Problem 2819: Minimum Relative Loss After Buying Chocolates

Problem Information

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

Problem Description

You are given an integer array

prices

, which shows the chocolate prices and a 2D integer array

queries

, where

queries[i] = [k

i

, m

i

]

.

Alice and Bob went to buy some chocolates, and Alice suggested a way to pay for them, and Bob agreed.

The terms for each query are as follows:

If the price of a chocolate is

less than or equal to

k

i

, Bob pays for it.

Otherwise, Bob pays

k

i

of it, and Alice pays the

rest

.

Bob wants to select

exactly

m

i

chocolates such that his

relative loss

is

minimized

, more formally, if, in total, Alice has paid

a

i

and Bob has paid

b

i

, Bob wants to minimize

b

i

- a

i

.

Return

an integer array

ans

where

ans[i]

is Bob's

minimum relative loss

possible for

queries[i]

.

Example 1:

Input:

prices = [1,9,22,10,19], queries = [[18,4],[5,2]]

Output:

[34,-21]

Explanation:

For the 1

st

query Bob selects the chocolates with prices [1,9,10,22]. He pays 1 + 9 + 10 + 18 = 38 and Alice pays 0 + 0 + 0 + 4 = 4. So Bob's relative loss is 38 - 4 = 34. For the 2

nd

query Bob selects the chocolates with prices [19,22]. He pays 5 + 5 = 10 and Alice pays 14 + 17 = 31. So Bob's relative loss is 10 - 31 = -21. It can be shown that these are the minimum possible relative losses.

Example 2:

Input:

prices = [1,5,4,3,7,11,9], queries = [[5,4],[5,7],[7,3],[4,5]]

Output:

[4,16,7,1]

Explanation:

For the 1

st

query Bob selects the chocolates with prices [1,3,9,11]. He pays 1 + 3 + 5 + 5 = 14 and Alice pays 0 + 0 + 4 + 6 = 10. So Bob's relative loss is 14 - 10 = 4. For the 2

nd

query Bob has to select all the chocolates. He pays 1 + 5 + 4 + 3 + 5 + 5 + 5 = 28 and Alice pays 0 + 0 + 0 + 0 + 2 + 6 + 4 = 12. So Bob's relative loss is 28 - 12 = 16. For the 3

rd

query Bob selects the chocolates with prices [1,3,11] and he pays 1 + 3 + 7 = 11 and Alice pays 0 + 0 + 4 = 4. So Bob's relative loss is 11 - 4 = 7. For the 4

th

query Bob selects the chocolates with prices [1,3,7,9,11] and he pays 1 + 3 + 4 + 4 + 4 = 16 and Alice pays 0 + 0 + 3 + 5 + 7 = 15. So Bob's relative loss is 16 - 15 = 1. It can be shown that these are the minimum possible relative losses.

Example 3:

Input:

prices = [5,6,7], queries = [[10,1],[5,3],[3,3]]

Output:

[5,12,0]

Explanation:

For the 1

st

query Bob selects the chocolate with price 5 and he pays 5 and Alice pays 0. So Bob's relative loss is 5 - 0 = 5. For the 2

nd

query Bob has to select all the chocolates. He pays 5 + 5 + 5 = 15 and Alice pays 0 + 1 + 2 = 3. So Bob's relative loss is 15 - 3 = 12. For the 3

rd

query Bob has to select all the chocolates. He pays 3 + 3 + 3 = 9 and Alice pays 2 + 3 + 4 = 9. So Bob's relative loss is 9 - 9 = 0. It can be shown that these are the minimum possible relative losses.

Constraints:

1 <= prices.length == n <= 10

5

1 <= prices[i] <= 10

9

1 <= queries.length <= 10

5

queries[i].length == 2

1 <= k

i

<= 10

9

1 <= m

i

<= n

Code Snippets

C++:

```
class Solution {
public:
vector<long long> minimumRelativeLosses(vector<int>& prices,
vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public long[] minimumRelativeLosses(int[] prices, int[][] queries) {

}
}
```

Python3:

```
class Solution:
def minimumRelativeLosses(self, prices: List[int], queries: List[List[int]])
-> List[int]:
```

Python:

```
class Solution(object):
def minimumRelativeLosses(self, prices, queries):
"""
:type prices: List[int]
```

```
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} prices
* @param {number[][]} queries
* @return {number[]}
*/
var minimumRelativeLosses = function(prices, queries) {

};
```

