Problem 3219: Minimum Cost for Cutting Cake II

Problem Information

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

Problem Description

There is an

m x n

cake that needs to be cut into

1 x 1

pieces.

You are given integers

m

,

n

, and two arrays:

horizontalCut

of size

m - 1

, where

horizontalCut[i]

represents the cost to cut along the horizontal line

i

.

verticalCut

of size

n - 1

, where

verticalCut[j]

represents the cost to cut along the vertical line

j

.

In one operation, you can choose any piece of cake that is not yet a

1 x 1

square and perform one of the following cuts:

Cut along a horizontal line

i

at a cost of

horizontalCut[i]

.

Cut along a vertical line

j

at a cost of

verticalCut[j]

.

After the cut, the piece of cake is divided into two distinct pieces.

The cost of a cut depends only on the initial cost of the line and does not change.

Return the

minimum

total cost to cut the entire cake into

1 x 1

pieces.

Example 1:

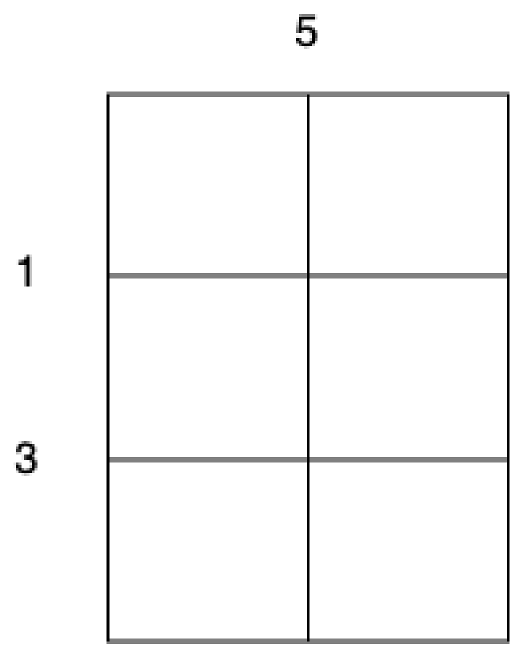
Input:

m = 3, n = 2, horizontalCut = [1,3], verticalCut = [5]

Output:

13

Explanation:



Perform a cut on the vertical line 0 with cost 5, current total cost is 5.

Perform a cut on the horizontal line 0 on

3 x 1

subgrid with cost 1.

Perform a cut on the horizontal line 0 on

3 x 1

subgrid with cost 1.

Perform a cut on the horizontal line 1 on

2 x 1

subgrid with cost 3.

Perform a cut on the horizontal line 1 on

2 x 1

subgrid with cost 3.

The total cost is

5 + 1 + 1 + 3 + 3 = 13

.

Example 2:

Input:

m = 2, n = 2, horizontalCut = [7], verticalCut = [4]

Output:

15

Explanation:

Perform a cut on the horizontal line 0 with cost 7.

Perform a cut on the vertical line 0 on

1 x 2

subgrid with cost 4.

Perform a cut on the vertical line 0 on

1 x 2

subgrid with cost 4.

The total cost is

7 + 4 + 4 = 15

.

Constraints:

1 <= m, n <= 10

5

horizontalCut.length == m - 1

verticalCut.length == n - 1

1 <= horizontalCut[i], verticalCut[i] <= 10

3

Code Snippets

C++:

```
class Solution {
public:
long long minimumCost(int m, int n, vector<int>& horizontalCut, vector<int>&
verticalCut) {

}
};
```

Java:

```
class Solution {
public long minimumCost(int m, int n, int[] horizontalCut, int[] verticalCut)
{

}
}
```

Python3:

```
class Solution:
def minimumCost(self, m: int, n: int, horizontalCut: List[int], verticalCut:
List[int]) -> int:
```

Python:

```
class Solution(object):
def minimumCost(self, m, n, horizontalCut, verticalCut):
"""
:type m: int
:type n: int
:type horizontalCut: List[int]
:type verticalCut: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} m
* @param {number} n
* @param {number[]} horizontalCut
* @param {number[]} verticalCut
* @return {number}
*/
var minimumCost = function(m, n, horizontalCut, verticalCut) {

};
```

TypeScript:

```
function minimumCost(m: number, n: number, horizontalCut: number[],
verticalCut: number[]): number {

};
```

C#:

```
public class Solution {
public long MinimumCost(int m, int n, int[] horizontalCut, int[] verticalCut)
{
```

```
}
}
```

C:

```
long long minimumCost(int m, int n, int* horizontalCut, int
horizontalCutSize, int* verticalCut, int verticalCutSize) {

}
```

Go:

```
func minimumCost(m int, n int, horizontalCut []int, verticalCut []int) int64
{

}
```

Kotlin:

```
class Solution {
fun minimumCost(m: Int, n: Int, horizontalCut: IntArray, verticalCut:
IntArray): Long {

}
}
```

Swift:

```
class Solution {
func minimumCost(_ m: Int, _ n: Int, _ horizontalCut: [Int], _ verticalCut:
[Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_cost(m: i32, n: i32, horizontal_cut: Vec<i32>, vertical_cut:
Vec<i32>) -> i64 {
```

```
}
}
```

