Problem 2332: The Latest Time to Catch a Bus

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

buses

of length

n

, where

buses[i]

represents the departure time of the

i

th

bus. You are also given a

0-indexed

integer array

passengers

of length

m

, where

passengers[j]

represents the arrival time of the

j

th

passenger. All bus departure times are unique. All passenger arrival times are unique.

You are given an integer

capacity

, which represents the

maximum

number of passengers that can get on each bus.

When a passenger arrives, they will wait in line for the next available bus. You can get on a bus that departs at

x

minutes if you arrive at

y

minutes where

y <= x

, and the bus is not full. Passengers with the

earliest

arrival times get on the bus first.

More formally when a bus arrives, either:

If

capacity

or fewer passengers are waiting for a bus, they will

all

get on the bus, or

The

capacity

passengers with the

earliest

arrival times will get on the bus.

Return

the latest time you may arrive at the bus station to catch a bus

. You

cannot

arrive at the same time as another passenger.

Note:

The arrays

buses

and

passengers

are not necessarily sorted.

Example 1:

Input:

buses = [10,20], passengers = [2,17,18,19], capacity = 2

Output:

16

Explanation:

Suppose you arrive at time 16. At time 10, the first bus departs with the 0

th

passenger. At time 20, the second bus departs with you and the 1

st

passenger. Note that you may not arrive at the same time as another passenger, which is why you must arrive before the 1

st

passenger to catch the bus.

Example 2:

Input:

buses = [20,30,10], passengers = [19,13,26,4,25,11,21], capacity = 2

Output:

20

Explanation:

Suppose you arrive at time 20. At time 10, the first bus departs with the 3

rd

passenger. At time 20, the second bus departs with the 5

th

and 1

st

passengers. At time 30, the third bus departs with the 0

th

passenger and you. Notice if you had arrived any later, then the 6

th

passenger would have taken your seat on the third bus.

Constraints:

n == buses.length

m == passengers.length

1 <= n, m, capacity <= 10

5

2 <= buses[i], passengers[i] <= 10

9

Each element in

buses

is

unique

.

Each element in

passengers

is

unique

.

Code Snippets

C++:

```
class Solution {
public:
```

```
int latestTimeCatchTheBus(vector<int>& buses, vector<int>& passengers, int
capacity) {

}
};
```

Java:

```
class Solution {
public int latestTimeCatchTheBus(int[] buses, int[] passengers, int capacity)
{

}
}
```

Python3:

```
class Solution:
def latestTimeCatchTheBus(self, buses: List[int], passengers: List[int],
capacity: int) -> int:
```

Python:

```
class Solution(object):
def latestTimeCatchTheBus(self, buses, passengers, capacity):
"""
:type buses: List[int]
:type passengers: List[int]
:type capacity: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} buses
* @param {number[]} passengers
* @param {number} capacity
* @return {number}
*/
var latestTimeCatchTheBus = function(buses, passengers, capacity) {
```

```
};
```

TypeScript:

```
function latestTimeCatchTheBus(buses: number[], passengers: number[],
capacity: number): number {

};
```

C#:

```
public class Solution {
public int LatestTimeCatchTheBus(int[] buses, int[] passengers, int capacity)
{

}
}
```

C:

```
int latestTimeCatchTheBus(int* buses, int busesSize, int* passengers, int
passengersSize, int capacity) {

}
```

Go:

```
func latestTimeCatchTheBus(buses []int, passengers []int, capacity int) int {

}
```

Kotlin:

```
class Solution {
fun latestTimeCatchTheBus(buses: IntArray, passengers: IntArray, capacity:
Int): Int {

}
}
```

Swift:

```
class Solution {
func latestTimeCatchTheBus(_ buses: [Int], _ passengers: [Int], _ capacity:
Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn latest_time_catch_the_bus(buses: Vec<i32>, passengers: Vec<i32>,
capacity: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} buses
# @param {Integer[]} passengers
# @param {Integer} capacity
# @return {Integer}
def latest_time_catch_the_bus(buses, passengers, capacity)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $buses
* @param Integer[] $passengers
* @param Integer $capacity
* @return Integer
*/
function latestTimeCatchTheBus($buses, $passengers, $capacity) {

}
}
```

Dart:

```
class Solution {
int latestTimeCatchTheBus(List<int> buses, List<int> passengers, int
capacity) {

}
}
```

