Problem 933: Number of Recent Calls

Problem Information

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

Problem Description

You have a

RecentCounter

class which counts the number of recent requests within a certain time frame.

Implement the

RecentCounter

class:

RecentCounter()

Initializes the counter with zero recent requests.

int ping(int t)

Adds a new request at time

t

, where

t

represents some time in milliseconds, and returns the number of requests that has happened in the past

3000

milliseconds (including the new request). Specifically, return the number of requests that have happened in the inclusive range

[t - 3000, t]

.

It is

guaranteed

that every call to

ping

uses a strictly larger value of

t

than the previous call.

Example 1:

Input

["RecentCounter", "ping", "ping", "ping", "ping"] [[], [1], [100], [3001], [3002]]

Output

[null, 1, 2, 3, 3]

Explanation

RecentCounter recentCounter = new RecentCounter(); recentCounter.ping(1); // requests = [

1

], range is [-2999,1], return 1 recentCounter.ping(100); // requests = [

1

,

100

], range is [-2900,100], return 2 recentCounter.ping(3001); // requests = [

1

,

100

,

3001

], range is [1,3001], return 3 recentCounter.ping(3002); // requests = [1,

100

,

3001

,

3002

], range is [2,3002], return 3

Constraints:

1 <= t <= 10

9

Each test case will call

ping

with

strictly increasing

values of

t

.

At most

10

4

calls will be made to

ping

.

Code Snippets

C++:

```
class RecentCounter {
public:
RecentCounter() {
```

```
}

int ping(int t) {

}
};

/**
* Your RecentCounter object will be instantiated and called as such:
* RecentCounter* obj = new RecentCounter();
* int param_1 = obj->ping(t);
*/
```

Java:

```
class RecentCounter {

public RecentCounter() {

}

public int ping(int t) {

}
}

/**
* Your RecentCounter object will be instantiated and called as such:
* RecentCounter obj = new RecentCounter();
* int param_1 = obj.ping(t);
*/
```

Python3:

```
class RecentCounter:

def __init__(self):


def ping(self, t: int) -> int:
```

```
# Your RecentCounter object will be instantiated and called as such:
# obj = RecentCounter()
# param_1 = obj.ping(t)
```

Python:

```
class RecentCounter(object):

def __init__(self):


def ping(self, t):
"""
:type t: int
:rtype: int
"""



# Your RecentCounter object will be instantiated and called as such:
# obj = RecentCounter()
# param_1 = obj.ping(t)
```

JavaScript:

```
var RecentCounter = function() {

};

/**
* @param {number} t
* @return {number}
*/
RecentCounter.prototype.ping = function(t) {

};

/**
* Your RecentCounter object will be instantiated and called as such:
* var obj = new RecentCounter()
```

```
* var param_1 = obj.ping(t)
*/
```

TypeScript:

```
class RecentCounter {
constructor() {

}

ping(t: number): number {

}
}

/**
* Your RecentCounter object will be instantiated and called as such:
* var obj = new RecentCounter()
* var param_1 = obj.ping(t)
*/
```

C#:

```
public class RecentCounter {

public RecentCounter() {

}

public int Ping(int t) {

}
}

/**
* Your RecentCounter object will be instantiated and called as such:
* RecentCounter obj = new RecentCounter();
* int param_1 = obj.Ping(t);
*/
```

C:

```
typedef struct {

} RecentCounter;


RecentCounter* recentCounterCreate() {

}

int recentCounterPing(RecentCounter* obj, int t) {

}

void recentCounterFree(RecentCounter* obj) {

}

/**
* Your RecentCounter struct will be instantiated and called as such:
* RecentCounter* obj = recentCounterCreate();
* int param_1 = recentCounterPing(obj, t);

* recentCounterFree(obj);
*/
```

Go:

```
type RecentCounter struct {

}


func Constructor() RecentCounter {

}


func (this *RecentCounter) Ping(t int) int {

}
```

```
/**
* Your RecentCounter object will be instantiated and called as such:
* obj := Constructor();
* param_1 := obj.Ping(t);
*/
```

Kotlin:

```
class RecentCounter() {

fun ping(t: Int): Int {

}

}

/**
* Your RecentCounter object will be instantiated and called as such:
* var obj = RecentCounter()
* var param_1 = obj.ping(t)
*/
```

