

# Problem 304: Range Sum Query 2D - Immutable

## Problem Information

**Difficulty:** Medium

**Acceptance Rate:** 0.00%

**Paid Only:** No

## Problem Description

Given a 2D matrix

matrix

, handle multiple queries of the following type:

Calculate the

sum

of the elements of

matrix

inside the rectangle defined by its

upper left corner

(row1, col1)

and

lower right corner

(row2, col2)

.

Implement the

NumMatrix

class:

NumMatrix(int[][] matrix)

Initializes the object with the integer matrix

matrix

.

int sumRegion(int row1, int col1, int row2, int col2)

Returns the

sum

of the elements of

matrix

inside the rectangle defined by its

upper left corner

(row1, col1)

and

lower right corner

(row2, col2)

.

You must design an algorithm where

sumRegion

works on

$O(1)$

time complexity.

Example 1:

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

Input

```
["NumMatrix", "sumRegion", "sumRegion", "sumRegion"] [[[[[3, 0, 1, 4, 2], [5, 6, 3, 2, 1], [1, 2, 0, 1, 5], [4, 1, 0, 1, 7], [1, 0, 3, 0, 5]]], [2, 1, 4, 3], [1, 1, 2, 2], [1, 2, 2, 4]]
```

Output

[null, 8, 11, 12]

Explanation

```
NumMatrix numMatrix = new NumMatrix([[3, 0, 1, 4, 2], [5, 6, 3, 2, 1], [1, 2, 0, 1, 5], [4, 1, 0, 1, 7], [1, 0, 3, 0, 5]]); numMatrix.sumRegion(2, 1, 4, 3); // return 8 (i.e sum of the red rectangle)
numMatrix.sumRegion(1, 1, 2, 2); // return 11 (i.e sum of the green rectangle)
numMatrix.sumRegion(1, 2, 2, 4); // return 12 (i.e sum of the blue rectangle)
```

Constraints:

$m == \text{matrix.length}$

$n == \text{matrix}[i].\text{length}$

$1 \leq m, n \leq 200$

-10

4

$\leq \text{matrix}[i][j] \leq 10$

4

$0 \leq \text{row1} \leq \text{row2} < m$

$0 \leq \text{col1} \leq \text{col2} < n$

At most

10

4

calls will be made to

sumRegion

.

## Code Snippets

### C++:

```
class NumMatrix {
public:
    NumMatrix(vector<vector<int>>& matrix) {

    }

    int sumRegion(int row1, int col1, int row2, int col2) {

    }
};

/**
 * Your NumMatrix object will be instantiated and called as such:
 * NumMatrix* obj = new NumMatrix(matrix);
 * int param_1 = obj->sumRegion(row1,col1,row2,col2);
 */
```

### Java:

```
class NumMatrix {

    public NumMatrix(int[][] matrix) {

    }

    public int sumRegion(int row1, int col1, int row2, int col2) {

    }
}

/**
 * Your NumMatrix object will be instantiated and called as such:
 * NumMatrix obj = new NumMatrix(matrix);
 */
```

```
* int param_1 = obj.sumRegion(row1,col1,row2,col2);
*/
```

### Python3:

```
class NumMatrix:

    def __init__(self, matrix: List[List[int]]):

    def sumRegion(self, row1: int, col1: int, row2: int, col2: int) -> int:

# Your NumMatrix object will be instantiated and called as such:
# obj = NumMatrix(matrix)
# param_1 = obj.sumRegion(row1,col1,row2,col2)
```

### Python:

```
class NumMatrix(object):

    def __init__(self, matrix):
        """
        :type matrix: List[List[int]]
        """

    def sumRegion(self, row1, col1, row2, col2):
        """
        :type row1: int
        :type col1: int
        :type row2: int
        :type col2: int
        :rtype: int
        """

# Your NumMatrix object will be instantiated and called as such:
# obj = NumMatrix(matrix)
# param_1 = obj.sumRegion(row1,col1,row2,col2)
```

## JavaScript:

```
/**
 * @param {number[][]} matrix
 */
var NumMatrix = function(matrix) {

};

/**
 * @param {number} row1
 * @param {number} col1
 * @param {number} row2
 * @param {number} col2
 * @return {number}
 */
NumMatrix.prototype.sumRegion = function(row1, col1, row2, col2) {

};

/**
 * Your NumMatrix object will be instantiated and called as such:
 * var obj = new NumMatrix(matrix)
 * var param_1 = obj.sumRegion(row1,col1,row2,col2)
 */
```