TypeScript:

```
function minimumRelativeLosses(prices: number[], queries: number[][]):
number[] {

};
```

C#:

```
public class Solution {
public long[] MinimumRelativeLosses(int[] prices, int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* minimumRelativeLosses(int* prices, int pricesSize, int** queries,
int queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go:

```
func minimumRelativeLosses(prices []int, queries [][]int) []int64 {

}
```

Kotlin:

```
class Solution {
fun minimumRelativeLosses(prices: IntArray, queries: Array<IntArray>):
LongArray {

}
}
```

Swift:

```
class Solution {
func minimumRelativeLosses(_ prices: [Int], _ queries: [[Int]]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_relative_losses(prices: Vec<i32>, queries: Vec<Vec<i32>>) ->
Vec<i64> {

}
}
```

Ruby:

```
# @param {Integer[]} prices
# @param {Integer[][]} queries
# @return {Integer[]}
def minimum_relative_losses(prices, queries)

end
```

PHP:

```
class Solution {
```

```
/**
* @param Integer[] $prices
* @param Integer[][] $queries
* @return Integer[]
*/
function minimumRelativeLosses($prices, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> minimumRelativeLosses(List<int> prices, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def minimumRelativeLosses(prices: Array[Int], queries: Array[Array[Int]]):
Array[Long] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_relative_losses(prices :: [integer], queries :: [[integer]]) ::
[integer]
def minimum_relative_losses(prices, queries) do

end
end
```

Erlang:

```
-spec minimum_relative_losses(Prices :: [integer()], Queries ::
[[integer()]]) -> [integer()].
minimum_relative_losses(Prices, Queries) ->
```

```
.
```

Racket:

```
(define/contract (minimum-relative-losses prices queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Relative Loss After Buying Chocolates
* Difficulty: Hard
* 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:
vector<long long> minimumRelativeLosses(vector<int>& prices,
vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Relative Loss After Buying Chocolates
* Difficulty: Hard
* 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 long[] minimumRelativeLosses(int[] prices, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Relative Loss After Buying Chocolates
Difficulty: Hard
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 minimumRelativeLosses(self, prices: List[int], queries: List[List[int]])
-> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumRelativeLosses(self, prices, queries):
"""
:type prices: List[int]
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Relative Loss After Buying Chocolates
```

```
* Difficulty: Hard
* 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[]} prices
* @param {number[][]} queries
* @return {number[]}
*/
var minimumRelativeLosses = function(prices, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Relative Loss After Buying Chocolates
* Difficulty: Hard
* 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 minimumRelativeLosses(prices: number[], queries: number[][]):
number[] {

};
```

C# Solution:

```
/*
* Problem: Minimum Relative Loss After Buying Chocolates
* Difficulty: Hard
* 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 long[] MinimumRelativeLosses(int[] prices, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Relative Loss After Buying Chocolates
* Difficulty: Hard
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* minimumRelativeLosses(int* prices, int pricesSize, int** queries,
int queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Minimum Relative Loss After Buying Chocolates
// Difficulty: Hard
// 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 minimumRelativeLosses(prices []int, queries [][]int) []int64 {

}
```

Kotlin Solution:

```
class Solution {
fun minimumRelativeLosses(prices: IntArray, queries: Array<IntArray>):
LongArray {

}
}
```

Swift Solution:

```
class Solution {
func minimumRelativeLosses(_ prices: [Int], _ queries: [[Int]]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Minimum Relative Loss After Buying Chocolates
// Difficulty: Hard
// 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 minimum_relative_losses(prices: Vec<i32>, queries: Vec<Vec<i32>>) ->
Vec<i64> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} prices
# @param {Integer[][]} queries
# @return {Integer[]}
def minimum_relative_losses(prices, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $prices
* @param Integer[][] $queries
* @return Integer[]
*/
function minimumRelativeLosses($prices, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> minimumRelativeLosses(List<int> prices, List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def minimumRelativeLosses(prices: Array[Int], queries: Array[Array[Int]]):
Array[Long] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_relative_losses(prices :: [integer], queries :: [[integer]]) ::
```

```
[integer]
def minimum_relative_losses(prices, queries) do

end
end
```

Erlang Solution:

```
-spec minimum_relative_losses(Prices :: [integer()], Queries ::
[[integer()]]) -> [integer()].
minimum_relative_losses(Prices, Queries) ->
.
```

Racket Solution:

```
(define/contract (minimum-relative-losses prices queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```