Problem 1606: Find Servers That Handled Most Number of Requests

Problem Information

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

Problem Description

You have

k

servers numbered from

0

to

k-1

that are being used to handle multiple requests simultaneously. Each server has infinite computational capacity but

cannot handle more than one request at a time

. The requests are assigned to servers according to a specific algorithm:

The

i

th

(0-indexed) request arrives.

If all servers are busy, the request is dropped (not handled at all).

If the

(i % k)

th

server is available, assign the request to that server.

Otherwise, assign the request to the next available server (wrapping around the list of servers and starting from 0 if necessary). For example, if the

i

th

server is busy, try to assign the request to the

(i+1)

th

server, then the

(i+2)

th

server, and so on.

You are given a

strictly increasing

array

arrival

of positive integers, where

arrival[i]

represents the arrival time of the

i

th

request, and another array

load

, where

load[i]

represents the load of the

i

th

request (the time it takes to complete). Your goal is to find the

busiest server(s)

. A server is considered

busiest

if it handled the most number of requests successfully among all the servers.

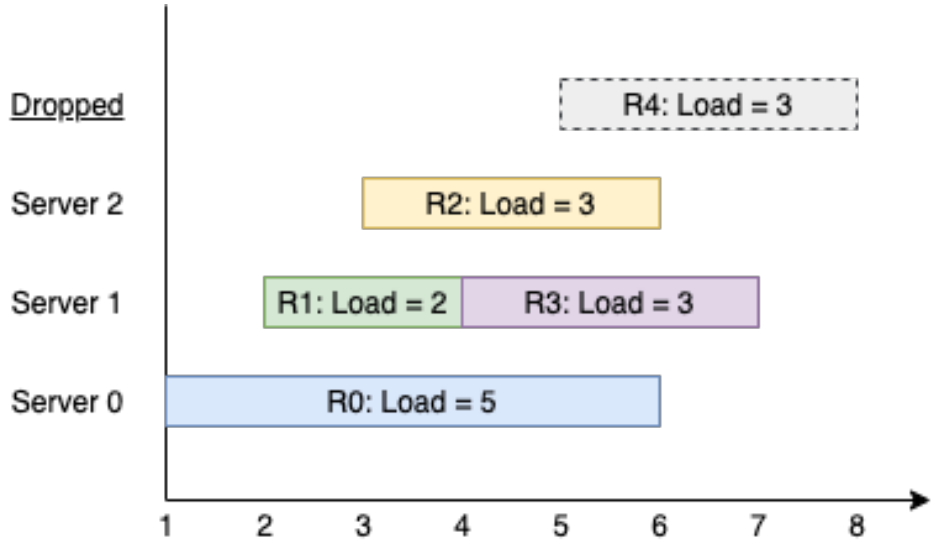
Return

a list containing the IDs (0-indexed) of the

busiest server(s)

. You may return the IDs in any order.

Example 1:



Input:

k = 3, arrival = [1,2,3,4,5], load = [5,2,3,3,3]

Output:

[1]

Explanation:

All of the servers start out available. The first 3 requests are handled by the first 3 servers in order. Request 3 comes in. Server 0 is busy, so it's assigned to the next available server, which is 1. Request 4 comes in. It cannot be handled since all servers are busy, so it is dropped. Servers 0 and 2 handled one request each, while server 1 handled two requests. Hence server 1 is the busiest server.

Example 2:

Input:

k = 3, arrival = [1,2,3,4], load = [1,2,1,2]

Output:

[0]

Explanation:

The first 3 requests are handled by first 3 servers. Request 3 comes in. It is handled by server 0 since the server is available. Server 0 handled two requests, while servers 1 and 2 handled one request each. Hence server 0 is the busiest server.

Example 3:

Input:

k = 3, arrival = [1,2,3], load = [10,12,11]

Output:

[0,1,2]

Explanation:

Each server handles a single request, so they are all considered the busiest.

Constraints:

1 <= k <= 10

5

1 <= arrival.length, load.length <= 10

5

arrival.length == load.length

1 <= arrival[i], load[i] <= 10

9

arrival

is

strictly increasing

.

Code Snippets

C++:

```
class Solution {
public:
vector<int> busiestServers(int k, vector<int>& arrival, vector<int>& load) {

}
};
```

Java:

```
class Solution {
public List<Integer> busiestServers(int k, int[] arrival, int[] load) {

}
}
```

Python3:

```
class Solution:
def busiestServers(self, k: int, arrival: List[int], load: List[int]) ->
List[int]:
```

Python:

```
class Solution(object):
def busiestServers(self, k, arrival, load):
"""
```

```
:type k: int
:type arrival: List[int]
:type load: List[int]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number} k
* @param {number[]} arrival
* @param {number[]} load
* @return {number[]}
*/
var busiestServers = function(k, arrival, load) {

};
```

TypeScript:

```
function busiestServers(k: number, arrival: number[], load: number[]):
number[] {

};
```

C#:

```
public class Solution {
public IList<int> BusiestServers(int k, int[] arrival, int[] load) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* busiestServers(int k, int* arrival, int arrivalSize, int* load, int
loadSize, int* returnSize) {

}
```