## TypeScript:

```
class NumMatrix {
  constructor(matrix: number[][]) {

  }

  sumRegion(row1: number, col1: number, row2: number, col2: number): number {

  }
}

/**
 * Your NumMatrix object will be instantiated and called as such:
 * var obj = new NumMatrix(matrix)
 * var param_1 = obj.sumRegion(row1,col1,row2,col2)
 */
```

```
*/
```

## C#:

```
public class NumMatrix {

    public NumMatrix(int[][] matrix) {

    }

    public int SumRegion(int row1, int coll, int row2, int col2) {

    }

}

/**
 * Your NumMatrix object will be instantiated and called as such:
 * NumMatrix obj = new NumMatrix(matrix);
 * int param_1 = obj.SumRegion(row1,coll,row2,col2);
 */
```

## C:

```
typedef struct {

} NumMatrix;

NumMatrix* numMatrixCreate(int** matrix, int matrixSize, int* matrixColSize)
{

}

int numMatrixSumRegion(NumMatrix* obj, int row1, int coll, int row2, int
col2) {

}

void numMatrixFree(NumMatrix* obj) {
```

```

}

/**
 * Your NumMatrix struct will be instantiated and called as such:
 * NumMatrix* obj = numMatrixCreate(matrix, matrixSize, matrixColSize);
 * int param_1 = numMatrixSumRegion(obj, row1, coll, row2, col2);
 *
 * numMatrixFree(obj);
 */

```

## Go:

```

type NumMatrix struct {

}

func Constructor(matrix [][]int) NumMatrix {

}

func (this *NumMatrix) SumRegion(row1 int, coll int, row2 int, col2 int) int
{

}

/**
 * Your NumMatrix object will be instantiated and called as such:
 * obj := Constructor(matrix);
 * param_1 := obj.SumRegion(row1,coll,row2,col2);
 */

```

## Kotlin:

```

class NumMatrix(matrix: Array<IntArray>) {

fun sumRegion(row1: Int, coll: Int, row2: Int, col2: Int): Int {

}

}

```

```

}

/**
 * Your NumMatrix object will be instantiated and called as such:
 * var obj = NumMatrix(matrix)
 * var param_1 = obj.sumRegion(row1,col1,row2,col2)
 */

```

### Swift:

```

class NumMatrix {

    init(_ matrix: [[Int]]) {

    }

    func sumRegion(_ row1: Int, _ col1: Int, _ row2: Int, _ col2: Int) -> Int {

    }

}

/**
 * Your NumMatrix object will be instantiated and called as such:
 * let obj = NumMatrix(matrix)
 * let ret_1: Int = obj.sumRegion(row1, col1, row2, col2)
 */

```

### Rust:

```

struct NumMatrix {

}

/**
 * `&self` means the method takes an immutable reference.
 * If you need a mutable reference, change it to `&mut self` instead.
 */

impl NumMatrix {

```

```

fn new(matrix: Vec<Vec<i32>>) -> Self {

}

fn sum_region(&self, row1: i32, col1: i32, row2: i32, col2: i32) -> i32 {

}
}

/**
 * Your NumMatrix object will be instantiated and called as such:
 * let obj = NumMatrix::new(matrix);
 * let ret_1: i32 = obj.sum_region(row1, col1, row2, col2);
 */

```

## Ruby:

```

class NumMatrix

=begin
:type matrix: Integer[][]
=end
def initialize(matrix)

end

=begin
:type row1: Integer
:type col1: Integer
:type row2: Integer
:type col2: Integer
:rtype: Integer
=end
def sum_region(row1, col1, row2, col2)

end

end

# Your NumMatrix object will be instantiated and called as such:

```

```
# obj = NumMatrix.new(matrix)
# param_1 = obj.sum_region(row1, col1, row2, col2)
```

## PHP:

```
class NumMatrix {
    /**
     * @param Integer[][] $matrix
     */
    function __construct($matrix) {

    }

    /**
     * @param Integer $row1
     * @param Integer $col1
     * @param Integer $row2
     * @param Integer $col2
     * @return Integer
     */
    function sumRegion($row1, $col1, $row2, $col2) {

    }
}

/**
 * Your NumMatrix object will be instantiated and called as such:
 * $obj = NumMatrix($matrix);
 * $ret_1 = $obj->sumRegion($row1, $col1, $row2, $col2);
 */
```

