Problem 16: 3Sum Closest

Problem Information

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

Problem Description

Given an integer array

nums

of length

n

and an integer

target

, find three integers in

nums

such that the sum is closest to

target

.

Return

the sum of the three integers

.

You may assume that each input would have exactly one solution.

Example 1:

Input:

nums = [-1,2,1,-4], target = 1

Output:

2

Explanation:

The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

Example 2:

Input:

nums = [0,0,0], target = 1

Output:

0

Explanation:

The sum that is closest to the target is 0. (0 + 0 + 0 = 0).

Constraints:

3 <= nums.length <= 500

-1000 <= nums[i] <= 1000

-10

4

<= target <= 10

4

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int threeSumClosest(int[] nums, int target) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int ThreeSumClosest(int[] nums, int target) {

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun threeSumClosest(nums: IntArray, target: Int): Int {

}
```

```
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

```
object Solution {
def threeSumClosest(nums: Array[Int], target: Int): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (three-sum-closest nums target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: 3Sum Closest
* Difficulty: Medium
* Tags: array, sort
*
* 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 threeSumClosest(vector<int>& nums, int target) {

}
};
```

Java Solution:

```
/**
* Problem: 3Sum Closest
* Difficulty: Medium
* Tags: array, sort
*
* 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 threeSumClosest(int[] nums, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: 3Sum Closest
Difficulty: Medium
Tags: array, sort
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: 3Sum Closest
* Difficulty: Medium
* Tags: array, sort
*
* 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
* @param {number} target
* @return {number}
*/
var threeSumClosest = function(nums, target) {

};
```

TypeScript Solution:

```
/**
* Problem: 3Sum Closest
* Difficulty: Medium
* Tags: array, sort
*
* 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 threeSumClosest(nums: number[], target: number): number {

};
```

C# Solution:

```
/*
* Problem: 3Sum Closest
* Difficulty: Medium
* Tags: array, sort
*
* 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 ThreeSumClosest(int[] nums, int target) {

}
}
```

C Solution:

```
/*
* Problem: 3Sum Closest
* Difficulty: Medium
* Tags: array, sort
*
* 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 threeSumClosest(int* nums, int numsSize, int target) {

}
```

Go Solution:

```
// Problem: 3Sum Closest
// Difficulty: Medium
// Tags: array, sort
//
// 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 threeSumClosest(nums []int, target int) int {

}
```

Kotlin Solution:

```
class Solution {
fun threeSumClosest(nums: IntArray, target: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: 3Sum Closest
// Difficulty: Medium
```

```
// Tags: array, sort
//
// 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 three_sum_closest(nums: Vec<i32>, target: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
```

```
}
```

Scala Solution:

```
object Solution {
def threeSumClosest(nums: Array[Int], target: Int): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (three-sum-closest nums target)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```