Problem 1217: Minimum Cost to Move Chips to The Same Position

Problem Information

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

Problem Description

We have

n

chips, where the position of the

i

th

chip is

position[i]

.

We need to move all the chips to

the same position

. In one step, we can change the position of the

i

th

chip from

position[i]

to:

position[i] + 2

or

position[i] - 2

with

cost = 0

.

position[i] + 1

or

position[i] - 1

with

cost = 1

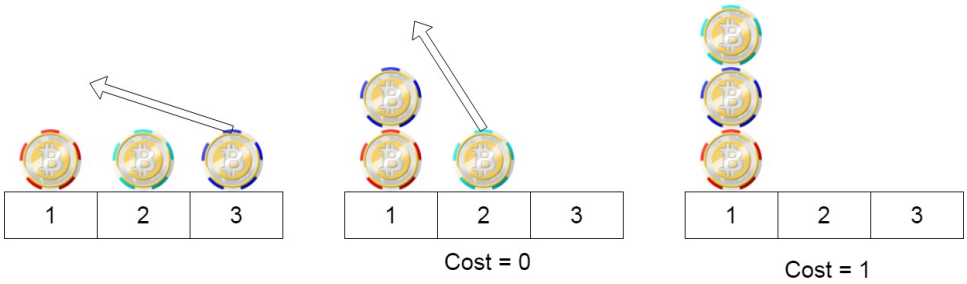
.

Return

the minimum cost

needed to move all the chips to the same position.

Example 1:



Input:

position = [1,2,3]

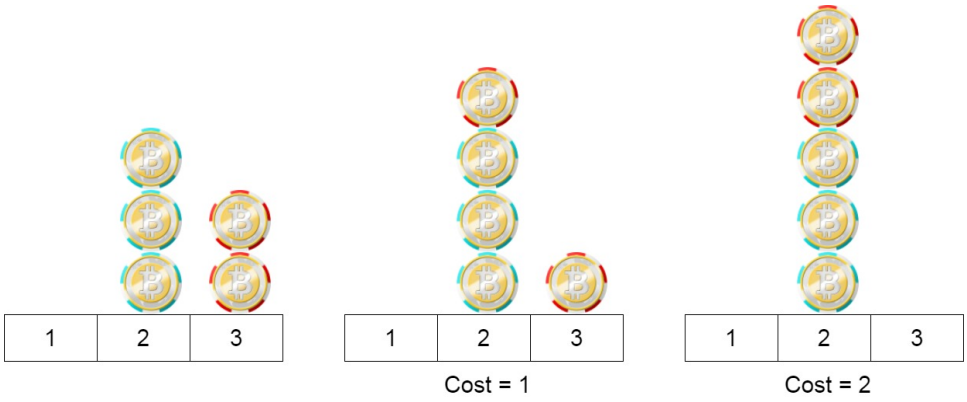
Output:

1

Explanation:

First step: Move the chip at position 3 to position 1 with cost = 0. Second step: Move the chip at position 2 to position 1 with cost = 1. Total cost is 1.

Example 2:



Input:

position = [2,2,2,3,3]

Output:

2

Explanation:

We can move the two chips at position 3 to position 2. Each move has cost = 1. The total cost = 2.

Example 3:

Input:

position = [1,1000000000]

Output:

1

Constraints:

1 <= position.length <= 100

1 <= position[i] <= 10^9

Code Snippets

C++:

```
class Solution {
public:
int minCostToMoveChips(vector<int>& position) {

}
};
```

Java:

```
class Solution {
public int minCostToMoveChips(int[] position) {

}
}
```

Python3:

```
class Solution:
def minCostToMoveChips(self, position: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minCostToMoveChips(position: number[]): number {

};
```

C#:

```
public class Solution {
public int MinCostToMoveChips(int[] position) {

}
}
```

C:

```
int minCostToMoveChips(int* position, int positionSize) {

}
```

Go:

```
func minCostToMoveChips(position []int) int {

}
```

Kotlin:

```
class Solution {
fun minCostToMoveChips(position: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minCostToMoveChips(_ position: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_cost_to_move_chips(position: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} position
# @return {Integer}
def min_cost_to_move_chips(position)

end
```

PHP:

```
class Solution {

/**
```

```
* @param Integer[] $position
* @return Integer
*/
function minCostToMoveChips($position) {

}
}
```

Dart:

```
class Solution {
int minCostToMoveChips(List<int> position) {

}
}
```

Scala:

```
object Solution {
def minCostToMoveChips(position: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_cost_to_move_chips(position :: [integer]) :: integer
def min_cost_to_move_chips(position) do

end
end
```

Erlang:

```
-spec min_cost_to_move_chips(Position :: [integer()]) -> integer().
min_cost_to_move_chips(Position) ->
.
```

Racket:

```
(define/contract (min-cost-to-move-chips position)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Cost to Move Chips to The Same Position
* Difficulty: Easy
* Tags: array, greedy, math
*
* 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 minCostToMoveChips(vector<int>& position) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Cost to Move Chips to The Same Position
* Difficulty: Easy
* Tags: array, greedy, math
*
* 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 minCostToMoveChips(int[] position) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Minimum Cost to Move Chips to The Same Position
Difficulty: Easy
Tags: array, greedy, math

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 minCostToMoveChips(self, position: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Cost to Move Chips to The Same Position
* Difficulty: Easy
* Tags: array, greedy, math
*
* 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[]} position
* @return {number}
*/
var minCostToMoveChips = function(position) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Cost to Move Chips to The Same Position
* Difficulty: Easy
* Tags: array, greedy, math
*
* 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 minCostToMoveChips(position: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Cost to Move Chips to The Same Position
* Difficulty: Easy
* Tags: array, greedy, math
*
* 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 MinCostToMoveChips(int[] position) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Cost to Move Chips to The Same Position
* Difficulty: Easy
* Tags: array, greedy, math
*
* 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 minCostToMoveChips(int* position, int positionSize) {

}
```

Go Solution:

```
// Problem: Minimum Cost to Move Chips to The Same Position
// Difficulty: Easy
// Tags: array, greedy, math
//
// 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 minCostToMoveChips(position []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minCostToMoveChips(position: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minCostToMoveChips(_ position: [Int]) -> Int {
```

```
}
}
```

Rust Solution:

```
// Problem: Minimum Cost to Move Chips to The Same Position
// Difficulty: Easy
// Tags: array, greedy, math
//
// 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_cost_to_move_chips(position: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} position
# @return {Integer}
def min_cost_to_move_chips(position)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $position
* @return Integer
*/
function minCostToMoveChips($position) {

}
}
```

Dart Solution:

```
class Solution {
int minCostToMoveChips(List<int> position) {

}
}
```

Scala Solution:

```
object Solution {
def minCostToMoveChips(position: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_cost_to_move_chips(position :: [integer]) :: integer
def min_cost_to_move_chips(position) do

end
end
```

Erlang Solution:

```
-spec min_cost_to_move_chips(Position :: [integer()]) -> integer().
min_cost_to_move_chips(Position) ->
.
```

Racket Solution:

```
(define/contract (min-cost-to-move-chips position)
(-> (listof exact-integer?) exact-integer?)
)
```