Problem 223: Rectangle Area

Problem Information

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

Problem Description

Given the coordinates of two

rectilinear

rectangles in a 2D plane, return

the total area covered by the two rectangles

.

The first rectangle is defined by its

bottom-left

corner

(ax1, ay1)

and its

top-right

corner

(ax2, ay2)

.

The second rectangle is defined by its

bottom-left

corner

(bx1, by1)

and its

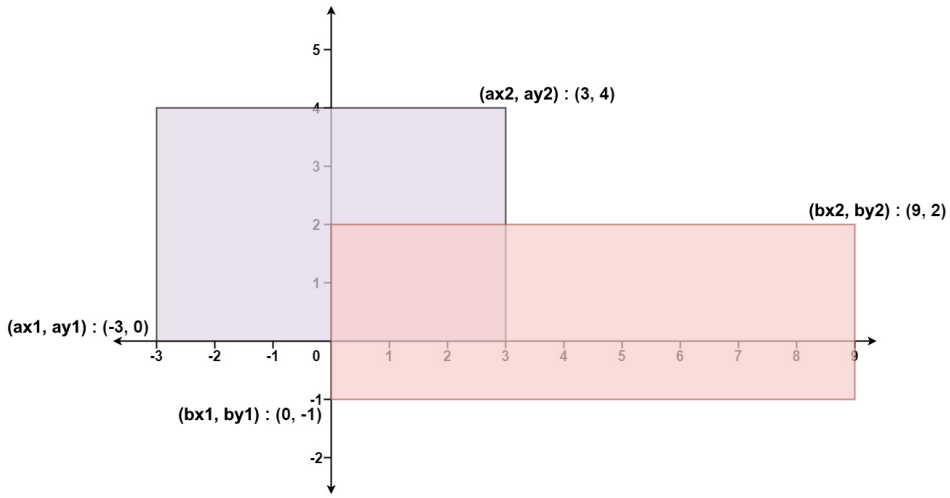
top-right

corner

(bx2, by2)

.

Example 1:



Input:

ax1 = -3, ay1 = 0, ax2 = 3, ay2 = 4, bx1 = 0, by1 = -1, bx2 = 9, by2 = 2

Output:

45

Example 2:

Input:

ax1 = -2, ay1 = -2, ax2 = 2, ay2 = 2, bx1 = -2, by1 = -2, bx2 = 2, by2 = 2

Output:

16

Constraints:

-10

4

<= ax1 <= ax2 <= 10

4

-10

4

<= ay1 <= ay2 <= 10

4

-10

4

<= bx1 <= bx2 <= 10

4

-10

4

<= by1 <= by2 <= 10

4

Code Snippets

C++:

```
class Solution {
public:
int computeArea(int ax1, int ay1, int ax2, int ay2, int bx1, int by1, int
bx2, int by2) {

}
};
```

Java:

```
class Solution {
public int computeArea(int ax1, int ay1, int ax2, int ay2, int bx1, int by1,
int bx2, int by2) {

}
}
```

Python3:

```
class Solution:
def computeArea(self, ax1: int, ay1: int, ax2: int, ay2: int, bx1: int, by1:
int, bx2: int, by2: int) -> int:
```

Python:

```
class Solution(object):
def computeArea(self, ax1, ay1, ax2, ay2, bx1, by1, bx2, by2):
"""
:type ax1: int
```

```
:type ay1: int
:type ax2: int
:type ay2: int
:type bx1: int
:type by1: int
:type bx2: int
:type by2: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} ax1
* @param {number} ay1
* @param {number} ax2
* @param {number} ay2
* @param {number} bx1
* @param {number} by1
* @param {number} bx2
* @param {number} by2
* @return {number}
*/
var computeArea = function(ax1, ay1, ax2, ay2, bx1, by1, bx2, by2) {

};
```

TypeScript:

```
function computeArea(ax1: number, ay1: number, ax2: number, ay2: number, bx1:
number, by1: number, bx2: number, by2: number): number {

};
```