Go:

```
func busiestServers(k int, arrival []int, load []int) []int {

}
```

Kotlin:

```
class Solution {
fun busiestServers(k: Int, arrival: IntArray, load: IntArray): List<Int> {

}
}
```

Swift:

```
class Solution {
func busiestServers(_ k: Int, _ arrival: [Int], _ load: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn busiest_servers(k: i32, arrival: Vec<i32>, load: Vec<i32>) -> Vec<i32>
{

}
}
```

Ruby:

```
# @param {Integer} k
# @param {Integer[]} arrival
# @param {Integer[]} load
# @return {Integer[]}
def busiest_servers(k, arrival, load)

end
```

PHP:

```
class Solution {

/**
* @param Integer $k
* @param Integer[] $arrival
* @param Integer[] $load
* @return Integer[]
*/
function busiestServers($k, $arrival, $load) {

}
}
```

Dart:

```
class Solution {
List<int> busiestServers(int k, List<int> arrival, List<int> load) {

}
}
```

Scala:

```
object Solution {
def busiestServers(k: Int, arrival: Array[Int], load: Array[Int]): List[Int]
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec busiest_servers(k :: integer, arrival :: [integer], load :: [integer])
:: [integer]
def busiest_servers(k, arrival, load) do

end
end
```

Erlang:

```
-spec busiest_servers(K :: integer(), Arrival :: [integer()], Load ::
[integer()]) -> [integer()].
busiest_servers(K, Arrival, Load) ->
.
```

Racket:

```
(define/contract (busiest-servers k arrival load)
(-> exact-integer? (listof exact-integer?) (listof exact-integer?) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Servers That Handled Most Number of Requests
* Difficulty: Hard
* Tags: array, queue, heap
*
* 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> busiestServers(int k, vector<int>& arrival, vector<int>& load) {

}
};
```

Java Solution:

```
/**
* Problem: Find Servers That Handled Most Number of Requests
* Difficulty: Hard
* Tags: array, queue, heap
*
* 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 List<Integer> busiestServers(int k, int[] arrival, int[] load) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Servers That Handled Most Number of Requests
Difficulty: Hard
Tags: array, queue, heap

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 busiestServers(self, k: int, arrival: List[int], load: List[int]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def busiestServers(self, k, arrival, load):
"""
:type k: int
:type arrival: List[int]
:type load: List[int]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Find Servers That Handled Most Number of Requests
* Difficulty: Hard
* Tags: array, queue, heap
*
* 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} k
* @param {number[]} arrival
* @param {number[]} load
* @return {number[]}
*/
var busiestServers = function(k, arrival, load) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Servers That Handled Most Number of Requests
* Difficulty: Hard
* Tags: array, queue, heap
*
* 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 busiestServers(k: number, arrival: number[], load: number[]):
number[] {

};
```

C# Solution:

```
/*
* Problem: Find Servers That Handled Most Number of Requests
* Difficulty: Hard
```

```
* Tags: array, queue, heap
*
* 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 IList<int> BusiestServers(int k, int[] arrival, int[] load) {

}
}
```

C Solution:

```
/*
* Problem: Find Servers That Handled Most Number of Requests
* Difficulty: Hard
* Tags: array, queue, heap
*
* 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* busiestServers(int k, int* arrival, int arrivalSize, int* load, int
loadSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Find Servers That Handled Most Number of Requests
// Difficulty: Hard
// Tags: array, queue, heap
//
// 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 busiestServers(k int, arrival []int, load []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun busiestServers(k: Int, arrival: IntArray, load: IntArray): List<Int> {

}
}
```

Swift Solution:

```
class Solution {
func busiestServers(_ k: Int, _ arrival: [Int], _ load: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Find Servers That Handled Most Number of Requests
// Difficulty: Hard
// Tags: array, queue, heap
//
// 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 busiest_servers(k: i32, arrival: Vec<i32>, load: Vec<i32>) -> Vec<i32>
{

}
}
```

Ruby Solution:

```
# @param {Integer} k
# @param {Integer[]} arrival
# @param {Integer[]} load
# @return {Integer[]}
def busiest_servers(k, arrival, load)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $k
* @param Integer[] $arrival
* @param Integer[] $load
* @return Integer[]
*/
function busiestServers($k, $arrival, $load) {

}
}
```

Dart Solution:

```
class Solution {
List<int> busiestServers(int k, List<int> arrival, List<int> load) {

}
}
```

Scala Solution:

```
object Solution {
def busiestServers(k: Int, arrival: Array[Int], load: Array[Int]): List[Int]
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec busiest_servers(k :: integer, arrival :: [integer], load :: [integer])
:: [integer]
def busiest_servers(k, arrival, load) do

end
end
```

Erlang Solution:

```
-spec busiest_servers(K :: integer(), Arrival :: [integer()], Load ::
[integer()]) -> [integer()].
busiest_servers(K, Arrival, Load) ->
.
```

Racket Solution:

```
(define/contract (busiest-servers k arrival load)
(-> exact-integer? (listof exact-integer?) (listof exact-integer?) (listof
exact-integer?))
)
```