Swift:

```
class RecentCounter {

init() {

}

func ping(_ t: Int) -> Int {

}
}

/**
* Your RecentCounter object will be instantiated and called as such:
* let obj = RecentCounter()
* let ret_1: Int = obj.ping(t)
```

```
*/
```

Rust:

```
struct RecentCounter {

}


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

fn new() -> Self {

}

fn ping(&self, t: i32) -> i32 {

}
}

/**
* Your RecentCounter object will be instantiated and called as such:
* let obj = RecentCounter::new();
* let ret_1: i32 = obj.ping(t);
*/
```

Ruby:

```
class RecentCounter
def initialize()

end


=begin
:type t: Integer
:rtype: Integer
=end
```

```
def ping(t)

end


end

# Your RecentCounter object will be instantiated and called as such:
# obj = RecentCounter.new()
# param_1 = obj.ping(t)
```

PHP:

```
class RecentCounter {
/**
*/
function __construct() {

}

/**
* @param Integer $t
* @return Integer
*/
function ping($t) {

}
}

/**
* Your RecentCounter object will be instantiated and called as such:
* $obj = RecentCounter();
* $ret_1 = $obj->ping($t);
*/
```

Dart:

```
class RecentCounter {

RecentCounter() {

}
```

```
int ping(int t) {

}
}

/**
* Your RecentCounter object will be instantiated and called as such:
* RecentCounter obj = RecentCounter();
* int param1 = obj.ping(t);
*/
```

Scala:

```
class RecentCounter() {

def ping(t: Int): Int = {

}

}

/**
* Your RecentCounter object will be instantiated and called as such:
* val obj = new RecentCounter()
* val param_1 = obj.ping(t)
*/
```

Elixir:

```
defmodule RecentCounter do
@spec init_() :: any
def init_() do

end

@spec ping(t :: integer) :: integer
def ping(t) do

end
end
```

```
# Your functions will be called as such:
# RecentCounter.init_()
# param_1 = RecentCounter.ping(t)

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

Erlang:

```
-spec recent_counter_init_() -> any().
recent_counter_init_() ->
.

-spec recent_counter_ping(T :: integer()) -> integer().
recent_counter_ping(T) ->
.


%% Your functions will be called as such:
%% recent_counter_init_(),
%% Param_1 = recent_counter_ping(T),

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

Racket:

```
(define recent-counter%
(class object%
(super-new)

(init-field)

; ping : exact-integer? -> exact-integer?
(define/public (ping t)
)))

;; Your recent-counter% object will be instantiated and called as such:
;; (define obj (new recent-counter%))
;; (define param_1 (send obj ping t))
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Recent Calls
* Difficulty: Easy
* Tags: queue
*
* 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 RecentCounter {
public:
RecentCounter() {

}

int ping(int t) {

}
};

/**
* Your RecentCounter object will be instantiated and called as such:
* RecentCounter* obj = new RecentCounter();
* int param_1 = obj->ping(t);
*/
```

Java Solution:

```
/**
* Problem: Number of Recent Calls
* Difficulty: Easy
* Tags: queue
*
* 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 RecentCounter {

public RecentCounter() {

}

public int ping(int t) {

}
}

/**
* Your RecentCounter object will be instantiated and called as such:
* RecentCounter obj = new RecentCounter();
* int param_1 = obj.ping(t);
*/
```

Python3 Solution:

```
"""
Problem: Number of Recent Calls
Difficulty: Easy
Tags: queue

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 RecentCounter:

def __init__(self):


def ping(self, t: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class RecentCounter(object):

def __init__(self):


def ping(self, t):
"""
:type t: int
:rtype: int
"""




# Your RecentCounter object will be instantiated and called as such:
# obj = RecentCounter()
# param_1 = obj.ping(t)
```

JavaScript Solution:

```
/**
* Problem: Number of Recent Calls
* Difficulty: Easy
* Tags: queue
*
* 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
*/



var RecentCounter = function() {

};

/**
* @param {number} t
* @return {number}
*/
RecentCounter.prototype.ping = function(t) {

};
```

```
/**
* Your RecentCounter object will be instantiated and called as such:
* var obj = new RecentCounter()
* var param_1 = obj.ping(t)
*/
```

TypeScript Solution:

```
/**
* Problem: Number of Recent Calls
* Difficulty: Easy
* Tags: queue
*
* 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 RecentCounter {
constructor() {

}

ping(t: number): number {

}
}

/**
* Your RecentCounter object will be instantiated and called as such:
* var obj = new RecentCounter()
* var param_1 = obj.ping(t)
*/
```

