Problem 1599: Maximum Profit of Operating a Centennial Wheel

Problem Information

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

Problem Description

You are the operator of a Centennial Wheel that has

four gondolas

, and each gondola has room for

up

to

four people

. You have the ability to rotate the gondolas

counterclockwise

, which costs you

runningCost

dollars.

You are given an array

customers

of length

n

where

customers[i]

is the number of new customers arriving just before the

i

th

rotation (0-indexed). This means you

must rotate the wheel

i

times before the

customers[i]

customers arrive

.

You cannot make customers wait if there is room in the gondola

. Each customer pays

boardingCost

dollars when they board on the gondola closest to the ground and will exit once that gondola reaches the ground again.

You can stop the wheel at any time, including before

serving

all

customers

. If you decide to stop serving customers,

all subsequent rotations are free

in order to get all the customers down safely. Note that if there are currently more than four customers waiting at the wheel, only four will board the gondola, and the rest will wait

for the next rotation

.

Return

the minimum number of rotations you need to perform to maximize your profit.

If there is

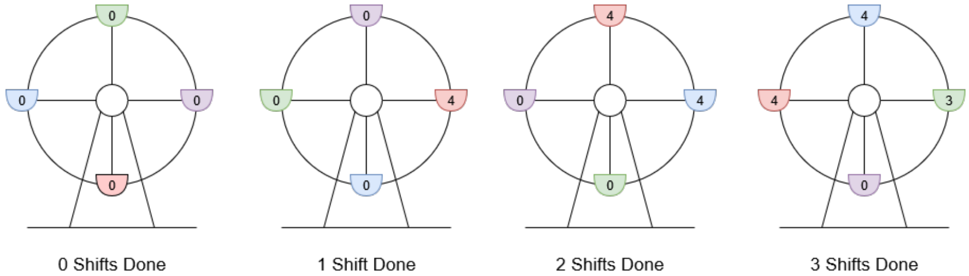
no scenario

where the profit is positive, return

-1

.

Example 1:



Input:

customers = [8,3], boardingCost = 5, runningCost = 6

Output:

3

Explanation:

The numbers written on the gondolas are the number of people currently there. 1. 8 customers arrive, 4 board and 4 wait for the next gondola, the wheel rotates. Current profit is 4 * $5 - 1 * $6 = $14. 2. 3 customers arrive, the 4 waiting board the wheel and the other 3 wait, the wheel rotates. Current profit is 8 * $5 - 2 * $6 = $28. 3. The final 3 customers board the gondola, the wheel rotates. Current profit is 11 * $5 - 3 * $6 = $37. The highest profit was $37 after rotating the wheel 3 times.

Example 2:

Input:

customers = [10,9,6], boardingCost = 6, runningCost = 4

Output:

7

Explanation:

1. 10 customers arrive, 4 board and 6 wait for the next gondola, the wheel rotates. Current profit is 4 * $6 - 1 * $4 = $20. 2. 9 customers arrive, 4 board and 11 wait (2 originally waiting, 9 newly waiting), the wheel rotates. Current profit is 8 * $6 - 2 * $4 = $40. 3. The final 6

customers arrive, 4 board and 13 wait, the wheel rotates. Current profit is 12 * $6 - 3 * $4 = $60. 4. 4 board and 9 wait, the wheel rotates. Current profit is 16 * $6 - 4 * $4 = $80. 5. 4 board and 5 wait, the wheel rotates. Current profit is 20 * $6 - 5 * $4 = $100. 6. 4 board and 1 waits, the wheel rotates. Current profit is 24 * $6 - 6 * $4 = $120. 7. 1 boards, the wheel rotates. Current profit is 25 * $6 - 7 * $4 = $122. The highest profit was $122 after rotating the wheel 7 times.

Example 3:

Input:

customers = [3,4,0,5,1], boardingCost = 1, runningCost = 92

Output:

-1

Explanation:

1. 3 customers arrive, 3 board and 0 wait, the wheel rotates. Current profit is 3 * $1 - 1 * $92 = -$89. 2. 4 customers arrive, 4 board and 0 wait, the wheel rotates. Current profit is 7 * $1 - 2 * $92 = -$177. 3. 0 customers arrive, 0 board and 0 wait, the wheel rotates. Current profit is 7 * $1 - 3 * $92 = -$269. 4. 5 customers arrive, 4 board and 1 waits, the wheel rotates. Current profit is 11 * $1 - 4 * $92 = -$357. 5. 1 customer arrives, 2 board and 0 wait, the wheel rotates. Current profit is 13 * $1 - 5 * $92 = -$447. The profit was never positive, so return -1.

Constraints:

n == customers.length

1 <= n <= 10

5

0 <= customers[i] <= 50

1 <= boardingCost, runningCost <= 100

Code Snippets

C++:

```
class Solution {
public:
int minOperationsMaxProfit(vector<int>& customers, int boardingCost, int
runningCost) {

}
};
```

Java:

```
class Solution {
public int minOperationsMaxProfit(int[] customers, int boardingCost, int
runningCost) {

}
}
```

Python3:

```
class Solution:
def minOperationsMaxProfit(self, customers: List[int], boardingCost: int,
runningCost: int) -> int:
```

Python:

```
class Solution(object):
def minOperationsMaxProfit(self, customers, boardingCost, runningCost):
"""
:type customers: List[int]
:type boardingCost: int
:type runningCost: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} customers
* @param {number} boardingCost
```

```
* @param {number} runningCost
* @return {number}
*/
var minOperationsMaxProfit = function(customers, boardingCost, runningCost) {

};
```

