Problem 3697: Compute Decimal Representation

Problem Information

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

Problem Description

You are given a

positive

integer

n

.

A positive integer is a

base-10 component

if it is the product of a single digit from 1 to 9 and a non-negative power of 10. For example, 500, 30, and 7 are

base-10 components

, while 537, 102, and 11 are not.

Express

n

as a sum of

only

base-10 components, using the

fewest

base-10 components possible.

Return an array containing these

base-10 components

in

descending

order.

Example 1:

Input:

n = 537

Output:

[500,30,7]

Explanation:

We can express 537 as

500 + 30 + 7

. It is impossible to express 537 as a sum using fewer than 3 base-10 components.

Example 2:

Input:

n = 102

Output:

[100,2]

Explanation:

We can express 102 as

100 + 2

. 102 is not a base-10 component, which means 2 base-10 components are needed.

Example 3:

Input:

n = 6

Output:

[6]

Explanation:

6 is a base-10 component.

Constraints:

1 <= n <= 10

9

Code Snippets

C++:

```
class Solution {
public:
vector<int> decimalRepresentation(int n) {

}
};
```

Java:

```
class Solution {
public int[] decimalRepresentation(int n) {

}
}
```

Python3:

```
class Solution:
def decimalRepresentation(self, n: int) -> List[int]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function decimalRepresentation(n: number): number[] {

};
```

C#:

```
public class Solution {
public int[] DecimalRepresentation(int n) {

}
}
```

C:

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

}
```

Go:

```
func decimalRepresentation(n int) []int {

}
```

Kotlin:

```
class Solution {
fun decimalRepresentation(n: Int): IntArray {

}
}
```

Swift:

```
class Solution {
func decimalRepresentation(_ n: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn decimal_representation(n: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer[]}
def decimal_representation(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer[]
*/
function decimalRepresentation($n) {

}
}
```

Dart:

```
class Solution {
List<int> decimalRepresentation(int n) {

}
}
```

Scala:

```
object Solution {
def decimalRepresentation(n: Int): Array[Int] = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec decimal_representation(n :: integer) :: [integer]
def decimal_representation(n) do

end
end
```

Erlang:

```
-spec decimal_representation(N :: integer()) -> [integer()].
decimal_representation(N) ->
.
```

Racket:

```
(define/contract (decimal-representation n)
(-> exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Compute Decimal Representation
* 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> decimalRepresentation(int n) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Compute Decimal Representation
* 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[] decimalRepresentation(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Compute Decimal Representation
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 decimalRepresentation(self, n: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Compute Decimal Representation
* 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} n
* @return {number[]}
*/
var decimalRepresentation = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Compute Decimal Representation
* 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 decimalRepresentation(n: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Compute Decimal Representation
* 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[] DecimalRepresentation(int n) {

}
}
```

C Solution:

```
/*
* Problem: Compute Decimal Representation
* 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* decimalRepresentation(int n, int* returnSize) {

}
```

Go Solution:

```
// Problem: Compute Decimal Representation
// 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 decimalRepresentation(n int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun decimalRepresentation(n: Int): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func decimalRepresentation(_ n: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Compute Decimal Representation
// 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 decimal_representation(n: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer[]}
def decimal_representation(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer[]
*/
function decimalRepresentation($n) {

}
}
```

Dart Solution:

```
class Solution {
List<int> decimalRepresentation(int n) {

}
}
```

Scala Solution:

```
object Solution {
def decimalRepresentation(n: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec decimal_representation(n :: integer) :: [integer]
def decimal_representation(n) do
```

```
end
end
```

Erlang Solution:

```
-spec decimal_representation(N :: integer()) -> [integer()].
decimal_representation(N) ->
.
```

Racket Solution:

```
(define/contract (decimal-representation n)
(-> exact-integer? (listof exact-integer?))
)
```