# Problem 2243: Calculate Digit Sum of a String

## Problem Information

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

## Problem Description

You are given a string

s

consisting of digits and an integer

k

.

A

round

can be completed if the length of

s

is greater than

k

. In one round, do the following:

Divide

s

into

consecutive groups

of size

k

such that the first

k

characters are in the first group, the next

k

characters are in the second group, and so on.

Note

that the size of the last group can be smaller than

k

.

Replace

each group of

s

with a string representing the sum of all its digits. For example,

"346"

is replaced with

"13"

because

3 + 4 + 6 = 13

.

Merge

consecutive groups together to form a new string. If the length of the string is greater than

k

, repeat from step

1

.

Return

s

after all rounds have been completed

.

Example 1:

Input:

s = "11111222223", k = 3

Output:

"135"

Explanation:

- For the first round, we divide s into groups of size 3: "111", "112", "222", and "23". Then we calculate the digit sum of each group: 1 + 1 + 1 = 3, 1 + 1 + 2 = 4, 2 + 2 + 2 = 6, and 2 + 3 = 5. So, s becomes "3" + "4" + "6" + "5" = "3465" after the first round. - For the second round, we divide s into "346" and "5". Then we calculate the digit sum of each group: 3 + 4 + 6 = 13, 5 = 5. So, s becomes "13" + "5" = "135" after second round. Now, s.length <= k, so we return "135" as the answer.

Example 2:

Input:

s = "00000000", k = 3

Output:

"000"

Explanation:

We divide s into "000", "000", and "00". Then we calculate the digit sum of each group: 0 + 0 + 0 = 0, 0 + 0 + 0 = 0, and 0 + 0 = 0. s becomes "0" + "0" + "0" = "000", whose length is equal to k, so we return "000".

Constraints:

1 <= s.length <= 100

2 <= k <= 100

s

consists of digits only.

## Code Snippets

**C++:**

```
class Solution {
public:
string digitSum(string s, int k) {

}
};
```

**Java:**

```
class Solution {
public String digitSum(String s, int k) {

}
}
```

**Python3:**

```
class Solution:
def digitSum(self, s: str, k: int) -> str:
```

**Python:**

```
class Solution(object):
def digitSum(self, s, k):
"""
:type s: str
:type k: int
:rtype: str
"""
```

**JavaScript:**

```
/**
* @param {string} s
* @param {number} k
* @return {string}
*/
var digitSum = function(s, k) {

};
```

**TypeScript:**

```
function digitSum(s: string, k: number): string {

};
```

**C#:**

```
public class Solution {
public string DigitSum(string s, int k) {

}
}
```

**C:**

```
char* digitSum(char* s, int k) {

}
```

**Go:**

```
func digitSum(s string, k int) string {

}
```

**Kotlin:**

```
class Solution {
fun digitSum(s: String, k: Int): String {

}
}
```

**Swift:**

```
class Solution {
func digitSum(_ s: String, _ k: Int) -> String {

}
}
```

**Rust:**

```
impl Solution {
pub fn digit_sum(s: String, k: i32) -> String {

}
}
```

**Ruby:**

```
# @param {String} s
# @param {Integer} k
# @return {String}
def digit_sum(s, k)

end
```

**PHP:**

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return String
*/
function digitSum($s, $k) {

}
}
```

**Dart:**

```
class Solution {
String digitSum(String s, int k) {

}
}
```

**Scala:**

```
object Solution {
def digitSum(s: String, k: Int): String = {

}
```

```
}
```

**Elixir:**

```
defmodule Solution do
@spec digit_sum(s :: String.t, k :: integer) :: String.t
def digit_sum(s, k) do

end
end
```

**Erlang:**

```
-spec digit_sum(S :: unicode:unicode_binary(), K :: integer()) ->
unicode:unicode_binary().
digit_sum(S, K) ->
.
```

**Racket:**

```
(define/contract (digit-sum s k)
(-> string? exact-integer? string?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Calculate Digit Sum of a String
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
```

```
string digitSum(string s, int k) {

}
};
```

**Java Solution:**

```
/**
* Problem: Calculate Digit Sum of a String
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public String digitSum(String s, int k) {

}
}
```

**Python3 Solution:**

```
"""
Problem: Calculate Digit Sum of a String
Difficulty: Easy
Tags: string

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def digitSum(self, s: str, k: int) -> str:
# TODO: Implement optimized solution
pass
```

**Python Solution:**

```
class Solution(object):
def digitSum(self, s, k):
"""
:type s: str
:type k: int
:rtype: str
"""
```

### JavaScript Solution:

```
/**
* Problem: Calculate Digit Sum of a String
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string} s
* @param {number} k
* @return {string}
*/
var digitSum = function(s, k) {

};
```

### TypeScript Solution:

```
/**
* Problem: Calculate Digit Sum of a String
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function digitSum(s: string, k: number): string {
```

```
};
```

### C# Solution:

```
/*
* Problem: Calculate Digit Sum of a String
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public string DigitSum(string s, int k) {

}
}
```

### C Solution:

```
/*
* Problem: Calculate Digit Sum of a String
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

char* digitSum(char* s, int k) {

}
```

### Go Solution:

```
// Problem: Calculate Digit Sum of a String
// Difficulty: Easy
```

```
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func digitSum(s string, k int) string {

}
```

### Kotlin Solution:

```
class Solution {
fun digitSum(s: String, k: Int): String {

}
}
```

### Swift Solution:

```
class Solution {
func digitSum(_ s: String, _ k: Int) -> String {

}
}
```

### Rust Solution:

```
// Problem: Calculate Digit Sum of a String
// Difficulty: Easy
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn digit_sum(s: String, k: i32) -> String {

}
}
```

### Ruby Solution:

```
# @param {String} s
# @param {Integer} k
# @return {String}
def digit_sum(s, k)

end
```

### PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return String
*/
function digitSum($s, $k) {

}
}
```

### Dart Solution:

```
class Solution {
String digitSum(String s, int k) {

}
}
```

### Scala Solution:

```
object Solution {
def digitSum(s: String, k: Int): String = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec digit_sum(s :: String.t, k :: integer) :: String.t
def digit_sum(s, k) do

end
end
```

### Erlang Solution:

```
-spec digit_sum(S :: unicode:unicode_binary(), K :: integer()) ->
unicode:unicode_binary().
digit_sum(S, K) ->
.
```

### Racket Solution:

```
(define/contract (digit-sum s k)
(-> string? exact-integer? string?)
)
```