TypeScript:

```
function minOperationsMaxProfit(customers: number[], boardingCost: number,
runningCost: number): number {

};
```

C#:

```
public class Solution {
public int MinOperationsMaxProfit(int[] customers, int boardingCost, int
runningCost) {

}
}
```

C:

```
int minOperationsMaxProfit(int* customers, int customersSize, int
boardingCost, int runningCost) {

}
```

Go:

```
func minOperationsMaxProfit(customers []int, boardingCost int, runningCost
int) int {

}
```

Kotlin:

```
class Solution {
fun minOperationsMaxProfit(customers: IntArray, boardingCost: Int,
runningCost: Int): Int {
```

```
}
}
```

Swift:

```
class Solution {
func minOperationsMaxProfit(_ customers: [Int], _ boardingCost: Int, _
runningCost: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_operations_max_profit(customers: Vec<i32>, boarding_cost: i32,
running_cost: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} customers
# @param {Integer} boarding_cost
# @param {Integer} running_cost
# @return {Integer}
def min_operations_max_profit(customers, boarding_cost, running_cost)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $customers
* @param Integer $boardingCost
* @param Integer $runningCost
* @return Integer
*/
```

```
function minOperationsMaxProfit($customers, $boardingCost, $runningCost) {

}
}
```

Dart:

```
class Solution {
int minOperationsMaxProfit(List<int> customers, int boardingCost, int
runningCost) {

}
}
```

Scala:

```
object Solution {
def minOperationsMaxProfit(customers: Array[Int], boardingCost: Int,
runningCost: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_operations_max_profit(customers :: [integer], boarding_cost ::
integer, running_cost :: integer) :: integer
def min_operations_max_profit(customers, boarding_cost, running_cost) do

end
end
```

Erlang:

```
-spec min_operations_max_profit(Customers :: [integer()], BoardingCost ::
integer(), RunningCost :: integer()) -> integer().
min_operations_max_profit(Customers, BoardingCost, RunningCost) ->
.
```

Racket:

```
(define/contract (min-operations-max-profit customers boardingCost
runningCost)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Profit of Operating a Centennial Wheel
* Difficulty: Medium
* Tags: array
*
* 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 minOperationsMaxProfit(vector<int>& customers, int boardingCost, int
runningCost) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Profit of Operating a Centennial Wheel
* Difficulty: Medium
* Tags: array
*
* 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 minOperationsMaxProfit(int[] customers, int boardingCost, int
```

```
runningCost) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Profit of Operating a Centennial Wheel
Difficulty: Medium
Tags: array

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 minOperationsMaxProfit(self, customers: List[int], boardingCost: int,
runningCost: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minOperationsMaxProfit(self, customers, boardingCost, runningCost):
"""
:type customers: List[int]
:type boardingCost: int
:type runningCost: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Profit of Operating a Centennial Wheel
* Difficulty: Medium
* Tags: array
*
* 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[]} customers
* @param {number} boardingCost
* @param {number} runningCost
* @return {number}
*/
var minOperationsMaxProfit = function(customers, boardingCost, runningCost) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Profit of Operating a Centennial Wheel
* Difficulty: Medium
* Tags: array
*
* 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 minOperationsMaxProfit(customers: number[], boardingCost: number,
runningCost: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Profit of Operating a Centennial Wheel
* Difficulty: Medium
* Tags: array
*
* 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 MinOperationsMaxProfit(int[] customers, int boardingCost, int
runningCost) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Profit of Operating a Centennial Wheel
* Difficulty: Medium
* Tags: array
*
* 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 minOperationsMaxProfit(int* customers, int customersSize, int
boardingCost, int runningCost) {

}
```

Go Solution:

```
// Problem: Maximum Profit of Operating a Centennial Wheel
// Difficulty: Medium
// Tags: array
//
// 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 minOperationsMaxProfit(customers []int, boardingCost int, runningCost
int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minOperationsMaxProfit(customers: IntArray, boardingCost: Int,
runningCost: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minOperationsMaxProfit(_ customers: [Int], _ boardingCost: Int, _
runningCost: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Profit of Operating a Centennial Wheel
// Difficulty: Medium
// Tags: array
//
// 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 min_operations_max_profit(customers: Vec<i32>, boarding_cost: i32,
running_cost: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} customers
# @param {Integer} boarding_cost
# @param {Integer} running_cost
# @return {Integer}
def min_operations_max_profit(customers, boarding_cost, running_cost)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $customers
* @param Integer $boardingCost
* @param Integer $runningCost
* @return Integer
*/
function minOperationsMaxProfit($customers, $boardingCost, $runningCost) {

}
}
```

Dart Solution:

```
class Solution {
int minOperationsMaxProfit(List<int> customers, int boardingCost, int
runningCost) {

}
}
```

Scala Solution:

```
object Solution {
def minOperationsMaxProfit(customers: Array[Int], boardingCost: Int,
runningCost: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_operations_max_profit(customers :: [integer], boarding_cost ::
integer, running_cost :: integer) :: integer
```

```
def min_operations_max_profit(customers, boarding_cost, running_cost) do

end
end
```

Erlang Solution:

```
-spec min_operations_max_profit(Customers :: [integer()], BoardingCost ::
integer(), RunningCost :: integer()) -> integer().
min_operations_max_profit(Customers, BoardingCost, RunningCost) ->
.
```

Racket Solution:

```
(define/contract (min-operations-max-profit customers boardingCost
runningCost)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```