C# Solution:

```
/*
* Problem: Number of Recent Calls
* Difficulty: Easy
* Tags: queue
*
```

```
* 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 RecentCounter {

public RecentCounter() {

}

public int Ping(int t) {

}
}

/**
* Your RecentCounter object will be instantiated and called as such:
* RecentCounter obj = new RecentCounter();
* int param_1 = obj.Ping(t);
*/
```

C Solution:

```
/*
* Problem: Number of Recent Calls
* Difficulty: Easy
* Tags: queue
*
* 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
*/




typedef struct {

} RecentCounter;
```

```
RecentCounter* recentCounterCreate() {

}

int recentCounterPing(RecentCounter* obj, int t) {

}

void recentCounterFree(RecentCounter* obj) {

}

/**
* Your RecentCounter struct will be instantiated and called as such:
* RecentCounter* obj = recentCounterCreate();
* int param_1 = recentCounterPing(obj, t);

* recentCounterFree(obj);
*/
```

Go Solution:

```
// Problem: Number of Recent Calls
// Difficulty: Easy
// Tags: queue
//
// 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

type RecentCounter struct {

}


func Constructor() RecentCounter {

}
```

```
func (this *RecentCounter) Ping(t int) int {

}


/**
* Your RecentCounter object will be instantiated and called as such:
* obj := Constructor();
* param_1 := obj.Ping(t);
*/
```

Kotlin Solution:

```
class RecentCounter() {

fun ping(t: Int): Int {

}

}

/**
* Your RecentCounter object will be instantiated and called as such:
* var obj = RecentCounter()
* var param_1 = obj.ping(t)
*/
```

Swift Solution:

```
class RecentCounter {

init() {

}

func ping(_ t: Int) -> Int {

}
}
```

```
/**
* Your RecentCounter object will be instantiated and called as such:
* let obj = RecentCounter()
* let ret_1: Int = obj.ping(t)
*/
```

Rust Solution:

```
// Problem: Number of Recent Calls
// Difficulty: Easy
// Tags: queue
//
// 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

struct RecentCounter {

}



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

fn new() -> Self {

}

fn ping(&self, t: i32) -> i32 {

}
}

/**
* Your RecentCounter object will be instantiated and called as such:
* let obj = RecentCounter::new();
* let ret_1: i32 = obj.ping(t);
*/
```

Ruby Solution:

```
class RecentCounter
def initialize()

end


=begin
:type t: Integer
:rtype: Integer
=end
def ping(t)

end



end

# Your RecentCounter object will be instantiated and called as such:
# obj = RecentCounter.new()
# param_1 = obj.ping(t)
```

PHP Solution:

```
class RecentCounter {
/**
*/
function __construct() {

}

/**
* @param Integer $t
* @return Integer
*/
function ping($t) {

}
}

/**
```

```
* Your RecentCounter object will be instantiated and called as such:
* $obj = RecentCounter();
* $ret_1 = $obj->ping($t);
*/
```

Dart Solution:

```
class RecentCounter {

RecentCounter() {

}

int ping(int t) {

}
}

/**
* Your RecentCounter object will be instantiated and called as such:
* RecentCounter obj = RecentCounter();
* int param1 = obj.ping(t);
*/
```

Scala Solution:

```
class RecentCounter() {

def ping(t: Int): Int = {

}

}

/**
* Your RecentCounter object will be instantiated and called as such:
* val obj = new RecentCounter()
* val param_1 = obj.ping(t)
*/
```

Elixir Solution:

```
defmodule RecentCounter do
@spec init_() :: any
def init_() do

end

@spec ping(t :: integer) :: integer
def ping(t) do

end
end

# Your functions will be called as such:
# RecentCounter.init_()
# param_1 = RecentCounter.ping(t)

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

Erlang Solution:

```
-spec recent_counter_init_() -> any().
recent_counter_init_() ->
.

-spec recent_counter_ping(T :: integer()) -> integer().
recent_counter_ping(T) ->
.


%% Your functions will be called as such:
%% recent_counter_init_(),
%% Param_1 = recent_counter_ping(T),

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

Racket Solution:

```
(define recent-counter%
(class object%
(super-new)
```

```
(init-field)

; ping : exact-integer? -> exact-integer?
(define/public (ping t)
)))

;; Your recent-counter% object will be instantiated and called as such:
;; (define obj (new recent-counter%))
;; (define param_1 (send obj ping t))
```