Problem 3711: Maximum Transactions Without Negative Balance

Problem Information

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

Problem Description

You are given an integer array

transactions

, where

transactions[i]

represents the amount of the

i

th

transaction:

A positive value means money is

received

.

A negative value means money is

sent

.

The account starts with a balance of 0, and the balance

must never become negative

. Transactions must be considered in the given order, but you are allowed to skip some transactions.

Return an integer denoting the

maximum number of transactions

that can be performed without the balance ever going negative.

Example 1:

Input:

transactions = [2,-5,3,-1,-2]

Output:

4

Explanation:

One optimal sequence is

[2, 3, -1, -2]

, balance:

0 → 2 → 5 → 4 → 2

.

Example 2:

Input:

transactions = [-1,-2,-3]

Output:

0

Explanation:

All transactions are negative. Including any would make the balance negative.

Example 3:

Input:

transactions = [3,-2,3,-2,1,-1]

Output:

6

Explanation:

All transactions can be taken in order, balance:

0 → 3 → 1 → 4 → 2 → 3 → 2

.

Constraints:

1 <= transactions.length <= 10

5

-10

9

<= transactions[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int maxTransactions(vector<int>& transactions) {

}
};
```

Java:

```
class Solution {
public int maxTransactions(int[] transactions) {

}
}
```

Python3:

```
class Solution:
def maxTransactions(self, transactions: List[int]) -> int:
```

Python:

```
class Solution(object):
def maxTransactions(self, transactions):
"""
:type transactions: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} transactions
* @return {number}
*/
var maxTransactions = function(transactions) {

};
```

TypeScript:

```
function maxTransactions(transactions: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxTransactions(int[] transactions) {

}
}
```

C:

```
int maxTransactions(int* transactions, int transactionsSize) {

}
```

Go:

```
func maxTransactions(transactions []int) int {

}
```

Kotlin:

```
class Solution {
fun maxTransactions(transactions: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxTransactions(_ transactions: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_transactions(transactions: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} transactions
# @return {Integer}
def max_transactions(transactions)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $transactions
* @return Integer
*/
function maxTransactions($transactions) {

}
}
```

Dart:

```
class Solution {
int maxTransactions(List<int> transactions) {

}
}
```

Scala:

```
object Solution {
def maxTransactions(transactions: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_transactions(transactions :: [integer]) :: integer
def max_transactions(transactions) do

end
end
```

Erlang:

```
-spec max_transactions(Transactions :: [integer()]) -> integer().
max_transactions(Transactions) ->
.
```

Racket:

```
(define/contract (max-transactions transactions)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Transactions Without Negative Balance
* Difficulty: Medium
* Tags: array, greedy, 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:
int maxTransactions(vector<int>& transactions) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Transactions Without Negative Balance
* Difficulty: Medium
* Tags: array, greedy, 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 int maxTransactions(int[] transactions) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Transactions Without Negative Balance
Difficulty: Medium
Tags: array, greedy, 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 maxTransactions(self, transactions: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def maxTransactions(self, transactions):
"""
:type transactions: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Transactions Without Negative Balance
* Difficulty: Medium
* Tags: array, greedy, 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[]} transactions
* @return {number}
*/
var maxTransactions = function(transactions) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Transactions Without Negative Balance
* Difficulty: Medium
* Tags: array, greedy, 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 maxTransactions(transactions: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Transactions Without Negative Balance
* Difficulty: Medium
* Tags: array, greedy, 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 int MaxTransactions(int[] transactions) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Transactions Without Negative Balance
* Difficulty: Medium
* Tags: array, greedy, 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
*/

int maxTransactions(int* transactions, int transactionsSize) {

}
```

Go Solution:

```
// Problem: Maximum Transactions Without Negative Balance
// Difficulty: Medium
// Tags: array, greedy, 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 maxTransactions(transactions []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxTransactions(transactions: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxTransactions(_ transactions: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Transactions Without Negative Balance
// Difficulty: Medium
// Tags: array, greedy, 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 max_transactions(transactions: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} transactions
# @return {Integer}
def max_transactions(transactions)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $transactions
* @return Integer
*/
function maxTransactions($transactions) {

}
}
```

Dart Solution:

```
class Solution {
int maxTransactions(List<int> transactions) {

}
}
```

Scala Solution:

```
object Solution {
def maxTransactions(transactions: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_transactions(transactions :: [integer]) :: integer
def max_transactions(transactions) do

end
end
```

Erlang Solution:

```
-spec max_transactions(Transactions :: [integer()]) -> integer().
max_transactions(Transactions) ->
.
```

Racket Solution:

```
(define/contract (max-transactions transactions)
(-> (listof exact-integer?) exact-integer?)
)
```