C#:

```
public class Solution {
public int ComputeArea(int ax1, int ay1, int ax2, int ay2, int bx1, int by1,
int bx2, int by2) {

}
}
```

C:

```
int computeArea(int ax1, int ay1, int ax2, int ay2, int bx1, int by1, int
bx2, int by2) {

}
```

Go:

```
func computeArea(ax1 int, ay1 int, ax2 int, ay2 int, bx1 int, by1 int, bx2
int, by2 int) int {

}
```

Kotlin:

```
class Solution {
fun computeArea(ax1: Int, ay1: Int, ax2: Int, ay2: Int, bx1: Int, by1: Int,
bx2: Int, by2: Int): Int {

}
}
```

Swift:

```
class Solution {
func computeArea(_ ax1: Int, _ ay1: Int, _ ax2: Int, _ ay2: Int, _ bx1: Int,
_ by1: Int, _ bx2: Int, _ by2: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn compute_area(ax1: i32, ay1: i32, ax2: i32, ay2: i32, bx1: i32, by1:
i32, bx2: i32, by2: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} ax1
# @param {Integer} ay1
# @param {Integer} ax2
# @param {Integer} ay2
# @param {Integer} bx1
# @param {Integer} by1
# @param {Integer} bx2
# @param {Integer} by2
# @return {Integer}
def compute_area(ax1, ay1, ax2, ay2, bx1, by1, bx2, by2)

end
```

PHP:

```
class Solution {

/**
* @param Integer $ax1
* @param Integer $ay1
* @param Integer $ax2
* @param Integer $ay2
* @param Integer $bx1
* @param Integer $by1
* @param Integer $bx2
* @param Integer $by2
* @return Integer
*/
function computeArea($ax1, $ay1, $ax2, $ay2, $bx1, $by1, $bx2, $by2) {

}
}
```

Dart:

```
class Solution {
int computeArea(int ax1, int ay1, int ax2, int ay2, int bx1, int by1, int
bx2, int by2) {

}
}
```

Scala:

```
object Solution {
def computeArea(ax1: Int, ay1: Int, ax2: Int, ay2: Int, bx1: Int, by1: Int,
bx2: Int, by2: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec compute_area(ax1 :: integer, ay1 :: integer, ax2 :: integer, ay2 ::
integer, bx1 :: integer, by1 :: integer, bx2 :: integer, by2 :: integer) ::
integer
def compute_area(ax1, ay1, ax2, ay2, bx1, by1, bx2, by2) do

end
end
```

Erlang:

```
-spec compute_area(Ax1 :: integer(), Ay1 :: integer(), Ax2 :: integer(), Ay2
:: integer(), Bx1 :: integer(), By1 :: integer(), Bx2 :: integer(), By2 ::
integer()) -> integer().
compute_area(Ax1, Ay1, Ax2, Ay2, Bx1, By1, Bx2, By2) ->
.
```

Racket:

```
(define/contract (compute-area ax1 ay1 ax2 ay2 bx1 by1 bx2 by2)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer? exact-integer? exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Rectangle Area
* Difficulty: Medium
* Tags: math
```

```
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int computeArea(int ax1, int ay1, int ax2, int ay2, int bx1, int by1, int
bx2, int by2) {

}
};
```

Java Solution:

```
/**
* Problem: Rectangle Area
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int computeArea(int ax1, int ay1, int ax2, int ay2, int bx1, int by1,
int bx2, int by2) {

}
}
```

