Problem 575: Distribute Candies

Problem Information

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

Problem Description

Alice has

n

candies, where the

i

th

candy is of type

candyType[i]

. Alice noticed that she started to gain weight, so she visited a doctor.

The doctor advised Alice to only eat

n / 2

of the candies she has (

n

is always even). Alice likes her candies very much, and she wants to eat the maximum number of different types of candies while still following the doctor's advice.

Given the integer array

candyType

of length

n

, return

the

maximum

number of different types of candies she can eat if she only eats

n / 2

of them

.

Example 1:

Input:

candyType = [1,1,2,2,3,3]

Output:

3

Explanation:

Alice can only eat 6 / 2 = 3 candies. Since there are only 3 types, she can eat one of each type.

Example 2:

Input:

candyType = [1,1,2,3]

Output:

2

Explanation:

Alice can only eat 4 / 2 = 2 candies. Whether she eats types [1,2], [1,3], or [2,3], she still can only eat 2 different types.

Example 3:

Input:

candyType = [6,6,6,6]

Output:

1

Explanation:

Alice can only eat 4 / 2 = 2 candies. Even though she can eat 2 candies, she only has 1 type.

Constraints:

n == candyType.length

2 <= n <= 10

4

n

is even.

-10

5

<= candyType[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int distributeCandies(vector<int>& candyType) {

}
};
```

Java:

```
class Solution {
public int distributeCandies(int[] candyType) {

}
}
```

Python3:

```
class Solution:
def distributeCandies(self, candyType: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function distributeCandies(candyType: number[]): number {

};
```

C#:

```
public class Solution {
public int DistributeCandies(int[] candyType) {

}
}
```

C:

```
int distributeCandies(int* candyType, int candyTypeSize) {

}
```

Go:

```
func distributeCandies(candyType []int) int {

}
```

Kotlin:

```
class Solution {
fun distributeCandies(candyType: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func distributeCandies(_ candyType: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn distribute_candies(candy_type: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} candy_type
# @return {Integer}
def distribute_candies(candy_type)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $candyType
* @return Integer
*/
function distributeCandies($candyType) {

}
}
```

Dart:

```
class Solution {
int distributeCandies(List<int> candyType) {

}
```

```
}
```

Scala:

```
object Solution {
def distributeCandies(candyType: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec distribute_candies(candy_type :: [integer]) :: integer
def distribute_candies(candy_type) do

end
end
```

Erlang:

```
-spec distribute_candies(CandyType :: [integer()]) -> integer().
distribute_candies(CandyType) ->
.
```

Racket:

```
(define/contract (distribute-candies candyType)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Distribute Candies
* 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 distributeCandies(vector<int>& candyType) {

}
};
```

Java Solution:

```
/**
* Problem: Distribute Candies
* 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 distributeCandies(int[] candyType) {

}
}
```

Python3 Solution:

```
"""
Problem: Distribute Candies
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 distributeCandies(self, candyType: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Distribute Candies
* 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[]} candyType
* @return {number}
*/
var distributeCandies = function(candyType) {

};
```

TypeScript Solution:

```
/**
* Problem: Distribute Candies
* 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 distributeCandies(candyType: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Distribute Candies
* 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 DistributeCandies(int[] candyType) {

}
}
```

C Solution:

```
/*
* Problem: Distribute Candies
* 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 distributeCandies(int* candyType, int candyTypeSize) {
```

```
}
```

Go Solution:

```
// Problem: Distribute Candies
// 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 distributeCandies(candyType []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun distributeCandies(candyType: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func distributeCandies(_ candyType: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Distribute Candies
// 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 distribute_candies(candy_type: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} candy_type
# @return {Integer}
def distribute_candies(candy_type)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $candyType
* @return Integer
*/
function distributeCandies($candyType) {

}
}
```

Dart Solution:

```
class Solution {
int distributeCandies(List<int> candyType) {

}
}
```

Scala Solution:

```
object Solution {
def distributeCandies(candyType: Array[Int]): Int = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec distribute_candies(candy_type :: [integer]) :: integer
def distribute_candies(candy_type) do

end
end
```

Erlang Solution:

```
-spec distribute_candies(CandyType :: [integer()]) -> integer().
distribute_candies(CandyType) ->
.
```

Racket Solution:

```
(define/contract (distribute-candies candyType)
(-> (listof exact-integer?) exact-integer?)
)
```