# Problem 3189: Minimum Moves to Get a Peaceful Board

## Problem Information

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

## Problem Description

Given a 2D array

rooks

of length

n

, where

rooks[i] = [x

i

, y

i

]

indicates the position of a rook on an

n x n

chess board. Your task is to move the rooks

1 cell

at a time vertically or horizontally (to an

adjacent

cell) such that the board becomes

peaceful

.

A board is

peaceful

if there is

exactly

one rook in each row and each column.

Return the

minimum

number of moves required to get a

peaceful board

.

Note

that

at no point

can there be two rooks in the same cell.

Example 1:

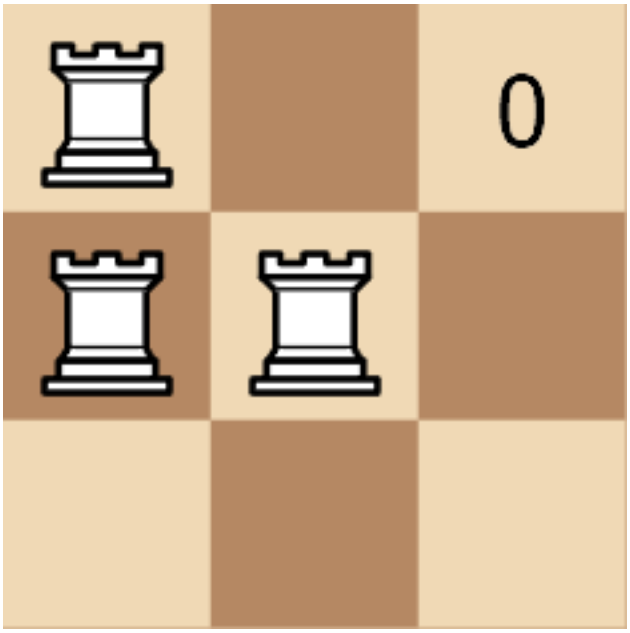
Input:

rooks = [[0,0],[1,0],[1,1]]

Output:

3

Explanation:




Example 2:

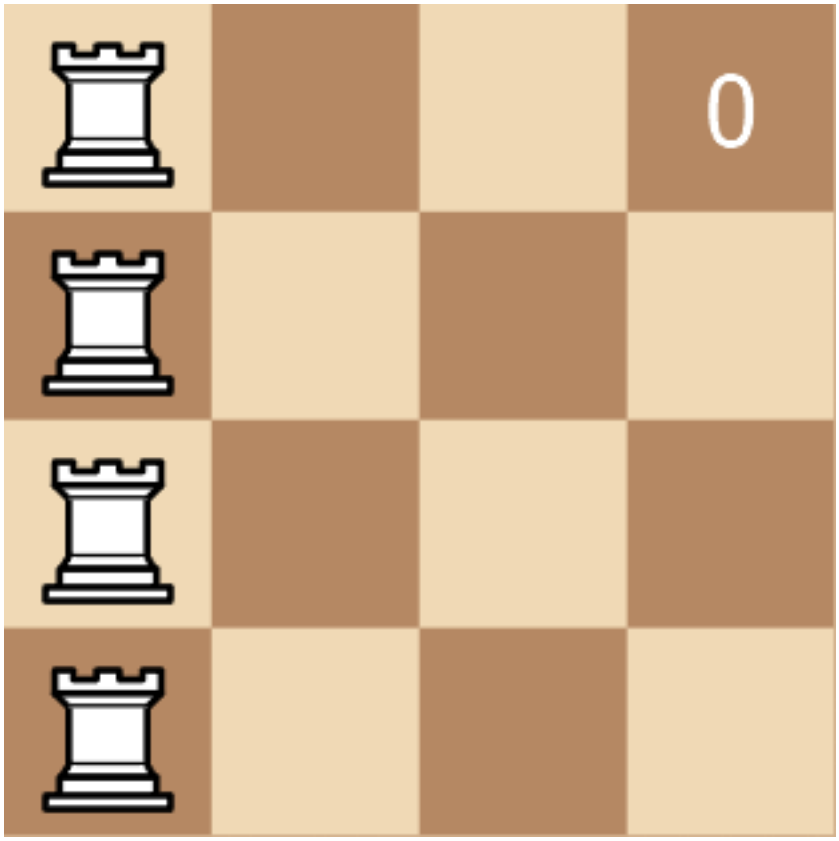
Input:

rooks = [[0,0],[0,1],[0,2],[0,3]]

Output:

6

Explanation:




Constraints:

1 <= n == rooks.length <= 500

0 <= x

i

, y

i

<= n - 1

The input is generated such that there are no 2 rooks in the same cell.