## Dart:

```
class NumMatrix {

    NumMatrix(List<List<int>> matrix) {

    }

    int sumRegion(int row1, int col1, int row2, int col2) {

    }
}
```

```

}

/**
 * Your NumMatrix object will be instantiated and called as such:
 * NumMatrix obj = NumMatrix(matrix);
 * int param1 = obj.sumRegion(row1,col1,row2,col2);
 */

```

## Scala:

```

class NumMatrix(_matrix: Array[Array[Int]]) {

  def sumRegion(row1: Int, col1: Int, row2: Int, col2: Int): Int = {

  }

}

/**
 * Your NumMatrix object will be instantiated and called as such:
 * val obj = new NumMatrix(matrix)
 * val param_1 = obj.sumRegion(row1,col1,row2,col2)
 */

```

## Elixir:

```

defmodule NumMatrix do
  @spec init_(matrix :: [[integer]]) :: any
  def init_(matrix) do

  end

  @spec sum_region(row1 :: integer, col1 :: integer, row2 :: integer, col2 :: integer) :: integer
  def sum_region(row1, col1, row2, col2) do

  end
end

# Your functions will be called as such:
# NumMatrix.init_(matrix)
# param_1 = NumMatrix.sum_region(row1, col1, row2, col2)

```

```
# NumMatrix.init_ will be called before every test case, in which you can do
some necessary initializations.
```

## Erlang:

```
-spec num_matrix_init_(Matrix :: [[integer()]]) -> any().
num_matrix_init_(Matrix) ->
.

-spec num_matrix_sum_region(Row1 :: integer(), Col1 :: integer(), Row2 ::
integer(), Col2 :: integer()) -> integer().
num_matrix_sum_region(Row1, Col1, Row2, Col2) ->
.

%% Your functions will be called as such:
%% num_matrix_init_(Matrix),
%% Param_1 = num_matrix_sum_region(Row1, Col1, Row2, Col2),

%% num_matrix_init_ will be called before every test case, in which you can
do some necessary initializations.
```

## Racket:

```
(define num-matrix%
(class object%
  (super-new)

  ; matrix : (listof (listof exact-integer?))
  (init-field
   matrix)

  ; sum-region : exact-integer? exact-integer? exact-integer? exact-integer? ->
exact-integer?
  (define/public (sum-region row1 col1 row2 col2)
    ))

;; Your num-matrix% object will be instantiated and called as such:
;; (define obj (new num-matrix% [matrix matrix]))
;; (define param_1 (send obj sum-region row1 col1 row2 col2))
```

## Solutions

### C++ Solution:

```
/*
 * Problem: Range Sum Query 2D - Immutable
 * Difficulty: Medium
 * Tags: array
 *
 * 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 NumMatrix {
public:
    NumMatrix(vector<vector<int>>& matrix) {

    }

    int sumRegion(int row1, int col1, int row2, int col2) {

    }
};

/**
 * Your NumMatrix object will be instantiated and called as such:
 * NumMatrix* obj = new NumMatrix(matrix);
 * int param_1 = obj->sumRegion(row1,col1,row2,col2);
 */
```

### Java Solution:

```
/**
 * Problem: Range Sum Query 2D - Immutable
 * Difficulty: Medium
 * Tags: array
 *
 * 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 NumMatrix {

public NumMatrix(int[][] matrix) {

}

public int sumRegion(int row1, int col1, int row2, int col2) {

}

}

/**
 * Your NumMatrix object will be instantiated and called as such:
 * NumMatrix obj = new NumMatrix(matrix);
 * int param_1 = obj.sumRegion(row1,col1,row2,col2);
 */

```

### Python3 Solution:

```

"""
Problem: Range Sum Query 2D - Immutable
Difficulty: Medium
Tags: array

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 NumMatrix:

    def __init__(self, matrix: List[List[int]]):

    def sumRegion(self, row1: int, col1: int, row2: int, col2: int) -> int:
        # TODO: Implement optimized solution
        pass

