Problem 66: Plus One

Problem Information

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

Problem Description

You are given a

large integer

represented as an integer array

digits

, where each

digits[i]

is the

i

th

digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading

0

's.

Increment the large integer by one and return

the resulting array of digits

.

Example 1:

Input:

digits = [1,2,3]

Output:

[1,2,4]

Explanation:

The array represents the integer 123. Incrementing by one gives 123 + 1 = 124. Thus, the result should be [1,2,4].

Example 2:

Input:

digits = [4,3,2,1]

Output:

[4,3,2,2]

Explanation:

The array represents the integer 4321. Incrementing by one gives 4321 + 1 = 4322. Thus, the result should be [4,3,2,2].

Example 3:

Input:

digits = [9]

Output:

[1,0]

Explanation:

The array represents the integer 9. Incrementing by one gives 9 + 1 = 10. Thus, the result should be [1,0].

Constraints:

1 <= digits.length <= 100

0 <= digits[i] <= 9

digits

does not contain any leading

0

's.

Code Snippets

C++:

```
class Solution {
public:
vector<int> plusOne(vector<int>& digits) {

}
};
```

Java:

```
class Solution {
public int[] plusOne(int[] digits) {
```

```
}
}
```

Python3:

```
class Solution:
def plusOne(self, digits: List[int]) -> List[int]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function plusOne(digits: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] PlusOne(int[] digits) {

}
}
```

C:

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

}
```

Go:

```
func plusOne(digits []int) []int {

}
```

Kotlin:

```
class Solution {
fun plusOne(digits: IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func plusOne(_ digits: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn plus_one(digits: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} digits
# @return {Integer[]}
def plus_one(digits)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer[] $digits
* @return Integer[]
*/
function plusOne($digits) {

}
}
```

Dart:

```
class Solution {
List<int> plusOne(List<int> digits) {

}
}
```

Scala:

```
object Solution {
def plusOne(digits: Array[Int]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec plus_one(digits :: [integer]) :: [integer]
def plus_one(digits) do

end
end
```

Erlang:

```
-spec plus_one(Digits :: [integer()]) -> [integer()].
plus_one(Digits) ->
.
```

Racket:

```
(define/contract (plus-one digits)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Plus One
* 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> plusOne(vector<int>& digits) {

}
};
```

Java Solution:

```
/**
* Problem: Plus One
* 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[] plusOne(int[] digits) {

}
}
```

Python3 Solution:

```
"""
Problem: Plus One
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 plusOne(self, digits: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Plus One
* 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[]} digits
* @return {number[]}
*/
var plusOne = function(digits) {

};
```

TypeScript Solution:

```
/**
* Problem: Plus One
* 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 plusOne(digits: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Plus One
* 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[] PlusOne(int[] digits) {

}
}
```

C Solution:

```
/*
* Problem: Plus One
* 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* plusOne(int* digits, int digitsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Plus One
// 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 plusOne(digits []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun plusOne(digits: IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func plusOne(_ digits: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Plus One
// 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 plus_one(digits: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} digits
# @return {Integer[]}
def plus_one(digits)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $digits
* @return Integer[]
*/
function plusOne($digits) {

}
}
```

Dart Solution:

```
class Solution {
List<int> plusOne(List<int> digits) {

}
}
```

Scala Solution:

```
object Solution {
def plusOne(digits: Array[Int]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec plus_one(digits :: [integer]) :: [integer]
def plus_one(digits) do

end
end
```

Erlang Solution:

```
-spec plus_one(Digits :: [integer()]) -> [integer()].
plus_one(Digits) ->
.
```

Racket Solution:

```
(define/contract (plus-one digits)
(-> (listof exact-integer?) (listof exact-integer?))
)
```