Problem 1103: Distribute Candies to People

Problem Information

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

Problem Description

We distribute some number of

candies

, to a row of

n = num_people

people in the following way:

We then give 1 candy to the first person, 2 candies to the second person, and so on until we give

n

candies to the last person.

Then, we go back to the start of the row, giving

n + 1

candies to the first person,

n + 2

candies to the second person, and so on until we give

2 * n

candies to the last person.

This process repeats (with us giving one more candy each time, and moving to the start of the row after we reach the end) until we run out of candies. The last person will receive all of our remaining candies (not necessarily one more than the previous gift).

Return an array (of length

num_people

and sum

candies

) that represents the final distribution of candies.

Example 1:

Input:

candies = 7, num_people = 4

Output:

[1,2,3,1]

Explanation:

On the first turn, ans[0] += 1, and the array is [1,0,0,0]. On the second turn, ans[1] += 2, and the array is [1,2,0,0]. On the third turn, ans[2] += 3, and the array is [1,2,3,0]. On the fourth turn, ans[3] += 1 (because there is only one candy left), and the final array is [1,2,3,1].

Example 2:

Input:

candies = 10, num_people = 3

Output:

[5,2,3]

Explanation:

On the first turn, ans[0] += 1, and the array is [1,0,0]. On the second turn, ans[1] += 2, and the array is [1,2,0]. On the third turn, ans[2] += 3, and the array is [1,2,3]. On the fourth turn, ans[0] += 4, and the final array is [5,2,3].

Constraints:

1 <= candies <= 10^9

1 <= num_people <= 1000

Code Snippets

C++:

```
class Solution {
public:
vector<int> distributeCandies(int candies, int num_people) {

}
};
```

Java:

```
class Solution {
public int[] distributeCandies(int candies, int num_people) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function distributeCandies(candies: number, num_people: number): number[] {

};
```

C#:

```
public class Solution {
public int[] DistributeCandies(int candies, int num_people) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* distributeCandies(int candies, int num_people, int* returnSize) {

}
```

Go:

```
func distributeCandies(candies int, num_people int) []int {

}
```

Kotlin:

```
class Solution {
fun distributeCandies(candies: Int, num_people: Int): IntArray {

}
}
```

Swift:

```
class Solution {
func distributeCandies(_ candies: Int, _ num_people: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn distribute_candies(candies: i32, num_people: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} candies
# @param {Integer} num_people
# @return {Integer[]}
def distribute_candies(candies, num_people)

end
```

PHP:

```
class Solution {
```

```
/**
* @param Integer $candies
* @param Integer $num_people
* @return Integer[]
*/
function distributeCandies($candies, $num_people) {

}
}
```

Dart:

```
class Solution {
List<int> distributeCandies(int candies, int num_people) {

}
}
```

Scala:

```
object Solution {
def distributeCandies(candies: Int, num_people: Int): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec distribute_candies(candies :: integer, num_people :: integer) ::
[integer]
def distribute_candies(candies, num_people) do

end
end
```

Erlang:

```
-spec distribute_candies(Candies :: integer(), Num_people :: integer()) ->
[integer()].
distribute_candies(Candies, Num_people) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Distribute Candies to People
* Difficulty: Easy
* Tags: array, math
*
* 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<int> distributeCandies(int candies, int num_people) {

}
};
```

Java Solution:

```
/**
* Problem: Distribute Candies to People
* Difficulty: Easy
* Tags: array, math
*
* 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 int[] distributeCandies(int candies, int num_people) {
}
}
```

Python3 Solution:

```
"""
Problem: Distribute Candies to People
Difficulty: Easy
Tags: array, math

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 distributeCandies(self, candies: int, num_people: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Distribute Candies to People
* Difficulty: Easy
* Tags: array, math
*
* 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} candies
* @param {number} num_people
* @return {number[]}
*/
var distributeCandies = function(candies, num_people) {

};
```

TypeScript Solution:

```
/**
* Problem: Distribute Candies to People
* Difficulty: Easy
* Tags: array, math
*
* 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 distributeCandies(candies: number, num_people: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Distribute Candies to People
* Difficulty: Easy
* Tags: array, math
*
* 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 int[] DistributeCandies(int candies, int num_people) {


}
}
```

C Solution:

```c
/*
* Problem: Distribute Candies to People
* Difficulty: Easy
* Tags: array, math
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* distributeCandies(int candies, int num_people, int* returnSize) {

}
```

Go Solution:

```go
// Problem: Distribute Candies to People
// Difficulty: Easy
// Tags: array, math
//
// 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 distributeCandies(candies int, num_people int) []int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun distributeCandies(candies: Int, num_people: Int): IntArray {
}
}
```

Swift Solution:

```
class Solution {
func distributeCandies(_ candies: Int, _ num_people: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Distribute Candies to People
// Difficulty: Easy
// Tags: array, math
//
// 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 distribute_candies(candies: i32, num_people: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} candies
# @param {Integer} num_people
# @return {Integer[]}
def distribute_candies(candies, num_people)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer $candies
* @param Integer $num_people
* @return Integer[]
*/
function distributeCandies($candies, $num_people) {

}
}
```

Dart Solution:

```
class Solution {
List<int> distributeCandies(int candies, int num_people) {

}
}
```

Scala Solution:

```
object Solution {
def distributeCandies(candies: Int, num_people: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec distribute_candies(candies :: integer, num_people :: integer) ::
[integer]
def distribute_candies(candies, num_people) do

end
end
```

Erlang Solution:

```
-spec distribute_candies(Candies :: integer(), Num_people :: integer()) ->
[integer()].
distribute_candies(Candies, Num_people) ->
```

```
.
```

Racket Solution:

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