## Code Snippets

**C++:**

```
class Solution {
public:
int minMoves(vector<vector<int>>& rooks) {

}
};
```

**Java:**

```
class Solution {
public int minMoves(int[][] rooks) {

}
}
```

**Python3:**

```
class Solution:
def minMoves(self, rooks: List[List[int]]) -> int:
```

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

```
function minMoves(rooks: number[][]): number {
```

```
};
```

**C#:**

```
public class Solution {
public int MinMoves(int[][] rooks) {

}
}
```

**C:**

```
int minMoves(int** rooks, int rooksSize, int* rooksColSize) {

}
```

**Go:**

```
func minMoves(rooks [][]int) int {

}
```

**Kotlin:**

```
class Solution {
fun minMoves(rooks: Array<IntArray>): Int {

}
}
```

**Swift:**

```
class Solution {
func minMoves(_ rooks: [[Int]]) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn min_moves(rooks: Vec<Vec<i32>>) -> i32 {
```

```
}
}
```

**Ruby:**

```
# @param {Integer[][]} rooks
# @return {Integer}
def min_moves(rooks)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[][] $rooks
* @return Integer
*/
function minMoves($rooks) {

}
}
```

**Dart:**

```
class Solution {
int minMoves(List<List<int>> rooks) {

}
}
```

**Scala:**

```
object Solution {
def minMoves(rooks: Array[Array[Int]]): Int = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec min_moves(rooks :: [[integer]]) :: integer
def min_moves(rooks) do

end
end
```

### Erlang:

```
-spec min_moves(Rooks :: [[integer()]]) -> integer().
min_moves(Rooks) ->
.
```

### Racket:

```
(define/contract (min-moves rooks)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Minimum Moves to Get a Peaceful Board
* 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 minMoves(vector<vector<int>>& rooks) {

}
};
```

### Java Solution:

```
/**
* Problem: Minimum Moves to Get a Peaceful Board
* 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 minMoves(int[][] rooks) {

}
}
```

### Python3 Solution:

```
"""
Problem: Minimum Moves to Get a Peaceful Board
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 minMoves(self, rooks: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Minimum Moves to Get a Peaceful Board
* 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[][]} rooks
* @return {number}
*/
var minMoves = function(rooks) {

};
```

### TypeScript Solution:

```
/**
* Problem: Minimum Moves to Get a Peaceful Board
* 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 minMoves(rooks: number[][]): number {

};
```

### C# Solution:

```
/*
* Problem: Minimum Moves to Get a Peaceful Board
* 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 MinMoves(int[][] rooks) {

}
}
```

### C Solution:

```
/*
* Problem: Minimum Moves to Get a Peaceful Board
* 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 minMoves(int** rooks, int rooksSize, int* rooksColSize) {

}
```

### Go Solution:

```
// Problem: Minimum Moves to Get a Peaceful Board
// 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 minMoves(rooks [][]int) int {

}
```

### Kotlin Solution:

```
class Solution {
fun minMoves(rooks: Array<IntArray>): Int {

}
}
```

### Swift Solution:

```
class Solution {
func minMoves(_ rooks: [[Int]]) -> Int {

}
}
```

### Rust Solution:

```
// Problem: Minimum Moves to Get a Peaceful Board
// 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 min_moves(rooks: Vec<Vec<i32>>) -> i32 {

}
}
```

### Ruby Solution:

```
# @param {Integer[][]} rooks
# @return {Integer}
def min_moves(rooks)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[][] $rooks
* @return Integer
*/
function minMoves($rooks) {

}
}
```

**Dart Solution:**

```
class Solution {
int minMoves(List<List<int>> rooks) {

}
}
```

**Scala Solution:**

```
object Solution {
def minMoves(rooks: Array[Array[Int]]): Int = {

}
}
```

**Elixir Solution:**

```
defmodule Solution do
@spec min_moves(rooks :: [[integer]]) :: integer
def min_moves(rooks) do

end
end
```

**Erlang Solution:**

```
-spec min_moves(Rooks :: [[integer()]]) -> integer().
min_moves(Rooks) ->
.
```

**Racket Solution:**

```
(define/contract (min-moves rooks)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```