Problem 1426: Counting Elements

Problem Information

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

Problem Description

Given an integer array

arr

, count how many elements

x

there are, such that

x + 1

is also in

arr

. If there are duplicates in

arr

, count them separately.

Example 1:

Input:

arr = [1,2,3]

Output:

2

Explanation:

1 and 2 are counted cause 2 and 3 are in arr.

Example 2:

Input:

arr = [1,1,3,3,5,5,7,7]

Output:

0

Explanation:

No numbers are counted, cause there is no 2, 4, 6, or 8 in arr.

Constraints:

1 <= arr.length <= 1000

0 <= arr[i] <= 1000

Code Snippets

C++:

```
class Solution {
public:
int countElements(vector<int>& arr) {
}
};
```

Java:

```
class Solution {
public int countElements(int[] arr) {

}
}
```

Python3:

```
class Solution:
def countElements(self, arr: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function countElements(arr: number[]): number {

};
```

C#:

```
}
```

```
public class Solution {
public int CountElements(int[] arr) {

}
}
```

C:

```
int countElements(int* arr, int arrSize) {

}
```

Go:

```
func countElements(arr []int) int {

}
```

Kotlin:

```
class Solution {
fun countElements(arr: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func countElements(_ arr: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_elements(arr: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @return {Integer}
def count_elements(arr)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer
*/
function countElements($arr) {

}
}
```

Dart:

```
class Solution {
int countElements(List<int> arr) {

}
}
```

Scala:

```
object Solution {
def countElements(arr: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_elements(arr :: [integer]) :: integer
def count_elements(arr) do

end
end
```

Erlang:

```
-spec count_elements(Arr :: [integer()]) -> integer().
count_elements(Arr) ->
.
```

Racket:

```
(define/contract (count-elements arr)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Counting Elements
* Difficulty: Easy
* Tags: array, hash
*
* 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 countElements(vector<int>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Counting Elements
* Difficulty: Easy
* Tags: array, hash
*
* 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 countElements(int[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Counting Elements
Difficulty: Easy
Tags: array, hash

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 countElements(self, arr: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Counting Elements
* Difficulty: Easy
* Tags: array, hash
*
* 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[]} arr
* @return {number}
*/
var countElements = function(arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Counting Elements
* Difficulty: Easy
* Tags: array, hash
*
* 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 countElements(arr: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Counting Elements
* Difficulty: Easy
* Tags: array, hash
*
* 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 CountElements(int[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Counting Elements
* Difficulty: Easy
* Tags: array, hash
*
* 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 countElements(int* arr, int arrSize) {

}
```

Go Solution:

```
// Problem: Counting Elements
// Difficulty: Easy
// Tags: array, hash
//
// 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 countElements(arr []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countElements(arr: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func countElements(_ arr: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Counting Elements
// Difficulty: Easy
// Tags: array, hash
//
// 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 count_elements(arr: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @return {Integer}
def count_elements(arr)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $arr
* @return Integer
*/
function countElements($arr) {

}
}
```

Dart Solution:

```
class Solution {
int countElements(List<int> arr) {

}
}
```

Scala Solution:

```
object Solution {
def countElements(arr: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_elements(arr :: [integer]) :: integer
def count_elements(arr) do

end
end
```

Erlang Solution:

```
-spec count_elements(Arr :: [integer()]) -> integer().
count_elements(Arr) ->
.
```

Racket Solution:

```
(define/contract (count-elements arr)
(-> (listof exact-integer?) exact-integer?)
)
```