Problem 1679: Max Number of K-Sum Pairs

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

k

.

In one operation, you can pick two numbers from the array whose sum equals

k

and remove them from the array.

Return

the maximum number of operations you can perform on the array

.

Example 1:

Input:

nums = [1,2,3,4], k = 5

Output:

2

Explanation:

Starting with nums = [1,2,3,4]: - Remove numbers 1 and 4, then nums = [2,3] - Remove numbers 2 and 3, then nums = [] There are no more pairs that sum up to 5, hence a total of 2 operations.

Example 2:

Input:

nums = [3,1,3,4,3], k = 6

Output:

1

Explanation:

Starting with nums = [3,1,3,4,3]: - Remove the first two 3's, then nums = [1,4,3] There are no more pairs that sum up to 6, hence a total of 1 operation.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

1 <= k <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int maxOperations(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public int maxOperations(int[] nums, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxOperations(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MaxOperations(int[] nums, int k) {

}
}
```

C:

```
int maxOperations(int* nums, int numsSize, int k){

}
```

Go:

```
func maxOperations(nums []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun maxOperations(nums: IntArray, k: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def max_operations(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function maxOperations($nums, $k) {

}
}
```

Scala:

```
object Solution {
def maxOperations(nums: Array[Int], k: Int): Int = {
}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Max Number of K-Sum Pairs
* Difficulty: Medium
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int maxOperations(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Max Number of K-Sum Pairs
* Difficulty: Medium
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int maxOperations(int[] nums, int k) {
}
}
```

Python3 Solution:

```
"""
Problem: Max Number of K-Sum Pairs
Difficulty: Medium
Tags: array, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Max Number of K-Sum Pairs
* Difficulty: Medium
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Max Number of K-Sum Pairs
* Difficulty: Medium
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function maxOperations(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Max Number of K-Sum Pairs
* Difficulty: Medium
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int MaxOperations(int[] nums, int k) {
}
}
```

C Solution:

```c
/*
* Problem: Max Number of K-Sum Pairs
* Difficulty: Medium
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/



int maxOperations(int* nums, int numsSize, int k){

}
```

Go Solution:

```go
// Problem: Max Number of K-Sum Pairs
// Difficulty: Medium
// Tags: array, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func maxOperations(nums []int, k int) int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun maxOperations(nums: IntArray, k: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Max Number of K-Sum Pairs
// Difficulty: Medium
// Tags: array, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn max_operations(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def max_operations(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function maxOperations($nums, $k) {

}
}
```

Scala Solution:

```
object Solution {
def maxOperations(nums: Array[Int], k: Int): Int = {

}
}
```