Problem 1399: Count Largest Group

Problem Information

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

Problem Description

You are given an integer

n

.

We need to group the numbers from

1

to

n

according to the sum of its digits. For example, the numbers 14 and 5 belong to the

same

group, whereas 13 and 3 belong to

different

groups.

Return the number of groups that have the largest size, i.e. the

maximum

number of elements.

Example 1:

Input:

n = 13

Output:

4

Explanation:

There are 9 groups in total, they are grouped according sum of its digits of numbers from 1 to 13: [1,10], [2,11], [3,12], [4,13], [5], [6], [7], [8], [9]. There are 4 groups with largest size.

Example 2:

Input:

n = 2

Output:

2

Explanation:

There are 2 groups [1], [2] of size 1.

Constraints:

1 <= n <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int countLargestGroup(int n) {

}
};
```

Java:

```
class Solution {
public int countLargestGroup(int n) {

}
}
```

Python3:

```
class Solution:
def countLargestGroup(self, n: int) -> int:
```

Python:

```
class Solution(object):
def countLargestGroup(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {number}
*/
var countLargestGroup = function(n) {

};
```

TypeScript:

```
function countLargestGroup(n: number): number {

};
```

C#:

```
public class Solution {
public int CountLargestGroup(int n) {

}
}
```

C:

```
int countLargestGroup(int n) {

}
```

Go:

```
func countLargestGroup(n int) int {

}
```

Kotlin:

```
class Solution {
fun countLargestGroup(n: Int): Int {

}
}
```

Swift:

```
class Solution {
func countLargestGroup(_ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_largest_group(n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer}
def count_largest_group(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function countLargestGroup($n) {

}
}
```

Dart:

```
class Solution {
int countLargestGroup(int n) {

}
}
```

Scala:

```
object Solution {
def countLargestGroup(n: Int): Int = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec count_largest_group(n :: integer) :: integer
def count_largest_group(n) do

end
end
```

Erlang:

```
-spec count_largest_group(N :: integer()) -> integer().
count_largest_group(N) ->
.
```

Racket:

```
(define/contract (count-largest-group n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Largest Group
* Difficulty: Easy
* Tags: math, hash
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int countLargestGroup(int n) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Count Largest Group
* Difficulty: Easy
* Tags: math, hash
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

class Solution {
public int countLargestGroup(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Largest Group
Difficulty: Easy
Tags: math, hash

Approach: Use hash map for O(1) lookups
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(n) for hash map
"""

class Solution:
def countLargestGroup(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countLargestGroup(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Largest Group
* Difficulty: Easy
* Tags: math, hash
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

/**
* @param {number} n
* @return {number}
*/
var countLargestGroup = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Largest Group
* Difficulty: Easy
* Tags: math, hash
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

function countLargestGroup(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Count Largest Group
* Difficulty: Easy
* Tags: math, hash
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int CountLargestGroup(int n) {

}
}
```

C Solution:

```
/*
* Problem: Count Largest Group
* Difficulty: Easy
* Tags: math, hash
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

int countLargestGroup(int n) {

}
```

Go Solution:

```
// Problem: Count Largest Group
// Difficulty: Easy
// Tags: math, hash
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
```

```
// Space Complexity: O(n) for hash map

func countLargestGroup(n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countLargestGroup(n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func countLargestGroup(_ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Largest Group
// Difficulty: Easy
// Tags: math, hash
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

impl Solution {
pub fn count_largest_group(n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer}
def count_largest_group(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function countLargestGroup($n) {

}
}
```

Dart Solution:

```
class Solution {
int countLargestGroup(int n) {

}
}
```

Scala Solution:

```
object Solution {
def countLargestGroup(n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_largest_group(n :: integer) :: integer
def count_largest_group(n) do

end
```

```
end
```

Erlang Solution:

```
-spec count_largest_group(N :: integer()) -> integer().
count_largest_group(N) ->
.
```

Racket Solution:

```
(define/contract (count-largest-group n)
(-> exact-integer? exact-integer?)
)
```