Problem 295: Find Median from Data Stream

Problem Information

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

Problem Description

The

median

is the middle value in an ordered integer list. If the size of the list is even, there is no middle value, and the median is the mean of the two middle values.

For example, for

arr = [2,3,4]

, the median is

3

.

For example, for

arr = [2,3]

, the median is

(2 + 3) / 2 = 2.5

.

Implement the MedianFinder class:

MedianFinder()

initializes the

MedianFinder

object.

void addNum(int num)

adds the integer

num

from the data stream to the data structure.

double findMedian()

returns the median of all elements so far. Answers within

10

-5

of the actual answer will be accepted.

Example 1:

Input

["MedianFinder", "addNum", "addNum", "findMedian", "addNum", "findMedian"] [[], [1], [2], [], [3], []]

Output

[null, null, null, 1.5, null, 2.0]

Explanation

MedianFinder medianFinder = new MedianFinder(); medianFinder.addNum(1); // arr = [1] medianFinder.addNum(2); // arr = [1, 2] medianFinder.findMedian(); // return 1.5 (i.e., (1 + 2) / 2) medianFinder.addNum(3); // arr[1, 2, 3] medianFinder.findMedian(); // return 2.0

Constraints:

-10

5

<= num <= 10

5

There will be at least one element in the data structure before calling

findMedian

.

At most

5 * 10

4

calls will be made to

addNum

and

findMedian

.

Follow up:

If all integer numbers from the stream are in the range

[0, 100]

, how would you optimize your solution?

If

99%

of all integer numbers from the stream are in the range

[0, 100]

, how would you optimize your solution?

Code Snippets

C++:

```
class MedianFinder {
public:
MedianFinder() {

}

void addNum(int num) {

}

double findMedian() {

}
};

/**
* Your MedianFinder object will be instantiated and called as such:
* MedianFinder* obj = new MedianFinder();
* obj->addNum(num);
* double param_2 = obj->findMedian();
*/
```

Java:

```
class MedianFinder {

public MedianFinder() {

}

public void addNum(int num) {

}

public double findMedian() {

}
}

/**
* Your MedianFinder object will be instantiated and called as such:
* MedianFinder obj = new MedianFinder();
* obj.addNum(num);
* double param_2 = obj.findMedian();
*/
```

Python3:

```
class MedianFinder:

def __init__(self):



def addNum(self, num: int) -> None:



def findMedian(self) -> float:




# Your MedianFinder object will be instantiated and called as such:
```

```
# obj = MedianFinder()
# obj.addNum(num)
# param_2 = obj.findMedian()
```

Python:

```
class MedianFinder(object):

def __init__(self):



def addNum(self, num):
"""
:type num: int
:rtype: None
"""



def findMedian(self):
"""
:rtype: float
"""





# Your MedianFinder object will be instantiated and called as such:
# obj = MedianFinder()
# obj.addNum(num)
# param_2 = obj.findMedian()
```

JavaScript:

```
var MedianFinder = function() {

};

/**
* @param {number} num
* @return {void}
*/
MedianFinder.prototype.addNum = function(num) {
};

/**
* @return {number}
*/
MedianFinder.prototype.findMedian = function() {

};

/**
* Your MedianFinder object will be instantiated and called as such:
* var obj = new MedianFinder()
* obj.addNum(num)
* var param_2 = obj.findMedian()
*/
```

TypeScript:

```
class MedianFinder {
constructor() {

}

addNum(num: number): void {

}

findMedian(): number {

}
}

/**
* Your MedianFinder object will be instantiated and called as such:
* var obj = new MedianFinder()
* obj.addNum(num)
* var param_2 = obj.findMedian()
*/
```

C#:

```
public class MedianFinder {

public MedianFinder() {

}

public void AddNum(int num) {

}

public double FindMedian() {

}
}

/**
* Your MedianFinder object will be instantiated and called as such:
* MedianFinder obj = new MedianFinder();
* obj.AddNum(num);
* double param_2 = obj.FindMedian();
*/
```