Ruby:

```ruby
# @param {Integer} m
# @param {Integer} n
# @param {Integer[]} horizontal_cut
# @param {Integer[]} vertical_cut
# @return {Integer}
def minimum_cost(m, n, horizontal_cut, vertical_cut)

end
```

PHP:

```php
class Solution {

/**
* @param Integer $m
* @param Integer $n
* @param Integer[] $horizontalCut
* @param Integer[] $verticalCut
* @return Integer
*/
function minimumCost($m, $n, $horizontalCut, $verticalCut) {

}
}
```

Dart:

```dart
class Solution {
int minimumCost(int m, int n, List<int> horizontalCut, List<int> verticalCut)
{

}
}
```

Scala:

```
object Solution {
def minimumCost(m: Int, n: Int, horizontalCut: Array[Int], verticalCut:
Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_cost(m :: integer, n :: integer, horizontal_cut :: [integer],
vertical_cut :: [integer]) :: integer
def minimum_cost(m, n, horizontal_cut, vertical_cut) do

end
end
```

Erlang:

```
-spec minimum_cost(M :: integer(), N :: integer(), HorizontalCut ::
[integer()], VerticalCut :: [integer()]) -> integer().
minimum_cost(M, N, HorizontalCut, VerticalCut) ->
.
```

Racket:

```
(define/contract (minimum-cost m n horizontalCut verticalCut)
(-> exact-integer? exact-integer? (listof exact-integer?) (listof
exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Cost for Cutting Cake II
* Difficulty: Hard
* Tags: array, greedy, sort
*
* 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 long minimumCost(int m, int n, vector<int>& horizontalCut, vector<int>&
verticalCut) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Cost for Cutting Cake II
* Difficulty: Hard
* Tags: array, greedy, sort
*
* 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 minimumCost(int m, int n, int[] horizontalCut, int[] verticalCut)
{

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Cost for Cutting Cake II
Difficulty: Hard
Tags: array, greedy, sort

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 minimumCost(self, m: int, n: int, horizontalCut: List[int], verticalCut:
List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumCost(self, m, n, horizontalCut, verticalCut):
"""
:type m: int
:type n: int
:type horizontalCut: List[int]
:type verticalCut: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Cost for Cutting Cake II
* Difficulty: Hard
* Tags: array, greedy, sort
*
* 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} m
* @param {number} n
* @param {number[]} horizontalCut
* @param {number[]} verticalCut
* @return {number}
*/
var minimumCost = function(m, n, horizontalCut, verticalCut) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Minimum Cost for Cutting Cake II
* Difficulty: Hard
* Tags: array, greedy, sort
*
* 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 minimumCost(m: number, n: number, horizontalCut: number[],
verticalCut: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Cost for Cutting Cake II
* Difficulty: Hard
* Tags: array, greedy, sort
*
* 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 MinimumCost(int m, int n, int[] horizontalCut, int[] verticalCut)
{

}
}
```

C Solution:

```
/*
* Problem: Minimum Cost for Cutting Cake II
* Difficulty: Hard
* Tags: array, greedy, sort
*
* 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
*/

long long minimumCost(int m, int n, int* horizontalCut, int
horizontalCutSize, int* verticalCut, int verticalCutSize) {

}
```

Go Solution:

```
// Problem: Minimum Cost for Cutting Cake II
// Difficulty: Hard
// Tags: array, greedy, sort
//
// 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 minimumCost(m int, n int, horizontalCut []int, verticalCut []int) int64
{

}
```

Kotlin Solution:

```
class Solution {
fun minimumCost(m: Int, n: Int, horizontalCut: IntArray, verticalCut:
IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func minimumCost(_ m: Int, _ n: Int, _ horizontalCut: [Int], _ verticalCut:
[Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Cost for Cutting Cake II
// Difficulty: Hard
// Tags: array, greedy, sort
//
// 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_cost(m: i32, n: i32, horizontal_cut: Vec<i32>, vertical_cut:
Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer} m
# @param {Integer} n
# @param {Integer[]} horizontal_cut
# @param {Integer[]} vertical_cut
# @return {Integer}
def minimum_cost(m, n, horizontal_cut, vertical_cut)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $m
* @param Integer $n
```

```
* @param Integer[] $horizontalCut
* @param Integer[] $verticalCut
* @return Integer
*/
function minimumCost($m, $n, $horizontalCut, $verticalCut) {

}
}
```

Dart Solution:

```
class Solution {
int minimumCost(int m, int n, List<int> horizontalCut, List<int> verticalCut)
{

}
}
```

Scala Solution:

```
object Solution {
def minimumCost(m: Int, n: Int, horizontalCut: Array[Int], verticalCut:
Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_cost(m :: integer, n :: integer, horizontal_cut :: [integer],
vertical_cut :: [integer]) :: integer
def minimum_cost(m, n, horizontal_cut, vertical_cut) do

end
end
```

Erlang Solution:

```
-spec minimum_cost(M :: integer(), N :: integer(), HorizontalCut ::
[integer()], VerticalCut :: [integer()]) -> integer().
```

```
minimum_cost(M, N, HorizontalCut, VerticalCut) ->
.
```

Racket Solution:

```
(define/contract (minimum-cost m n horizontalCut verticalCut)
(-> exact-integer? exact-integer? (listof exact-integer?) (listof
exact-integer?) exact-integer?)
)
```