Problem 167: Two Sum II - Input Array Is Sorted

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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[index

1

]

and

numbers[index

2

]

where

1 <= index

1

< index

2

<= numbers.length

.

Return

the indices of the two numbers,

index

1

and

index

2

,

added by one

as an integer array

[index

1

, index

2

]

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, index

1

= 1, index

2

= 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 index

1

= 1, index

2

= 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 index

1

= 1, index

2

= 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

The tests are generated such that there is

exactly one solution

.

Code Snippets

C++:

```
class Solution {
public:
vector<int> twoSum(vector<int>& numbers, int target) {

}
};
```

Java:

```
class Solution {
public int[] twoSum(int[] numbers, int target) {

}
```

```
}
```

Python3:

```
class Solution:
def twoSum(self, numbers: List[int], target: int) -> List[int]:
```

Python:

```
class Solution(object):
def twoSum(self, numbers, target):
"""
:type numbers: List[int]
:type target: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} numbers
* @param {number} target
* @return {number[]}
*/
var twoSum = function(numbers, target) {

};
```

TypeScript:

```
function twoSum(numbers: number[], target: number): number[] {

};
```

C#:

```
public class Solution {
public int[] TwoSum(int[] numbers, int target) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* twoSum(int* numbers, int numbersSize, int target, int* returnSize) {

}
```

Go:

```
func twoSum(numbers []int, target int) []int {

}
```

Kotlin:

```
class Solution {
fun twoSum(numbers: IntArray, target: Int): IntArray {

}
}
```

Swift:

```
class Solution {
func twoSum(_ numbers: [Int], _ target: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn two_sum(numbers: Vec<i32>, target: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} numbers
# @param {Integer} target
```

```
# @return {Integer[]}
def two_sum(numbers, target)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $numbers
* @param Integer $target
* @return Integer[]
*/
function twoSum($numbers, $target) {

}
}
```

Dart:

```
class Solution {
List<int> twoSum(List<int> numbers, int target) {

}
}
```

Scala:

```
object Solution {
def twoSum(numbers: Array[Int], target: Int): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec two_sum(numbers :: [integer], target :: integer) :: [integer]
def two_sum(numbers, target) do

end
```

```
end
```

Erlang:

```
-spec two_sum(Numbers :: [integer()], Target :: integer()) -> [integer()].
two_sum(Numbers, Target) ->
.
```

Racket:

```
(define/contract (two-sum numbers target)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Two Sum II - Input Array Is Sorted
* Difficulty: Medium
* Tags: array, sort, search
*
* 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:
vector<int> twoSum(vector<int>& numbers, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Two Sum II - Input Array Is Sorted
* Difficulty: Medium
```

```
* Tags: array, sort, search
*
* 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[] twoSum(int[] numbers, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Two Sum II - Input Array Is Sorted
Difficulty: Medium
Tags: array, sort, search

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 twoSum(self, numbers: List[int], target: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def twoSum(self, numbers, target):
"""
:type numbers: List[int]
:type target: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Two Sum II - Input Array Is Sorted
* Difficulty: Medium
* Tags: array, sort, search
*
* 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[]} numbers
* @param {number} target
* @return {number[]}
*/
var twoSum = function(numbers, target) {

};
```

TypeScript Solution:

```
/**
* Problem: Two Sum II - Input Array Is Sorted
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 twoSum(numbers: number[], target: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Two Sum II - Input Array Is Sorted
* Difficulty: Medium
* Tags: array, sort, search
*
```

```
* 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[] TwoSum(int[] numbers, int target) {

}
}
```

C Solution:

```
/*
* Problem: Two Sum II - Input Array Is Sorted
* Difficulty: Medium
* Tags: array, sort, search
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* twoSum(int* numbers, int numbersSize, int target, int* returnSize) {

}
```

Go Solution:

```
// Problem: Two Sum II - Input Array Is Sorted
// Difficulty: Medium
// Tags: array, sort, search
//
// 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 twoSum(numbers []int, target int) []int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun twoSum(numbers: IntArray, target: Int): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func twoSum(_ numbers: [Int], _ target: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Two Sum II - Input Array Is Sorted
// Difficulty: Medium
// Tags: array, sort, search
//
// 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 two_sum(numbers: Vec<i32>, target: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} numbers
# @param {Integer} target
# @return {Integer[]}
def two_sum(numbers, target)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $numbers
* @param Integer $target
* @return Integer[]
*/
function twoSum($numbers, $target) {

}
}
```

Dart Solution:

```
class Solution {
List<int> twoSum(List<int> numbers, int target) {

}
}
```

Scala Solution:

```
object Solution {
def twoSum(numbers: Array[Int], target: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec two_sum(numbers :: [integer], target :: integer) :: [integer]
def two_sum(numbers, target) do

end
end
```

Erlang Solution:

```
-spec two_sum(Numbers :: [integer()], Target :: integer()) -> [integer()].
two_sum(Numbers, Target) ->
.
```

Racket Solution:

```
(define/contract (two-sum numbers target)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```