Problem 2915: Length of the Longest Subsequence That Sums to Target

Problem Information

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

Problem Description

You are given a

0-indexed

array of integers

nums

, and an integer

target

.

Return

the

length of the longest subsequence

of

nums

that sums up to

target

.

If no such subsequence exists, return

-1

.

A

subsequence

is an array that can be derived from another array by deleting some or no elements without changing the order of the remaining elements.

Example 1:

Input:

nums = [1,2,3,4,5], target = 9

Output:

3

Explanation:

There are 3 subsequences with a sum equal to 9: [4,5], [1,3,5], and [2,3,4]. The longest subsequences are [1,3,5], and [2,3,4]. Hence, the answer is 3.

Example 2:

Input:

nums = [4,1,3,2,1,5], target = 7

Output:

4

Explanation:

There are 5 subsequences with a sum equal to 7: [4,3], [4,1,2], [4,2,1], [1,1,5], and [1,3,2,1]. The longest subsequence is [1,3,2,1]. Hence, the answer is 4.

Example 3:

Input:

nums = [1,1,5,4,5], target = 3

Output:

-1

Explanation:

It can be shown that nums has no subsequence that sums up to 3.

Constraints:

1 <= nums.length <= 1000

1 <= nums[i] <= 1000

1 <= target <= 1000

Code Snippets

C++:

```
class Solution {
public:
int lengthOfLongestSubsequence(vector<int>& nums, int target) {

}
```

```
};
```

Java:

```
class Solution {
public int lengthOfLongestSubsequence(List<Integer> nums, int target) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function lengthOfLongestSubsequence(nums: number[], target: number): number {

};
```

C#:

```
public class Solution {
public int LengthOfLongestSubsequence(IList<int> nums, int target) {

}
}
```

C:

```
int lengthOfLongestSubsequence(int* nums, int numsSize, int target) {

}
```

Go:

```
func lengthOfLongestSubsequence(nums []int, target int) int {

}
```

Kotlin:

```
class Solution {
fun lengthOfLongestSubsequence(nums: List<Int>, target: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} target
# @return {Integer}
def length_of_longest_subsequence(nums, target)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $target
* @return Integer
*/
function lengthOfLongestSubsequence($nums, $target) {

}
}
```

Dart:

```
class Solution {
int lengthOfLongestSubsequence(List<int> nums, int target) {

}
}
```

Scala:

```
object Solution {
def lengthOfLongestSubsequence(nums: List[Int], target: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec length_of_longest_subsequence(nums :: [integer], target :: integer) ::
```

```
integer
def length_of_longest_subsequence(nums, target) do

end
end
```

Erlang:

```
-spec length_of_longest_subsequence(Nums :: [integer()], Target :: integer())
-> integer().
length_of_longest_subsequence(Nums, Target) ->
.
```

Racket:

```
(define/contract (length-of-longest-subsequence nums target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Length of the Longest Subsequence That Sums to Target
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int lengthOfLongestSubsequence(vector<int>& nums, int target) {

}
};
```

Java Solution:

```
/**
* Problem: Length of the Longest Subsequence That Sums to Target
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int lengthOfLongestSubsequence(List<Integer> nums, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: Length of the Longest Subsequence That Sums to Target
Difficulty: Medium
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Length of the Longest Subsequence That Sums to Target
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Length of the Longest Subsequence That Sums to Target
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function lengthOfLongestSubsequence(nums: number[], target: number): number {

};
```

C# Solution:

```
/*
* Problem: Length of the Longest Subsequence That Sums to Target
* Difficulty: Medium
```

```
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int LengthOfLongestSubsequence(IList<int> nums, int target) {

}
}
```

C Solution:

```
/*
* Problem: Length of the Longest Subsequence That Sums to Target
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int lengthOfLongestSubsequence(int* nums, int numsSize, int target) {

}
```

Go Solution:

```
// Problem: Length of the Longest Subsequence That Sums to Target
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func lengthOfLongestSubsequence(nums []int, target int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun lengthOfLongestSubsequence(nums: List<Int>, target: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Length of the Longest Subsequence That Sums to Target
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn length_of_longest_subsequence(nums: Vec<i32>, target: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} target
# @return {Integer}
def length_of_longest_subsequence(nums, target)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $target
* @return Integer
*/
function lengthOfLongestSubsequence($nums, $target) {

}
}
```

Dart Solution:

```
class Solution {
int lengthOfLongestSubsequence(List<int> nums, int target) {

}
}
```

Scala Solution:

```
object Solution {
def lengthOfLongestSubsequence(nums: List[Int], target: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec length_of_longest_subsequence(nums :: [integer], target :: integer) ::
integer
def length_of_longest_subsequence(nums, target) do

end
end
```

Erlang Solution:

```
-spec length_of_longest_subsequence(Nums :: [integer()], Target :: integer())
-> integer().
length_of_longest_subsequence(Nums, Target) ->
.
```

Racket Solution:

```
(define/contract (length-of-longest-subsequence nums target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```