Problem 18: 4Sum

Problem Information

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

Problem Description

Given an array

nums

of

n

integers, return

an array of all the

unique

quadruplets

[nums[a], nums[b], nums[c], nums[d]]

such that:

0 <= a, b, c, d < n

a

,

b

,

c

, and

d

are

distinct

.

nums[a] + nums[b] + nums[c] + nums[d] == target

You may return the answer in

any order

.

Example 1:

Input:

nums = [1,0,-1,0,-2,2], target = 0

Output:

[[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]

Example 2:

Input:

nums = [2,2,2,2,2], target = 8

Output:

[[2,2,2,2]]

Constraints:

1 <= nums.length <= 200

-10

9

<= nums[i] <= 10

9

-10

9

<= target <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<List<Integer>> fourSum(int[] nums, int target) {
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public IList<IList<int>> FourSum(int[] nums, int target) {

}

}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** fourSum(int* nums, int numsSize, int target, int* returnSize, int**
returnColumnSizes) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: 4Sum
* 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:
vector<vector<int>> fourSum(vector<int>& nums, int target) {

}
};
```

Java Solution:

```
/**
* Problem: 4Sum
* 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 List<List<Integer>> fourSum(int[] nums, int target) {

}
}
```

Python3 Solution:

```
"""
Problem: 4Sum
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 fourSum(self, nums: List[int], target: int) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: 4Sum
* 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 fourSum = function(nums, target) {

};
```

TypeScript Solution:

```
/**
* Problem: 4Sum
* 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 fourSum(nums: number[], target: number): number[][] {

};
```

C# Solution:

```
/*
* Problem: 4Sum
* 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 IList<IList<int>> FourSum(int[] nums, int target) {

}
}
```

C Solution:

```
/*
* Problem: 4Sum
* 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** fourSum(int* nums, int numsSize, int target, int* returnSize, int**
returnColumnSizes) {

}
```

Go Solution:

```
// Problem: 4Sum
// 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 fourSum(nums []int, target int) [][]int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: 4Sum
// 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 four_sum(nums: Vec<i32>, target: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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