Problem 2597: The Number of Beautiful Subsets

Problem Information

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

Problem Description

You are given an array

nums

of positive integers and a

positive

integer

k

.

A subset of

nums

is

beautiful

if it does not contain two integers with an absolute difference equal to

k

.

Return

the number of

non-empty beautiful

subsets of the array

nums

.

A

subset

of

nums

is an array that can be obtained by deleting some (possibly none) elements from

nums

. Two subsets are different if and only if the chosen indices to delete are different.

Example 1:

Input:

nums = [2,4,6], k = 2

Output:

4

Explanation:

The beautiful subsets of the array nums are: [2], [4], [6], [2, 6]. It can be proved that there are only 4 beautiful subsets in the array [2,4,6].

Example 2:

Input:

nums = [1], k = 1

Output:

1

Explanation:

The beautiful subset of the array nums is [1]. It can be proved that there is only 1 beautiful subset in the array [1].

Constraints:

1 <= nums.length <= 18

1 <= nums[i], k <= 1000

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

```
}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int beautifulSubsets(List<int> nums, int k) {

}
}
```

Scala:

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

}
}
```

Elixir:

```
defmodule Solution do
@spec beautiful_subsets(nums :: [integer], k :: integer) :: integer
def beautiful_subsets(nums, k) do
```

```
end
end
```

Erlang:

```
-spec beautiful_subsets(Nums :: [integer()], K :: integer()) -> integer().
beautiful_subsets(Nums, K) ->
.
```

Racket:

```
(define/contract (beautiful-subsets nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: The Number of Beautiful Subsets
* Difficulty: Medium
* Tags: array, dp, math, hash, 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 beautifulSubsets(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: The Number of Beautiful Subsets
```

```
* Difficulty: Medium
* Tags: array, dp, math, hash, 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 beautifulSubsets(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: The Number of Beautiful Subsets
Difficulty: Medium
Tags: array, dp, math, hash, 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 beautifulSubsets(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: The Number of Beautiful Subsets
* Difficulty: Medium
* Tags: array, dp, math, hash, 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} k
* @return {number}
*/
var beautifulSubsets = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: The Number of Beautiful Subsets
* Difficulty: Medium
* Tags: array, dp, math, hash, 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 beautifulSubsets(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: The Number of Beautiful Subsets
* Difficulty: Medium
* Tags: array, dp, math, hash, 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 BeautifulSubsets(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: The Number of Beautiful Subsets
* Difficulty: Medium
* Tags: array, dp, math, hash, 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 beautifulSubsets(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: The Number of Beautiful Subsets
// Difficulty: Medium
// Tags: array, dp, math, hash, 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 beautifulSubsets(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: The Number of Beautiful Subsets
// Difficulty: Medium
// Tags: array, dp, math, hash, 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 beautiful_subsets(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int beautifulSubsets(List<int> nums, int k) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec beautiful_subsets(nums :: [integer], k :: integer) :: integer
def beautiful_subsets(nums, k) do

end
end
```

Erlang Solution:

```
-spec beautiful_subsets(Nums :: [integer()], K :: integer()) -> integer().
beautiful_subsets(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (beautiful-subsets nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```