Problem 15: 3Sum

Problem Information

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

Problem Description

Given an integer array nums, return all the triplets

[nums[i], nums[j], nums[k]]

such that

i != j

,

i != k

, and

j != k

, and

nums[i] + nums[j] + nums[k] == 0

.

Notice that the solution set must not contain duplicate triplets.

Example 1:

Input:

nums = [-1,0,1,2,-1,-4]

Output:

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

Explanation:

nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0. nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0. nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0. The distinct triplets are [-1,0,1] and [-1,-1,2]. Notice that the order of the output and the order of the triplets does not matter.

Example 2:

Input:

nums = [0,1,1]

Output:

[]

Explanation:

The only possible triplet does not sum up to 0.

Example 3:

Input:

nums = [0,0,0]

Output:

[[0,0,0]]

Explanation:

The only possible triplet sums up to 0.

Constraints:

3 <= nums.length <= 3000

-10

5

<= nums[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> threeSum(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public List<List<Integer>> threeSum(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function threeSum(nums: number[]): number[][] {

};
```

C#:

```
public class Solution {
public IList<IList<int>> ThreeSum(int[] nums) {

}
}
```

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** threeSum(int* nums, int numsSize, int* returnSize, int**
returnColumnSizes) {

}
```

Go:

```
func threeSum(nums []int) [][]int {

}
```

Kotlin:

```
class Solution {
fun threeSum(nums: IntArray): List<List<Int>> {

}
}
```

Swift:

```
class Solution {
func threeSum(_ nums: [Int]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn three_sum(nums: Vec<i32>) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer[][]}
def three_sum(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer[][]
*/
function threeSum($nums) {

}
}
```

Dart:

```
class Solution {
List<List<int>> threeSum(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def threeSum(nums: Array[Int]): List[List[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec three_sum(nums :: [integer]) :: [[integer]]
def three_sum(nums) do

end
end
```

Erlang:

```
-spec three_sum(Nums :: [integer()]) -> [[integer()]].
three_sum(Nums) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: 3Sum
* 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>> threeSum(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: 3Sum
* 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>> threeSum(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: 3Sum
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 threeSum(self, nums: List[int]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: 3Sum
* 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
* @return {number[][]}
*/
var threeSum = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: 3Sum
* 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 threeSum(nums: number[]): number[][] {

};
```

C# Solution:

```
/*
* Problem: 3Sum
* 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>> ThreeSum(int[] nums) {

}
}
```

C Solution:

```c
/*
* Problem: 3Sum
* 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** threeSum(int* nums, int numsSize, int* returnSize, int**
returnColumnSizes) {

}
```

Go Solution:

```go
// Problem: 3Sum
// 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 threeSum(nums []int) [][]int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun threeSum(nums: IntArray): List<List<Int>> {

}
}
```

Swift Solution:

```
class Solution {
func threeSum(_ nums: [Int]) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: 3Sum
// 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(nums: Vec<i32>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer[][]}
def three_sum(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer[][]
*/
function threeSum($nums) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> threeSum(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def threeSum(nums: Array[Int]): List[List[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec three_sum(nums :: [integer]) :: [[integer]]
def three_sum(nums) do

end
end
```

Erlang Solution:

```
-spec three_sum(Nums :: [integer()]) -> [[integer()]].
three_sum(Nums) ->
.
```

Racket Solution:

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