Problem 1755: Closest Subsequence Sum

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

goal

.

You want to choose a subsequence of

nums

such that the sum of its elements is the closest possible to

goal

. That is, if the sum of the subsequence's elements is

sum

, then you want to

minimize the absolute difference

abs(sum - goal)

.

Return

the

minimum

possible value of

abs(sum - goal)

.

Note that a subsequence of an array is an array formed by removing some elements

(possibly all or none)

of the original array.

Example 1:

Input:

nums = [5,-7,3,5], goal = 6

Output:

0

Explanation:

Choose the whole array as a subsequence, with a sum of 6. This is equal to the goal, so the absolute difference is 0.

Example 2:

Input:

nums = [7,-9,15,-2], goal = -5

Output:

1

Explanation:

Choose the subsequence [7,-9,-2], with a sum of -4. The absolute difference is abs(-4 - (-5)) = abs(1) = 1, which is the minimum.

Example 3:

Input:

nums = [1,2,3], goal = -7

Output:

7

Constraints:

1 <= nums.length <= 40

-10

7

<= nums[i] <= 10

7

-10

9

<= goal <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int minAbsDifference(vector<int>& nums, int goal) {

}
};
```

Java:

```
class Solution {
public int minAbsDifference(int[] nums, int goal) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minAbsDifference(nums: number[], goal: number): number {

};
```

C#:

```
public class Solution {
public int MinAbsDifference(int[] nums, int goal) {

}
}
```

C:

```
int minAbsDifference(int* nums, int numsSize, int goal) {

}
```

Go:

```
func minAbsDifference(nums []int, goal int) int {

}
```

Kotlin:

```
class Solution {
fun minAbsDifference(nums: IntArray, goal: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} goal
# @return {Integer}
def min_abs_difference(nums, goal)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $goal
* @return Integer
*/
function minAbsDifference($nums, $goal) {

}
}
```

Dart:

```
class Solution {
int minAbsDifference(List<int> nums, int goal) {
```

```
}
}
```

Scala:

```
object Solution {
def minAbsDifference(nums: Array[Int], goal: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_abs_difference(nums :: [integer], goal :: integer) :: integer
def min_abs_difference(nums, goal) do

end
end
```

Erlang:

```
-spec min_abs_difference(Nums :: [integer()], Goal :: integer()) ->
integer().
min_abs_difference(Nums, Goal) ->
.
```

Racket:

```
(define/contract (min-abs-difference nums goal)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Closest Subsequence Sum
```

```
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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 minAbsDifference(vector<int>& nums, int goal) {

}
};
```

Java Solution:

```
/**
* Problem: Closest Subsequence Sum
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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 minAbsDifference(int[] nums, int goal) {

}
}
```

Python3 Solution:

```
"""
Problem: Closest Subsequence Sum
Difficulty: Hard
Tags: array, dp, sort

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 minAbsDifference(self, nums: List[int], goal: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Closest Subsequence Sum
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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} goal
* @return {number}
*/
var minAbsDifference = function(nums, goal) {

};
```

TypeScript Solution:

```
/**
* Problem: Closest Subsequence Sum
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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 minAbsDifference(nums: number[], goal: number): number {

};
```

C# Solution:

```
/*
* Problem: Closest Subsequence Sum
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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 MinAbsDifference(int[] nums, int goal) {

}
}
```

C Solution:

```
/*
* Problem: Closest Subsequence Sum
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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 minAbsDifference(int* nums, int numsSize, int goal) {

}
```

Go Solution:

```
// Problem: Closest Subsequence Sum
// Difficulty: Hard
// Tags: array, dp, sort
//
// 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 minAbsDifference(nums []int, goal int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minAbsDifference(nums: IntArray, goal: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Closest Subsequence Sum
// Difficulty: Hard
// Tags: array, dp, sort
```

```
//
// 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 min_abs_difference(nums: Vec<i32>, goal: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} goal
# @return {Integer}
def min_abs_difference(nums, goal)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $goal
* @return Integer
*/
function minAbsDifference($nums, $goal) {

}
}
```

Dart Solution:

```
class Solution {
int minAbsDifference(List<int> nums, int goal) {

}
}
```

Scala Solution:

```
object Solution {
def minAbsDifference(nums: Array[Int], goal: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_abs_difference(nums :: [integer], goal :: integer) :: integer
def min_abs_difference(nums, goal) do

end
end
```

Erlang Solution:

```
-spec min_abs_difference(Nums :: [integer()], Goal :: integer()) ->
integer().
min_abs_difference(Nums, Goal) ->
.
```

Racket Solution:

```
(define/contract (min-abs-difference nums goal)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```