Problem 303: Range Sum Query - Immutable

Problem Information

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

Problem Description

Given an integer array

nums

, handle multiple queries of the following type:

Calculate the

sum

of the elements of

nums

between indices

left

and

right

inclusive

where

left <= right

.

Implement the

NumArray

class:

NumArray(int[] nums)

Initializes the object with the integer array

nums

.

int sumRange(int left, int right)

Returns the

sum

of the elements of

nums

between indices

left

and

right

inclusive

(i.e.

nums[left] + nums[left + 1] + ... + nums[right]

).

Example 1:

Input

["NumArray", "sumRange", "sumRange", "sumRange"] [[[-2, 0, 3, -5, 2, -1]], [0, 2], [2, 5], [0, 5]]

Output

[null, 1, -1, -3]

Explanation

NumArray numArray = new NumArray([-2, 0, 3, -5, 2, -1]); numArray.sumRange(0, 2); // return (-2) + 0 + 3 = 1 numArray.sumRange(2, 5); // return 3 + (-5) + 2 + (-1) = -1 numArray.sumRange(0, 5); // return (-2) + 0 + 3 + (-5) + 2 + (-1) = -3

Constraints:

1 <= nums.length <= 10

4

-10

5

<= nums[i] <= 10

5

0 <= left <= right < nums.length

At most

10

4

calls will be made to

sumRange

.

Code Snippets

C++:

```
class NumArray {
public:
NumArray(vector<int>& nums) {

}

int sumRange(int left, int right) {

}
};

/**
* Your NumArray object will be instantiated and called as such:
* NumArray* obj = new NumArray(nums);
* int param_1 = obj->sumRange(left,right);
*/
```

Java:

```
class NumArray {

public NumArray(int[] nums) {

}

public int sumRange(int left, int right) {

}
}

/**
* Your NumArray object will be instantiated and called as such:
* NumArray obj = new NumArray(nums);
* int param_1 = obj.sumRange(left,right);
*/
```

Python3:

```
class NumArray:

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



def sumRange(self, left: int, right: int) -> int:




# Your NumArray object will be instantiated and called as such:
# obj = NumArray(nums)
# param_1 = obj.sumRange(left,right)
```

Python:

```
class NumArray(object):

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



def sumRange(self, left, right):
"""
:type left: int
:type right: int
:rtype: int
"""
```

```
# Your NumArray object will be instantiated and called as such:
# obj = NumArray(nums)
# param_1 = obj.sumRange(left,right)
```

JavaScript:

```
/**
* @param {number[]} nums
*/
var NumArray = function(nums) {

};

/**
* @param {number} left
* @param {number} right
* @return {number}
*/
NumArray.prototype.sumRange = function(left, right) {

};

/**
* Your NumArray object will be instantiated and called as such:
* var obj = new NumArray(nums)
* var param_1 = obj.sumRange(left,right)
*/
```

TypeScript:

```
class NumArray {
constructor(nums: number[]) {

}

sumRange(left: number, right: number): number {

}
}

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

```
* var obj = new NumArray(nums)
* var param_1 = obj.sumRange(left,right)
*/
```

C#:

```
public class NumArray {

public NumArray(int[] nums) {

}

public int SumRange(int left, int right) {

}
}

/**
* Your NumArray object will be instantiated and called as such:
* NumArray obj = new NumArray(nums);
* int param_1 = obj.SumRange(left,right);
*/
```

C:

```



typedef struct {

} NumArray;


NumArray* numArrayCreate(int* nums, int numsSize) {

}

int numArraySumRange(NumArray* obj, int left, int right) {

}

void numArrayFree(NumArray* obj) {
}

/**
* Your NumArray struct will be instantiated and called as such:
* NumArray* obj = numArrayCreate(nums, numsSize);
* int param_1 = numArraySumRange(obj, left, right);

* numArrayFree(obj);
*/
```

Go:

```
type NumArray struct {

}



func Constructor(nums []int) NumArray {

}



func (this *NumArray) SumRange(left int, right int) int {

}



/**
* Your NumArray object will be instantiated and called as such:
* obj := Constructor(nums);
* param_1 := obj.SumRange(left,right);
*/
```

Kotlin:

```
class NumArray(nums: IntArray) {

fun sumRange(left: Int, right: Int): Int {

}
}

/**
* Your NumArray object will be instantiated and called as such:
* var obj = NumArray(nums)
* var param_1 = obj.sumRange(left,right)
*/
```

Swift:

```
class NumArray {

init(_ nums: [Int]) {

}

func sumRange(_ left: Int, _ right: Int) -> Int {

}
}

/**
* Your NumArray object will be instantiated and called as such:
* let obj = NumArray(nums)
* let ret_1: Int = obj.sumRange(left, right)
*/
```

Rust:

```
struct NumArray {

}



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

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



}

fn sum_range(&self, left: i32, right: i32) -> i32 {

}
}

/**
* Your NumArray object will be instantiated and called as such:
* let obj = NumArray::new(nums);
* let ret_1: i32 = obj.sum_range(left, right);
*/
```

Ruby:

```
class NumArray

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

end


=begin
:type left: Integer
:type right: Integer
:rtype: Integer
=end
def sum_range(left, right)

end


end

# Your NumArray object will be instantiated and called as such:
# obj = NumArray.new(nums)
# param_1 = obj.sum_range(left, right)
```

PHP:

```
class NumArray {
/**
* @param Integer[] $nums
*/
function __construct($nums) {

}

/**
* @param Integer $left
* @param Integer $right
* @return Integer
*/
function sumRange($left, $right) {

}
}

/**
* Your NumArray object will be instantiated and called as such:
* $obj = NumArray($nums);
* $ret_1 = $obj->sumRange($left, $right);
*/
```

Dart:

```
class NumArray {

NumArray(List<int> nums) {

}

int sumRange(int left, int right) {

}
}

/**
* Your NumArray object will be instantiated and called as such:
* NumArray obj = NumArray(nums);
* int param1 = obj.sumRange(left,right);
*/
```

Scala:

```
class NumArray(_nums: Array[Int]) {

def sumRange(left: Int, right: Int): Int = {

}

}

/**
* Your NumArray object will be instantiated and called as such:
* val obj = new NumArray(nums)
* val param_1 = obj.sumRange(left,right)
*/
```

Elixir:

```
defmodule NumArray do
@spec init_(nums :: [integer]) :: any
def init_(nums) do

end

@spec sum_range(left :: integer, right :: integer) :: integer
def sum_range(left, right) do

end
end

# Your functions will be called as such:
# NumArray.init_(nums)
# param_1 = NumArray.sum_range(left, right)

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

Erlang:

```
-spec num_array_init_(Nums :: [integer()]) -> any().
num_array_init_(Nums) ->
.

-spec num_array_sum_range(Left :: integer(), Right :: integer()) ->
integer().
num_array_sum_range(Left, Right) ->
.


%% Your functions will be called as such:
%% num_array_init_(Nums),
%% Param_1 = num_array_sum_range(Left, Right),

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

Racket:

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

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

; sum-range : exact-integer? exact-integer? -> exact-integer?
(define/public (sum-range left right)
)))

;; Your num-array% object will be instantiated and called as such:
;; (define obj (new num-array% [nums nums]))
;; (define param_1 (send obj sum-range left right))
```

Solutions

C++ Solution:

```
/*
* Problem: Range Sum Query - Immutable
* Difficulty: Easy
* 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 NumArray {
public:
NumArray(vector<int>& nums) {

}

int sumRange(int left, int right) {

}
};

/**
* Your NumArray object will be instantiated and called as such:
* NumArray* obj = new NumArray(nums);
* int param_1 = obj->sumRange(left,right);
*/
```

Java Solution:

```
/**
* Problem: Range Sum Query - Immutable
* Difficulty: Easy
* 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 NumArray {

public NumArray(int[] nums) {
}

public int sumRange(int left, int right) {

}
}

/**
* Your NumArray object will be instantiated and called as such:
* NumArray obj = new NumArray(nums);
* int param_1 = obj.sumRange(left,right);
*/
```

Python3 Solution:

```
"""
Problem: Range Sum Query - Immutable
Difficulty: Easy
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 NumArray:

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



def sumRange(self, left: int, right: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class NumArray(object):

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

```
"""



def sumRange(self, left, right):
"""
:type left: int
:type right: int
:rtype: int
"""




# Your NumArray object will be instantiated and called as such:
# obj = NumArray(nums)
# param_1 = obj.sumRange(left,right)
```

JavaScript Solution:

```
/**
* Problem: Range Sum Query - Immutable
* Difficulty: Easy
* 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[]} nums
*/
var NumArray = function(nums) {

};

/**
* @param {number} left
* @param {number} right
* @return {number}
*/
NumArray.prototype.sumRange = function(left, right) {
};

/**
* Your NumArray object will be instantiated and called as such:
* var obj = new NumArray(nums)
* var param_1 = obj.sumRange(left,right)
*/
```

TypeScript Solution:

```
/**
* Problem: Range Sum Query - Immutable
* Difficulty: Easy
* 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 NumArray {
constructor(nums: number[]) {

}

sumRange(left: number, right: number): number {

}
}

/**
* Your NumArray object will be instantiated and called as such:
* var obj = new NumArray(nums)
* var param_1 = obj.sumRange(left,right)
*/
```

C# Solution:

```
/*
* Problem: Range Sum Query - Immutable
* Difficulty: Easy
* 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 NumArray {

public NumArray(int[] nums) {

}

public int SumRange(int left, int right) {

}
}

/**
* Your NumArray object will be instantiated and called as such:
* NumArray obj = new NumArray(nums);
* int param_1 = obj.SumRange(left,right);
*/
```

C Solution:

```
/*
* Problem: Range Sum Query - Immutable
* Difficulty: Easy
* 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 {
} NumArray;


NumArray* numArrayCreate(int* nums, int numsSize) {

}

int numArraySumRange(NumArray* obj, int left, int right) {

}

void numArrayFree(NumArray* obj) {

}

/**
* Your NumArray struct will be instantiated and called as such:
* NumArray* obj = numArrayCreate(nums, numsSize);
* int param_1 = numArraySumRange(obj, left, right);

* numArrayFree(obj);
*/
```

Go Solution:

```go
// Problem: Range Sum Query - Immutable
// Difficulty: Easy
// 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 NumArray struct {

}


func Constructor(nums []int) NumArray {
}


func (this *NumArray) SumRange(left int, right int) int {

}


/**
* Your NumArray object will be instantiated and called as such:
* obj := Constructor(nums);
* param_1 := obj.SumRange(left,right);
*/
```

Kotlin Solution:

```
class NumArray(nums: IntArray) {

fun sumRange(left: Int, right: Int): Int {

}

}

/**
* Your NumArray object will be instantiated and called as such:
* var obj = NumArray(nums)
* var param_1 = obj.sumRange(left,right)
*/
```

Swift Solution:

```

class NumArray {

init(_ nums: [Int]) {

}

func sumRange(_ left: Int, _ right: Int) -> Int {
}
}

/**
* Your NumArray object will be instantiated and called as such:
* let obj = NumArray(nums)
* let ret_1: Int = obj.sumRange(left, right)
*/
```

Rust Solution:

```
// Problem: Range Sum Query - Immutable
// Difficulty: Easy
// 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 NumArray {

}



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

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

}

fn sum_range(&self, left: i32, right: i32) -> i32 {

}
}

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

```
* let obj = NumArray::new(nums);
* let ret_1: i32 = obj.sum_range(left, right);
*/
```

Ruby Solution:

```
class NumArray

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

end



=begin
:type left: Integer
:type right: Integer
:rtype: Integer
=end
def sum_range(left, right)

end



end

# Your NumArray object will be instantiated and called as such:
# obj = NumArray.new(nums)
# param_1 = obj.sum_range(left, right)
```

PHP Solution:

```
class NumArray {
/**
* @param Integer[] $nums
*/
function __construct($nums) {

}
/**
* @param Integer $left
* @param Integer $right
* @return Integer
*/
function sumRange($left, $right) {

}
}

/**
* Your NumArray object will be instantiated and called as such:
* $obj = NumArray($nums);
* $ret_1 = $obj->sumRange($left, $right);
*/
```

Dart Solution:

```
class NumArray {

NumArray(List<int> nums) {

}

int sumRange(int left, int right) {

}
}

/**
* Your NumArray object will be instantiated and called as such:
* NumArray obj = NumArray(nums);
* int param1 = obj.sumRange(left,right);
*/
```

Scala Solution:

```
class NumArray(_nums: Array[Int]) {

def sumRange(left: Int, right: Int): Int = {



}

}

/**
* Your NumArray object will be instantiated and called as such:
* val obj = new NumArray(nums)
* val param_1 = obj.sumRange(left,right)
*/
```

Elixir Solution:

```
defmodule NumArray do
@spec init_(nums :: [integer]) :: any
def init_(nums) do

end

@spec sum_range(left :: integer, right :: integer) :: integer
def sum_range(left, right) do

end
end

# Your functions will be called as such:
# NumArray.init_(nums)
# param_1 = NumArray.sum_range(left, right)

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

Erlang Solution:

```
-spec num_array_init_(Nums :: [integer()]) -> any().
num_array_init_(Nums) ->
.

-spec num_array_sum_range(Left :: integer(), Right :: integer()) ->
integer().
num_array_sum_range(Left, Right) ->
```

```
.



%% Your functions will be called as such:
%% num_array_init_(Nums),
%% Param_1 = num_array_sum_range(Left, Right),

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

Racket Solution:

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

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

; sum-range : exact-integer? exact-integer? -> exact-integer?
(define/public (sum-range left right)
)))

;; Your num-array% object will be instantiated and called as such:
;; (define obj (new num-array% [nums nums]))
;; (define param_1 (send obj sum-range left right))
```