C:

```



typedef struct {

} MedianFinder;


MedianFinder* medianFinderCreate() {

}

void medianFinderAddNum(MedianFinder* obj, int num) {

}

double medianFinderFindMedian(MedianFinder* obj) {

}
```

```
void medianFinderFree(MedianFinder* obj) {

}

/**
* Your MedianFinder struct will be instantiated and called as such:
* MedianFinder* obj = medianFinderCreate();
* medianFinderAddNum(obj, num);

* double param_2 = medianFinderFindMedian(obj);

* medianFinderFree(obj);
*/
```

Go:

```
type MedianFinder struct {

}


func Constructor() MedianFinder {

}


func (this *MedianFinder) AddNum(num int) {

}


func (this *MedianFinder) FindMedian() float64 {

}


/**
* Your MedianFinder object will be instantiated and called as such:
* obj := Constructor();
* obj.AddNum(num);
* param_2 := obj.FindMedian();
```

```
*/
```

Kotlin:

```
class MedianFinder() {

fun addNum(num: Int) {

}

fun findMedian(): Double {

}

}

/**
* Your MedianFinder object will be instantiated and called as such:
* var obj = MedianFinder()
* obj.addNum(num)
* var param_2 = obj.findMedian()
*/
```

Swift:

```

class MedianFinder {

init() {

}

func addNum(_ num: Int) {

}

func findMedian() -> Double {

}
}

/**
```

```
* Your MedianFinder object will be instantiated and called as such:
* let obj = MedianFinder()
* obj.addNum(num)
* let ret_2: Double = obj.findMedian()
*/
```

Rust:

```
struct MedianFinder {

}



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

fn new() -> Self {

}

fn add_num(&self, num: i32) {

}

fn find_median(&self) -> f64 {

}
}

/**
* Your MedianFinder object will be instantiated and called as such:
* let obj = MedianFinder::new();
* obj.add_num(num);
* let ret_2: f64 = obj.find_median();
*/
```

Ruby:

```
class MedianFinder
def initialize()

end



=begin
:type num: Integer
:rtype: Void
=end
def add_num(num)

end



=begin
:rtype: Float
=end
def find_median()

end



end

# Your MedianFinder object will be instantiated and called as such:
# obj = MedianFinder.new()
# obj.add_num(num)
# param_2 = obj.find_median()
```

PHP:

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

}

/**
* @param Integer $num
* @return NULL
*/
function addNum($num) {

}

/**
* @return Float
*/
function findMedian() {

}
}

/**
* Your MedianFinder object will be instantiated and called as such:
* $obj = MedianFinder();
* $obj->addNum($num);
* $ret_2 = $obj->findMedian();
*/
```

Dart:

```
class MedianFinder {

MedianFinder() {

}

void addNum(int num) {

}

double findMedian() {

}
}

/**
* Your MedianFinder object will be instantiated and called as such:
* MedianFinder obj = MedianFinder();
* obj.addNum(num);
* double param2 = obj.findMedian();
*/
```

Scala:

```
class MedianFinder() {

def addNum(num: Int): Unit = {

}

def findMedian(): Double = {

}

}

/**
* Your MedianFinder object will be instantiated and called as such:
* val obj = new MedianFinder()
* obj.addNum(num)
* val param_2 = obj.findMedian()
*/
```

Elixir:

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

end

@spec add_num(num :: integer) :: any
def add_num(num) do

end

@spec find_median() :: float
def find_median() do

end
end

# Your functions will be called as such:
# MedianFinder.init_()
# MedianFinder.add_num(num)
# param_2 = MedianFinder.find_median()

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

Erlang:

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

-spec median_finder_add_num(Num :: integer()) -> any().
median_finder_add_num(Num) ->
.

-spec median_finder_find_median() -> float().
median_finder_find_median() ->
.



%% Your functions will be called as such:
%% median_finder_init_(),
%% median_finder_add_num(Num),
%% Param_2 = median_finder_find_median(),

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

Racket:

```
(define median-finder%
(class object%
(super-new)

(init-field)

