, nums):
"""
:type nums: List[int]
:rtype: int
"""
```

### JavaScript Solution:

```
/**
* Problem: Find Pivot Index
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[]} nums
* @return {number}
*/
var pivotIndex = function(nums) {

};
```

### TypeScript Solution:

```
/**
* Problem: Find Pivot Index
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function pivotIndex(nums: number[]): number {

};
```

### C# Solution:

```
/*
* Problem: Find Pivot Index
* Difficulty: Easy
* Tags: array
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int PivotIndex(int[] nums) {

}
}
```

### C Solution:

```
/*
* Problem: Find Pivot Index
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int pivotIndex(int* nums, int numsSize) {

}
```

### Go Solution:

```
// Problem: Find Pivot Index
// Difficulty: Easy
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func pivotIndex(nums []int) int {

}
```

### Kotlin Solution:

```
class Solution {
fun pivotIndex(nums: IntArray): Int {

}
}
```

### Swift Solution:

```
class Solution {
func pivotIndex(_ nums: [Int]) -> Int {

}
}
```

### Rust Solution:

```
// Problem: Find Pivot Index
// Difficulty: Easy
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn pivot_index(nums: Vec<i32>) -> i32 {

}
}
```

### Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def pivot_index(nums)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function pivotIndex($nums) {

}
}
```

### Dart Solution:

```
class Solution {
int pivotIndex(List<int> nums) {

}
}
```

### Scala Solution:

```
object Solution {
def pivotIndex(nums: Array[Int]): Int = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec pivot_index(nums :: [integer]) :: integer
def pivot_index(nums) do

end
end
```

### Erlang Solution:

```
-spec pivot_index(Nums :: [integer()]) -> integer().
pivot_index(Nums) ->
.
```

**Racket Solution:**

```
(define/contract (pivot-index nums)
(-> (listof exact-integer?) exact-integer?)
)
```