Scala:

```
object Solution {
def latestTimeCatchTheBus(buses: Array[Int], passengers: Array[Int],
capacity: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec latest_time_catch_the_bus(buses :: [integer], passengers :: [integer],
capacity :: integer) :: integer
def latest_time_catch_the_bus(buses, passengers, capacity) do

end
end
```

Erlang:

```
-spec latest_time_catch_the_bus(Buses :: [integer()], Passengers ::
[integer()], Capacity :: integer()) -> integer().
latest_time_catch_the_bus(Buses, Passengers, Capacity) ->
.
```

Racket:

```
(define/contract (latest-time-catch-the-bus buses passengers capacity)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: The Latest Time to Catch a Bus
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 latestTimeCatchTheBus(vector<int>& buses, vector<int>& passengers, int
capacity) {

}
};
```

Java Solution:

```
/**
* Problem: The Latest Time to Catch a Bus
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 latestTimeCatchTheBus(int[] buses, int[] passengers, int capacity)
{

}
}
```

Python3 Solution:

```
"""
Problem: The Latest Time to Catch a Bus
```

```
Difficulty: Medium
Tags: array, sort, search

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 latestTimeCatchTheBus(self, buses: List[int], passengers: List[int],
capacity: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def latestTimeCatchTheBus(self, buses, passengers, capacity):
"""
:type buses: List[int]
:type passengers: List[int]
:type capacity: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: The Latest Time to Catch a Bus
* Difficulty: Medium
* Tags: array, sort, search
*
* 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[]} buses
* @param {number[]} passengers
* @param {number} capacity
```

```
* @return {number}
*/
var latestTimeCatchTheBus = function(buses, passengers, capacity) {

};
```

TypeScript Solution:

```
/**
* Problem: The Latest Time to Catch a Bus
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 latestTimeCatchTheBus(buses: number[], passengers: number[],
capacity: number): number {

};
```

C# Solution:

```
/*
* Problem: The Latest Time to Catch a Bus
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 LatestTimeCatchTheBus(int[] buses, int[] passengers, int capacity)
{

}
}
```

C Solution:

```
/*
* Problem: The Latest Time to Catch a Bus
* Difficulty: Medium
* Tags: array, sort, search
*
* 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
*/

int latestTimeCatchTheBus(int* buses, int busesSize, int* passengers, int
passengersSize, int capacity) {

}
```

Go Solution:

```
// Problem: The Latest Time to Catch a Bus
// Difficulty: Medium
// Tags: array, sort, search
//
// 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 latestTimeCatchTheBus(buses []int, passengers []int, capacity int) int {

}
```

Kotlin Solution:

```
class Solution {
fun latestTimeCatchTheBus(buses: IntArray, passengers: IntArray, capacity:
Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func latestTimeCatchTheBus(_ buses: [Int], _ passengers: [Int], _ capacity:
Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: The Latest Time to Catch a Bus
// Difficulty: Medium
// Tags: array, sort, search
//
// 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 latest_time_catch_the_bus(buses: Vec<i32>, passengers: Vec<i32>,
capacity: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} buses
# @param {Integer[]} passengers
# @param {Integer} capacity
# @return {Integer}
def latest_time_catch_the_bus(buses, passengers, capacity)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $buses
* @param Integer[] $passengers
* @param Integer $capacity
```

```
* @return Integer
*/
function latestTimeCatchTheBus($buses, $passengers, $capacity) {

}
}
```

Dart Solution:

```
class Solution {
int latestTimeCatchTheBus(List<int> buses, List<int> passengers, int
capacity) {

}
}
```

Scala Solution:

```
object Solution {
def latestTimeCatchTheBus(buses: Array[Int], passengers: Array[Int],
capacity: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec latest_time_catch_the_bus(buses :: [integer], passengers :: [integer],
capacity :: integer) :: integer
def latest_time_catch_the_bus(buses, passengers, capacity) do

end
end
```

Erlang Solution:

```
-spec latest_time_catch_the_bus(Buses :: [integer()], Passengers ::
[integer()], Capacity :: integer()) -> integer().
latest_time_catch_the_bus(Buses, Passengers, Capacity) ->
.
```

Racket Solution:

```
(define/contract (latest-time-catch-the-bus buses passengers capacity)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?
exact-integer?)
)
```