; add-num : exact-integer? -> void?
(define/public (add-num num)
)
; find-median : -> flonum?
(define/public (find-median)
)))

;; Your median-finder% object will be instantiated and called as such:
;; (define obj (new median-finder%))
;; (send obj add-num num)
;; (define param_2 (send obj find-median))
```

Solutions

C++ Solution:

```
/*
* Problem: Find Median from Data Stream
* Difficulty: Hard
* Tags: array, sort, queue, heap
*
* 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 MedianFinder {
public:
MedianFinder() {

}

void addNum(int num) {

}

double findMedian() {

}
};

/**
* Your MedianFinder object will be instantiated and called as such:
* MedianFinder* obj = new MedianFinder();
* obj->addNum(num);
* double param_2 = obj->findMedian();
*/
```

Java Solution:

```
/**
* Problem: Find Median from Data Stream
* Difficulty: Hard
* Tags: array, sort, queue, heap
*
* 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 MedianFinder {

public MedianFinder() {

}

public void addNum(int num) {

}

public double findMedian() {

}
}

/**
* Your MedianFinder object will be instantiated and called as such:
* MedianFinder obj = new MedianFinder();
* obj.addNum(num);
* double param_2 = obj.findMedian();
*/
```

Python3 Solution:

```
"""
Problem: Find Median from Data Stream
```

```
Difficulty: Hard
Tags: array, sort, queue, heap

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 MedianFinder:

def __init__(self):



def addNum(self, num: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class MedianFinder(object):

def __init__(self):



def addNum(self, num):
"""
:type num: int
:rtype: None
"""



def findMedian(self):
"""
:rtype: float
"""




# Your MedianFinder object will be instantiated and called as such:
# obj = MedianFinder()
# obj.addNum(num)
# param_2 = obj.findMedian()
```

JavaScript Solution:

```
/**
* Problem: Find Median from Data Stream
* Difficulty: Hard
* Tags: array, sort, queue, heap
*
* 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
*/


var MedianFinder = function() {

};

/**
* @param {number} num
* @return {void}
*/
MedianFinder.prototype.addNum = function(num) {

};

/**
* @return {number}
*/
MedianFinder.prototype.findMedian = function() {

};

/**
* Your MedianFinder object will be instantiated and called as such:
* var obj = new MedianFinder()
* obj.addNum(num)
* var param_2 = obj.findMedian()
*/
```

TypeScript Solution:

```
/**
* Problem: Find Median from Data Stream
* Difficulty: Hard
* Tags: array, sort, queue, heap
*
* 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 MedianFinder {
constructor() {

}

addNum(num: number): void {

}

findMedian(): number {

}
}

/**
* Your MedianFinder object will be instantiated and called as such:
* var obj = new MedianFinder()
* obj.addNum(num)
* var param_2 = obj.findMedian()
*/
```

C# Solution:

```
/*
* Problem: Find Median from Data Stream
* Difficulty: Hard
* Tags: array, sort, queue, heap
*
* 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 MedianFinder {

public MedianFinder() {

}

public void AddNum(int num) {

}

public double FindMedian() {

}
}

/**
* Your MedianFinder object will be instantiated and called as such:
* MedianFinder obj = new MedianFinder();
* obj.AddNum(num);
* double param_2 = obj.FindMedian();
*/
```

C Solution:

```
/*
* Problem: Find Median from Data Stream
* Difficulty: Hard
* Tags: array, sort, queue, heap
*
* 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 {
} MedianFinder;


MedianFinder* medianFinderCreate() {

}

void medianFinderAddNum(MedianFinder* obj, int num) {

}

double medianFinderFindMedian(MedianFinder* obj) {

}

void medianFinderFree(MedianFinder* obj) {

}

/**
* Your MedianFinder struct will be instantiated and called as such:
* MedianFinder* obj = medianFinderCreate();
* medianFinderAddNum(obj, num);

* double param_2 = medianFinderFindMedian(obj);

* medianFinderFree(obj);
*/
```

Go Solution:

