Problem 1029: Two City Scheduling

Problem Information

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

Problem Description

A company is planning to interview

2n

people. Given the array

costs

where

costs[i] = [aCost

i

, bCost

i

]

, the cost of flying the

i

th

person to city

a

is

aCost

i

, and the cost of flying the

i

th

person to city

b

is

bCost

i

.

Return

the minimum cost to fly every person to a city

such that exactly

n

people arrive in each city.

Example 1:

Input:

costs = [[10,20],[30,200],[400,50],[30,20]]

Output:

110

Explanation:

The first person goes to city A for a cost of 10. The second person goes to city A for a cost of 30. The third person goes to city B for a cost of 50. The fourth person goes to city B for a cost of 20.

The total minimum cost is 10 + 30 + 50 + 20 = 110 to have half the people interviewing in each city.

Example 2:

Input:

costs = [[259,770],[448,54],[926,667],[184,139],[840,118],[577,469]]

Output:

1859

Example 3:

Input:

costs = [[515,563],[451,713],[537,709],[343,819],[855,779],[457,60],[650,359],[631,42]]

Output:

3086

Constraints:

2 * n == costs.length

2 <= costs.length <= 100

costs.length

is even.

1 <= aCost

i

, bCost

i

<= 1000

Code Snippets

C++:

```
class Solution {
public:
int twoCitySchedCost(vector<vector<int>>& costs) {

}
};
```

Java:

```
class Solution {
public int twoCitySchedCost(int[][] costs) {

}
}
```

Python3:

```
class Solution:
def twoCitySchedCost(self, costs: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function twoCitySchedCost(costs: number[][]): number {

};
```

C#:

```
public class Solution {
public int TwoCitySchedCost(int[][] costs) {

}
}
```

C:

```
int twoCitySchedCost(int** costs, int costsSize, int* costsColSize) {

}
```

Go:

```
func twoCitySchedCost(costs [][]int) int {

}
```

Kotlin:

```
class Solution {
fun twoCitySchedCost(costs: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func twoCitySchedCost(_ costs: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn two_city_sched_cost(costs: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} costs
# @return {Integer}
def two_city_sched_cost(costs)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $costs
* @return Integer
```

```
*/
function twoCitySchedCost($costs) {

}
}
```

Dart:

```
class Solution {
int twoCitySchedCost(List<List<int>> costs) {

}
}
```

Scala:

```
object Solution {
def twoCitySchedCost(costs: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec two_city_sched_cost(costs :: [[integer]]) :: integer
def two_city_sched_cost(costs) do

end
end
```

Erlang:

```
-spec two_city_sched_cost(Costs :: [[integer()]]) -> integer().
two_city_sched_cost(Costs) ->
.
```

Racket:

```
(define/contract (two-city-sched-cost costs)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Two City Scheduling
* Difficulty: Medium
* 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:
int twoCitySchedCost(vector<vector<int>>& costs) {

}
};
```

Java Solution:

```
/**
* Problem: Two City Scheduling
* Difficulty: Medium
* 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 int twoCitySchedCost(int[][] costs) {

}
}
```

Python3 Solution:

```
"""
Problem: Two City Scheduling
Difficulty: Medium
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 twoCitySchedCost(self, costs: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Two City Scheduling
* Difficulty: Medium
* 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[][]} costs
* @return {number}
*/
var twoCitySchedCost = function(costs) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Two City Scheduling
* Difficulty: Medium
* 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 twoCitySchedCost(costs: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Two City Scheduling
* Difficulty: Medium
* 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 int TwoCitySchedCost(int[][] costs) {

}
}
```

C Solution:

```
/*
* Problem: Two City Scheduling
* Difficulty: Medium
```

```
* 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
*/

int twoCitySchedCost(int** costs, int costsSize, int* costsColSize) {

}
```

Go Solution:

```
// Problem: Two City Scheduling
// Difficulty: Medium
// 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 twoCitySchedCost(costs [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun twoCitySchedCost(costs: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func twoCitySchedCost(_ costs: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Two City Scheduling
// Difficulty: Medium
// 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 two_city_sched_cost(costs: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} costs
# @return {Integer}
def two_city_sched_cost(costs)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $costs
* @return Integer
*/
function twoCitySchedCost($costs) {

}
}
```

Dart Solution:

```
class Solution {
int twoCitySchedCost(List<List<int>> costs) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def twoCitySchedCost(costs: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec two_city_sched_cost(costs :: [[integer]]) :: integer
def two_city_sched_cost(costs) do

end
end
```

Erlang Solution:

```
-spec two_city_sched_cost(Costs :: [[integer()]]) -> integer().
two_city_sched_cost(Costs) ->
.
```

Racket Solution:

```
(define/contract (two-city-sched-cost costs)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```