Problem 2073: Time Needed to Buy Tickets

Problem Information

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

Problem Description

There are

n

people in a line queuing to buy tickets, where the

0

th

person is at the

front

of the line and the

(n - 1)

th

person is at the

back

of the line.

You are given a

0-indexed

integer array

tickets

of length

n

where the number of tickets that the

i

th

person would like to buy is

tickets[i]

.

Each person takes

exactly 1 second

to buy a ticket. A person can only buy

1 ticket at a time

and has to go back to

the end

of the line (which happens

instantaneously

) in order to buy more tickets. If a person does not have any tickets left to buy, the person will

leave

the line.

Return the

time taken

for the person

initially

at position

k

(0-indexed) to finish buying tickets.

Example 1:

Input:

tickets = [2,3,2], k = 2

Output:

6

Explanation:

The queue starts as [2,3,

2

], where the kth person is underlined.

After the person at the front has bought a ticket, the queue becomes [3,

2

,1] at 1 second.

Continuing this process, the queue becomes [

2

,1,2] at 2 seconds.

Continuing this process, the queue becomes [1,2,

1

] at 3 seconds.

Continuing this process, the queue becomes [2,

1

] at 4 seconds. Note: the person at the front left the queue.

Continuing this process, the queue becomes [

1

,1] at 5 seconds.

Continuing this process, the queue becomes [1] at 6 seconds. The kth person has bought all their tickets, so return 6.

Example 2:

Input:

tickets = [5,1,1,1], k = 0

Output:

8

Explanation:

The queue starts as [

5

,1,1,1], where the kth person is underlined.

After the person at the front has bought a ticket, the queue becomes [1,1,1,

4

] at 1 second.

Continuing this process for 3 seconds, the queue becomes [

4]

at 4 seconds.

Continuing this process for 4 seconds, the queue becomes [] at 8 seconds. The kth person has bought all their tickets, so return 8.

Constraints:

n == tickets.length

1 <= n <= 100

1 <= tickets[i] <= 100

0 <= k < n

Code Snippets

C++:

```
class Solution {
public:
int timeRequiredToBuy(vector<int>& tickets, int k) {

}
};
```

Java:

```
class Solution {
public int timeRequiredToBuy(int[] tickets, int k) {

}
}
```

Python3:

```
class Solution:
def timeRequiredToBuy(self, tickets: List[int], k: int) -> int:
```

Python:

```
class Solution(object):
def timeRequiredToBuy(self, tickets, k):
"""
:type tickets: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} tickets
* @param {number} k
* @return {number}
*/
var timeRequiredToBuy = function(tickets, k) {

};
```

TypeScript:

```
function timeRequiredToBuy(tickets: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int TimeRequiredToBuy(int[] tickets, int k) {

}
}
```

C:

```
int timeRequiredToBuy(int* tickets, int ticketsSize, int k) {

}
```

Go:

```
func timeRequiredToBuy(tickets []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun timeRequiredToBuy(tickets: IntArray, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func timeRequiredToBuy(_ tickets: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn time_required_to_buy(tickets: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} tickets
# @param {Integer} k
# @return {Integer}
def time_required_to_buy(tickets, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $tickets
* @param Integer $k
* @return Integer
*/
function timeRequiredToBuy($tickets, $k) {

}
}
```

Dart:

```
class Solution {
int timeRequiredToBuy(List<int> tickets, int k) {

}
}
```

Scala:

```
object Solution {
def timeRequiredToBuy(tickets: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec time_required_to_buy(tickets :: [integer], k :: integer) :: integer
def time_required_to_buy(tickets, k) do

end
end
```

Erlang:

```
-spec time_required_to_buy(Tickets :: [integer()], K :: integer()) ->
integer().
time_required_to_buy(Tickets, K) ->
.
```

Racket:

```
(define/contract (time-required-to-buy tickets k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Time Needed to Buy Tickets
* Difficulty: Easy
* Tags: array, queue
*
* 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 timeRequiredToBuy(vector<int>& tickets, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Time Needed to Buy Tickets
* Difficulty: Easy
* Tags: array, queue
*
* 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 timeRequiredToBuy(int[] tickets, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Time Needed to Buy Tickets
Difficulty: Easy
Tags: array, queue

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 timeRequiredToBuy(self, tickets: List[int], k: int) -> int:
# TODO: Implement optimized solution
```

```

pass
```

Python Solution:

```
class Solution(object):
def timeRequiredToBuy(self, tickets, k):
"""
:type tickets: List[int]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Time Needed to Buy Tickets
* Difficulty: Easy
* Tags: array, queue
*
* 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[]} tickets
* @param {number} k
* @return {number}
*/
var timeRequiredToBuy = function(tickets, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Time Needed to Buy Tickets
* Difficulty: Easy
* Tags: array, queue
*
* 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 timeRequiredToBuy(tickets: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Time Needed to Buy Tickets
* Difficulty: Easy
* Tags: array, queue
*
* 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 TimeRequiredToBuy(int[] tickets, int k) {

}
}
```

C Solution:

```
/*
* Problem: Time Needed to Buy Tickets
* Difficulty: Easy
* Tags: array, queue
*
* 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 timeRequiredToBuy(int* tickets, int ticketsSize, int k) {

}
```

Go Solution:

```
// Problem: Time Needed to Buy Tickets
// Difficulty: Easy
// Tags: array, queue
//
// 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 timeRequiredToBuy(tickets []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun timeRequiredToBuy(tickets: IntArray, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func timeRequiredToBuy(_ tickets: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Time Needed to Buy Tickets
// Difficulty: Easy
// Tags: array, queue
//
// 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 time_required_to_buy(tickets: Vec<i32>, k: i32) -> i32 {


}
}
```

Ruby Solution:

```
# @param {Integer[]} tickets
# @param {Integer} k
# @return {Integer}
def time_required_to_buy(tickets, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $tickets
* @param Integer $k
* @return Integer
*/
function timeRequiredToBuy($tickets, $k) {

}
}
```

Dart Solution:

```
class Solution {
int timeRequiredToBuy(List<int> tickets, int k) {

}
}
```

Scala Solution:

```
object Solution {
def timeRequiredToBuy(tickets: Array[Int], k: Int): Int = {

}

}
```

Elixir Solution:

```
defmodule Solution do
@spec time_required_to_buy(tickets :: [integer], k :: integer) :: integer
def time_required_to_buy(tickets, k) do

end
end
```

Erlang Solution:

```
-spec time_required_to_buy(Tickets :: [integer()], K :: integer()) ->
integer().
time_required_to_buy(Tickets, K) ->
.
```

Racket Solution:

```
(define/contract (time-required-to-buy tickets k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```