```
// Problem: Find Median from Data Stream
// Difficulty: Hard
// Tags: array, sort, queue, heap
//
// 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 MedianFinder struct {
}


func Constructor() MedianFinder {

}


func (this *MedianFinder) AddNum(num int) {

}


func (this *MedianFinder) FindMedian() float64 {

}


/**
* Your MedianFinder object will be instantiated and called as such:
* obj := Constructor();
* obj.AddNum(num);
* param_2 := obj.FindMedian();
*/
```

Kotlin Solution:

```
class MedianFinder() {

fun addNum(num: Int) {

}

fun findMedian(): Double {

}

}

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

```
* var obj = MedianFinder()
* obj.addNum(num)
* var param_2 = obj.findMedian()
*/
```

Swift Solution:

```
class MedianFinder {

init() {

}

func addNum(_ num: Int) {

}

func findMedian() -> Double {

}
}

/**
* Your MedianFinder object will be instantiated and called as such:
* let obj = MedianFinder()
* obj.addNum(num)
* let ret_2: Double = obj.findMedian()
*/
```

Rust Solution:

```
// Problem: Find Median from Data Stream
// Difficulty: Hard
// Tags: array, sort, queue, heap
//
// 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 MedianFinder {
}



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

fn new() -> Self {

}

fn add_num(&self, num: i32) {

}

fn find_median(&self) -> f64 {

}
}

/**
* Your MedianFinder object will be instantiated and called as such:
* let obj = MedianFinder::new();
* obj.add_num(num);
* let ret_2: f64 = obj.find_median();
*/
```

Ruby Solution:

```
class MedianFinder
def initialize()

end



=begin
:type num: Integer
:rtype: Void
```

```
=end
def add_num(num)

end



=begin
:rtype: Float
=end
def find_median()

end



end

# Your MedianFinder object will be instantiated and called as such:
# obj = MedianFinder.new()
# obj.add_num(num)
# param_2 = obj.find_median()
```

PHP Solution:

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

}

/**
* @param Integer $num
* @return NULL
*/
function addNum($num) {

}

/**
* @return Float
*/
function findMedian() {

}
}

/**
* Your MedianFinder object will be instantiated and called as such:
* $obj = MedianFinder();
* $obj->addNum($num);
* $ret_2 = $obj->findMedian();
*/
```

Dart Solution:

```
class MedianFinder {

MedianFinder() {

}

void addNum(int num) {

}

double findMedian() {

}
}

/**
* Your MedianFinder object will be instantiated and called as such:
* MedianFinder obj = MedianFinder();
* obj.addNum(num);
* double param2 = obj.findMedian();
*/
```

Scala Solution:

```
class MedianFinder() {

def addNum(num: Int): Unit = {


}

def findMedian(): Double = {

}

}

/**
* Your MedianFinder object will be instantiated and called as such:
* val obj = new MedianFinder()
* obj.addNum(num)
* val param_2 = obj.findMedian()
*/
```

Elixir Solution:

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

end

@spec add_num(num :: integer) :: any
def add_num(num) do

end

@spec find_median() :: float
def find_median() do

end
end

# Your functions will be called as such:
# MedianFinder.init_()
# MedianFinder.add_num(num)
# param_2 = MedianFinder.find_median()

# MedianFinder.init_ will be called before every test case, in which you can
```

```
do some necessary initializations.
```

Erlang Solution:

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

-spec median_finder_add_num(Num :: integer()) -> any().
median_finder_add_num(Num) ->
.

-spec median_finder_find_median() -> float().
median_finder_find_median() ->
.



%% Your functions will be called as such:
%% median_finder_init_(),
%% median_finder_add_num(Num),
%% Param_2 = median_finder_find_median(),

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

Racket Solution:

```
(define median-finder%
(class object%
(super-new)

(init-field)

; add-num : exact-integer? -> void?
(define/public (add-num num)
)
; find-median : -> flonum?
(define/public (find-median)
)))

;; Your median-finder% object will be instantiated and called as such:
;; (define obj (new median-finder%))
;; (send obj add-num num)
;; (define param_2 (send obj find-median))
```