Python3 Solution:

```
"""
Problem: Rectangle Area
Difficulty: Medium
Tags: math

Approach: Optimized algorithm based on problem constraints
```

```
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def computeArea(self, ax1: int, ay1: int, ax2: int, ay2: int, bx1: int, by1:
int, bx2: int, by2: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def computeArea(self, ax1, ay1, ax2, ay2, bx1, by1, bx2, by2):
"""
:type ax1: int
:type ay1: int
:type ax2: int
:type ay2: int
:type bx1: int
:type by1: int
:type bx2: int
:type by2: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Rectangle Area
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} ax1
* @param {number} ay1
```

```
* @param {number} ax2
* @param {number} ay2
* @param {number} bx1
* @param {number} by1
* @param {number} bx2
* @param {number} by2
* @return {number}
*/
var computeArea = function(ax1, ay1, ax2, ay2, bx1, by1, bx2, by2) {

};
```

TypeScript Solution:

```
/**
* Problem: Rectangle Area
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function computeArea(ax1: number, ay1: number, ax2: number, ay2: number, bx1:
number, by1: number, bx2: number, by2: number): number {

};
```

C# Solution:

```
/*
* Problem: Rectangle Area
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
public class Solution {
public int ComputeArea(int ax1, int ay1, int ax2, int ay2, int bx1, int by1,
int bx2, int by2) {

}
}
```

C Solution:

```
/*
* Problem: Rectangle Area
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int computeArea(int ax1, int ay1, int ax2, int ay2, int bx1, int by1, int
bx2, int by2) {

}
```

Go Solution:

```
// Problem: Rectangle Area
// Difficulty: Medium
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func computeArea(ax1 int, ay1 int, ax2 int, ay2 int, bx1 int, by1 int, bx2
int, by2 int) int {

}
```

Kotlin Solution:

```
class Solution {
fun computeArea(ax1: Int, ay1: Int, ax2: Int, ay2: Int, bx1: Int, by1: Int,
bx2: Int, by2: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func computeArea(_ ax1: Int, _ ay1: Int, _ ax2: Int, _ ay2: Int, _ bx1: Int,
_ by1: Int, _ bx2: Int, _ by2: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Rectangle Area
// Difficulty: Medium
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn compute_area(ax1: i32, ay1: i32, ax2: i32, ay2: i32, bx1: i32, by1:
i32, bx2: i32, by2: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} ax1
# @param {Integer} ay1
# @param {Integer} ax2
# @param {Integer} ay2
# @param {Integer} bx1
# @param {Integer} by1
# @param {Integer} bx2
```

```
# @param {Integer} by2
# @return {Integer}
def compute_area(ax1, ay1, ax2, ay2, bx1, by1, bx2, by2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $ax1
* @param Integer $ay1
* @param Integer $ax2
* @param Integer $ay2
* @param Integer $bx1
* @param Integer $by1
* @param Integer $bx2
* @param Integer $by2
* @return Integer
*/
function computeArea($ax1, $ay1, $ax2, $ay2, $bx1, $by1, $bx2, $by2) {

}
}
```

Dart Solution:

```
class Solution {
int computeArea(int ax1, int ay1, int ax2, int ay2, int bx1, int by1, int
bx2, int by2) {

}
}
```

Scala Solution:

```
object Solution {
def computeArea(ax1: Int, ay1: Int, ax2: Int, ay2: Int, bx1: Int, by1: Int,
bx2: Int, by2: Int): Int = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec compute_area(ax1 :: integer, ay1 :: integer, ax2 :: integer, ay2 ::
integer, bx1 :: integer, by1 :: integer, bx2 :: integer, by2 :: integer) ::
integer
def compute_area(ax1, ay1, ax2, ay2, bx1, by1, bx2, by2) do

end
end
```

Erlang Solution:

```
-spec compute_area(Ax1 :: integer(), Ay1 :: integer(), Ax2 :: integer(), Ay2
:: integer(), Bx1 :: integer(), By1 :: integer(), Bx2 :: integer(), By2 ::
integer()) -> integer().
compute_area(Ax1, Ay1, Ax2, Ay2, Bx1, By1, Bx2, By2) ->
.
```

Racket Solution:

```
(define/contract (compute-area ax1 ay1 ax2 ay2 bx1 by1 bx2 by2)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer? exact-integer? exact-integer? exact-integer? exact-integer?)
)
```