```

### Python Solution:

```

class NumMatrix(object):

    def __init__(self, matrix):
        """
        :type matrix: List[List[int]]
        """

    def sumRegion(self, row1, col1, row2, col2):
        """
        :type row1: int
        :type col1: int
        :type row2: int
        :type col2: int
        :rtype: int
        """

# Your NumMatrix object will be instantiated and called as such:
# obj = NumMatrix(matrix)
# param_1 = obj.sumRegion(row1,col1,row2,col2)

```

### JavaScript Solution:

```

/**
 * Problem: Range Sum Query 2D - Immutable
 * Difficulty: Medium
 * Tags: array
 *
 * 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[][]} matrix
 */
var NumMatrix = function(matrix) {

};

```

```

/**
 * @param {number} row1
 * @param {number} col1
 * @param {number} row2
 * @param {number} col2
 * @return {number}
 */
NumMatrix.prototype.sumRegion = function(row1, col1, row2, col2) {

};

/**
 * Your NumMatrix object will be instantiated and called as such:
 * var obj = new NumMatrix(matrix)
 * var param_1 = obj.sumRegion(row1,col1,row2,col2)
 */

```

## TypeScript Solution:

```

/**
 * Problem: Range Sum Query 2D - Immutable
 * Difficulty: Medium
 * Tags: array
 *
 * 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 NumMatrix {
  constructor(matrix: number[][]) {

  }

  sumRegion(row1: number, col1: number, row2: number, col2: number): number {

  }
}

/**
 * Your NumMatrix object will be instantiated and called as such:

```

```

* var obj = new NumMatrix(matrix)
* var param_1 = obj.sumRegion(row1,col1,row2,col2)
*/

```

## C# Solution:

```

/*
* Problem: Range Sum Query 2D - Immutable
* Difficulty: Medium
* Tags: array
*
* 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 NumMatrix {

    public NumMatrix(int[][] matrix) {

    }

    public int SumRegion(int row1, int col1, int row2, int col2) {

    }

}

/**
* Your NumMatrix object will be instantiated and called as such:
* NumMatrix obj = new NumMatrix(matrix);
* int param_1 = obj.SumRegion(row1,col1,row2,col2);
*/

```

## C Solution:

```

/*
* Problem: Range Sum Query 2D - Immutable
* Difficulty: Medium
* Tags: array
*
* 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
*/

typedef struct {

} NumMatrix;

NumMatrix* numMatrixCreate(int** matrix, int matrixSize, int* matrixColSize)
{

}

int numMatrixSumRegion(NumMatrix* obj, int row1, int col1, int row2, int
col2) {

}

void numMatrixFree(NumMatrix* obj) {

}

/**
 * Your NumMatrix struct will be instantiated and called as such:
 * NumMatrix* obj = numMatrixCreate(matrix, matrixSize, matrixColSize);
 * int param_1 = numMatrixSumRegion(obj, row1, col1, row2, col2);
 * numMatrixFree(obj);
 */

```

### Go Solution:

```

// Problem: Range Sum Query 2D - Immutable
// Difficulty: Medium
// Tags: array
//
// 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

type NumMatrix struct {

}

func Constructor(matrix [][]int) NumMatrix {

}

func (this *NumMatrix) SumRegion(row1 int, col1 int, row2 int, col2 int) int
{

}

/**
 * Your NumMatrix object will be instantiated and called as such:
 * obj := Constructor(matrix);
 * param_1 := obj.SumRegion(row1,col1,row2,col2);
 */

```

### Kotlin Solution:

```

class NumMatrix(matrix: Array<IntArray>) {

    fun sumRegion(row1: Int, col1: Int, row2: Int, col2: Int): Int {

    }

}

/**
 * Your NumMatrix object will be instantiated and called as such:
 * var obj = NumMatrix(matrix)
 * var param_1 = obj.sumRegion(row1,col1,row2,col2)
 */

```

## Swift Solution:

```
class NumMatrix {

    init(_ matrix: [[Int]]) {

    }

    func sumRegion(_ row1: Int, _ col1: Int, _ row2: Int, _ col2: Int) -> Int {

    }
}

/**
 * Your NumMatrix object will be instantiated and called as such:
 * let obj = NumMatrix(matrix)
 * let ret_1: Int = obj.sumRegion(row1, col1, row2, col2)
 */
```

