Problem 1860: Incremental Memory Leak

Problem Information

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

Problem Description

You are given two integers

memory1

and

memory2

representing the available memory in bits on two memory sticks. There is currently a faulty program running that consumes an increasing amount of memory every second.

At the

i

th

second (starting from 1),

i

bits of memory are allocated to the stick with

more available memory

(or from the first memory stick if both have the same available memory). If neither stick has at least

i

bits of available memory, the program

crashes

.

Return

an array containing

[crashTime, memory1

crash

, memory2

crash

]

, where

crashTime

is the time (in seconds) when the program crashed and

memory1

crash

and

memory2

crash

are the available bits of memory in the first and second sticks respectively

.

Example 1:

Input:

memory1 = 2, memory2 = 2

Output:

[3,1,0]

Explanation:

The memory is allocated as follows: - At the 1

st

second, 1 bit of memory is allocated to stick 1. The first stick now has 1 bit of available memory. - At the 2

nd

second, 2 bits of memory are allocated to stick 2. The second stick now has 0 bits of available memory. - At the 3

rd

second, the program crashes. The sticks have 1 and 0 bits available respectively.

Example 2:

Input:

memory1 = 8, memory2 = 11

Output:

[6,0,4]

Explanation:

The memory is allocated as follows: - At the 1

st

second, 1 bit of memory is allocated to stick 2. The second stick now has 10 bit of available memory. - At the 2

nd

second, 2 bits of memory are allocated to stick 2. The second stick now has 8 bits of available memory. - At the 3

rd

second, 3 bits of memory are allocated to stick 1. The first stick now has 5 bits of available memory. - At the 4

th

second, 4 bits of memory are allocated to stick 2. The second stick now has 4 bits of available memory. - At the 5

th

second, 5 bits of memory are allocated to stick 1. The first stick now has 0 bits of available memory. - At the 6

th

second, the program crashes. The sticks have 0 and 4 bits available respectively.

Constraints:

0 <= memory1, memory2 <= 2

31

- 1

Code Snippets

C++:

```
class Solution {
public:
vector<int> memLeak(int memory1, int memory2) {

}
};
```

Java:

```
class Solution {
public int[] memLeak(int memory1, int memory2) {

}
}
```

Python3:

```
class Solution:
def memLeak(self, memory1: int, memory2: int) -> List[int]:
```

Python:

```
class Solution(object):
def memLeak(self, memory1, memory2):
"""
:type memory1: int
:type memory2: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number} memory1
* @param {number} memory2
* @return {number[]}
*/
var memLeak = function(memory1, memory2) {

};
```

TypeScript:

```
function memLeak(memory1: number, memory2: number): number[] {

};
```

C#:

```
public class Solution {
public int[] MemLeak(int memory1, int memory2) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* memLeak(int memory1, int memory2, int* returnSize) {

}
```

Go:

```
func memLeak(memory1 int, memory2 int) []int {

}
```

Kotlin:

```
class Solution {
fun memLeak(memory1: Int, memory2: Int): IntArray {

}
}
```

Swift:

```
class Solution {
func memLeak(_ memory1: Int, _ memory2: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn mem_leak(memory1: i32, memory2: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer} memory1
# @param {Integer} memory2
# @return {Integer[]}
def mem_leak(memory1, memory2)

end
```

PHP:

```
class Solution {

/**
* @param Integer $memory1
* @param Integer $memory2
* @return Integer[]
*/
function memLeak($memory1, $memory2) {

}
}
```

Dart:

```
class Solution {
List<int> memLeak(int memory1, int memory2) {

}
}
```

Scala:

```
object Solution {
def memLeak(memory1: Int, memory2: Int): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec mem_leak(memory1 :: integer, memory2 :: integer) :: [integer]
def mem_leak(memory1, memory2) do

end
end
```

Erlang:

```
-spec mem_leak(Memory1 :: integer(), Memory2 :: integer()) -> [integer()].
mem_leak(Memory1, Memory2) ->
.
```

Racket:

```
(define/contract (mem-leak memory1 memory2)
(-> exact-integer? exact-integer? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Incremental Memory Leak
* Difficulty: Medium
* 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> memLeak(int memory1, int memory2) {

}
};
```

Java Solution:

```
/**
* Problem: Incremental Memory Leak
* Difficulty: Medium
* 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[] memLeak(int memory1, int memory2) {

}
}
```

Python3 Solution:

```
"""
Problem: Incremental Memory Leak
Difficulty: Medium
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 memLeak(self, memory1: int, memory2: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def memLeak(self, memory1, memory2):
"""
:type memory1: int
:type memory2: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Incremental Memory Leak
* Difficulty: Medium
* 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} memory1
* @param {number} memory2
* @return {number[]}
*/
var memLeak = function(memory1, memory2) {

};
```

TypeScript Solution:

```
/**
* Problem: Incremental Memory Leak
* Difficulty: Medium
* 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 memLeak(memory1: number, memory2: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Incremental Memory Leak
* Difficulty: Medium
* 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[] MemLeak(int memory1, int memory2) {

}
}
```

C Solution:

```
/*
* Problem: Incremental Memory Leak
* Difficulty: Medium
* 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* memLeak(int memory1, int memory2, int* returnSize) {

}
```

Go Solution:

```go
// Problem: Incremental Memory Leak
// Difficulty: Medium
// 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 memLeak(memory1 int, memory2 int) []int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun memLeak(memory1: Int, memory2: Int): IntArray {

}
}
```

Swift Solution:

```swift
class Solution {
func memLeak(_ memory1: Int, _ memory2: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Incremental Memory Leak
// Difficulty: Medium
// 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 mem_leak(memory1: i32, memory2: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer} memory1
# @param {Integer} memory2
# @return {Integer[]}
def mem_leak(memory1, memory2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $memory1
* @param Integer $memory2
* @return Integer[]
*/
function memLeak($memory1, $memory2) {

}
}
```

Dart Solution:

```
class Solution {
List<int> memLeak(int memory1, int memory2) {
}
}
```

Scala Solution:

```scala
object Solution {
def memLeak(memory1: Int, memory2: Int): Array[Int] = {

}
}
```

Elixir Solution:

```elixir
defmodule Solution do
@spec mem_leak(memory1 :: integer, memory2 :: integer) :: [integer]
def mem_leak(memory1, memory2) do

end
end
```

Erlang Solution:

```erlang
-spec mem_leak(Memory1 :: integer(), Memory2 :: integer()) -> [integer()].
mem_leak(Memory1, Memory2) ->
.
```

Racket Solution:

```racket
(define/contract (mem-leak memory1 memory2)
(-> exact-integer? exact-integer? (listof exact-integer?))
)
```