## Rust Solution:

```
// Problem: Range Sum Query 2D - Immutable
// Difficulty: Medium
// Tags: array
//
// 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

struct NumMatrix {

}

/**
 * `&self` means the method takes an immutable reference.
 * If you need a mutable reference, change it to `&mut self` instead.
 */

impl NumMatrix {

    fn new(matrix: Vec<Vec<i32>>) -> Self {
```

```

}

fn sum_region(&self, row1: i32, col1: i32, row2: i32, col2: i32) -> i32 {

}

}

/**
 * Your NumMatrix object will be instantiated and called as such:
 * let obj = NumMatrix::new(matrix);
 * let ret_1: i32 = obj.sum_region(row1, col1, row2, col2);
 */

```

## Ruby Solution:

```

class NumMatrix

  =begin
  :type matrix: Integer[][]
  =end
  def initialize(matrix)

  end

  =begin
  :type row1: Integer
  :type col1: Integer
  :type row2: Integer
  :type col2: Integer
  :rtype: Integer
  =end
  def sum_region(row1, col1, row2, col2)

  end

end

# Your NumMatrix object will be instantiated and called as such:
# obj = NumMatrix.new(matrix)

```

```
# param_1 = obj.sum_region(row1, col1, row2, col2)
```

### PHP Solution:

```
class NumMatrix {  
    /**  
     * @param Integer[][] $matrix  
     */  
    function __construct($matrix) {  
  
    }  
  
    /**  
     * @param Integer $row1  
     * @param Integer $col1  
     * @param Integer $row2  
     * @param Integer $col2  
     * @return Integer  
     */  
    function sumRegion($row1, $col1, $row2, $col2) {  
  
    }  
}  
  
/**  
 * Your NumMatrix object will be instantiated and called as such:  
 * $obj = NumMatrix($matrix);  
 * $ret_1 = $obj->sumRegion($row1, $col1, $row2, $col2);  
 */
```

### Dart Solution:

```
class NumMatrix {  
  
    NumMatrix(List<List<int>> matrix) {  
  
    }  
  
    int sumRegion(int row1, int col1, int row2, int col2) {  
  
    }  
}
```

```

}

/**
 * Your NumMatrix object will be instantiated and called as such:
 * NumMatrix obj = NumMatrix(matrix);
 * int param1 = obj.sumRegion(row1,col1,row2,col2);
 */

```

### Scala Solution:

```

class NumMatrix(_matrix: Array[Array[Int]]) {

  def sumRegion(row1: Int, col1: Int, row2: Int, col2: Int): Int = {

  }

}

/**
 * Your NumMatrix object will be instantiated and called as such:
 * val obj = new NumMatrix(matrix)
 * val param_1 = obj.sumRegion(row1,col1,row2,col2)
 */

```

### Elixir Solution:

```

defmodule NumMatrix do
  @spec init_(matrix :: [[integer]]) :: any
  def init_(matrix) do

  end

  @spec sum_region(row1 :: integer, col1 :: integer, row2 :: integer, col2 ::
integer) :: integer
  def sum_region(row1, col1, row2, col2) do

  end
end

# Your functions will be called as such:
# NumMatrix.init_(matrix)

```

```
# param_1 = NumMatrix.sum_region(row1, col1, row2, col2)

# NumMatrix.init_ will be called before every test case, in which you can do
some necessary initializations.
```

## Erlang Solution:

```
-spec num_matrix_init_(Matrix :: [[integer()]]) -> any().
num_matrix_init_(Matrix) ->
.

-spec num_matrix_sum_region(Row1 :: integer(), Col1 :: integer(), Row2 ::
integer(), Col2 :: integer()) -> integer().
num_matrix_sum_region(Row1, Col1, Row2, Col2) ->
.

%% Your functions will be called as such:
%% num_matrix_init_(Matrix),
%% Param_1 = num_matrix_sum_region(Row1, Col1, Row2, Col2),

%% num_matrix_init_ will be called before every test case, in which you can
do some necessary initializations.
```

## Racket Solution:

```
(define num-matrix%
(class object%
(super-new)

; matrix : (listof (listof exact-integer?))
(init-field
matrix)

; sum-region : exact-integer? exact-integer? exact-integer? exact-integer? ->
exact-integer?
(define/public (sum-region row1 col1 row2 col2)
)))

;; Your num-matrix% object will be instantiated and called as such:
;; (define obj (new num-matrix% [matrix matrix]))
```

```
;; (define param_1 (send obj sum-region row1 col1 row2 col2))
```