# Problem 170: Two Sum III - Data structure design

## Problem Information

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

## Problem Description

Design a data structure that accepts a stream of integers and checks if it has a pair of integers that sum up to a particular value.

Implement the

TwoSum

class:

TwoSum()

Initializes the

TwoSum

object, with an empty array initially.

void add(int number)

Adds

number

to the data structure.

boolean find(int value)

Returns

true

if there exists any pair of numbers whose sum is equal to

value

, otherwise, it returns

false

.

Example 1:

Input

["TwoSum", "add", "add", "add", "find", "find"] [[], [1], [3], [5], [4], [7]]

Output

[null, null, null, null, true, false]

Explanation

TwoSum twoSum = new TwoSum(); twoSum.add(1); // [] --> [1] twoSum.add(3); // [1] --> [1,3] twoSum.add(5); // [1,3] --> [1,3,5] twoSum.find(4); // 1 + 3 = 4, return true twoSum.find(7); // No two integers sum up to 7, return false

Constraints:

-10

5

<= number <= 10

5

-2

31

<= value <= 2

31

- 1

At most

10

4

calls will be made to

add

and

find

.

## Code Snippets

**C++:**

```
class TwoSum {
public:
TwoSum() {

}
```

```
void add(int number) {

}

bool find(int value) {

}
};

/**
* Your TwoSum object will be instantiated and called as such:
* TwoSum* obj = new TwoSum();
* obj->add(number);
* bool param_2 = obj->find(value);
*/
```

**Java:**

```
class TwoSum {

public TwoSum() {

}

public void add(int number) {

}

public boolean find(int value) {

}
}

/**
* Your TwoSum object will be instantiated and called as such:
* TwoSum obj = new TwoSum();
* obj.add(number);
* boolean param_2 = obj.find(value);
*/
```

**Python3:**

```
class TwoSum:

def __init__(self):


def add(self, number: int) -> None:


def find(self, value: int) -> bool:



# Your TwoSum object will be instantiated and called as such:
# obj = TwoSum()
# obj.add(number)
# param_2 = obj.find(value)
```

**Python:**

```
class TwoSum(object):

def __init__(self):


def add(self, number):
"""
:type number: int
:rtype: None
"""


def find(self, value):
"""
:type value: int
:rtype: bool
"""



# Your TwoSum object will be instantiated and called as such:
# obj = TwoSum()
# obj.add(number)
# param_2 = obj.find(value)
```

**JavaScript:**

```
var TwoSum = function() {

};

/**
* @param {number} number
* @return {void}
*/
TwoSum.prototype.add = function(number) {

};

/**
* @param {number} value
* @return {boolean}
*/
TwoSum.prototype.find = function(value) {

};

/**
* Your TwoSum object will be instantiated and called as such:
* var obj = new TwoSum()
* obj.add(number)
* var param_2 = obj.find(value)
*/
```

**TypeScript:**

```
class TwoSum {
constructor() {

}

add(number: number): void {

}

find(value: number): boolean {
```

```
}
}

/**
* Your TwoSum object will be instantiated and called as such:
* var obj = new TwoSum()
* obj.add(number)
* var param_2 = obj.find(value)
*/
```

**C#:**

```
public class TwoSum {

public TwoSum() {

}

public void Add(int number) {

}

public bool Find(int value) {

}
}

/**
* Your TwoSum object will be instantiated and called as such:
* TwoSum obj = new TwoSum();
* obj.Add(number);
* bool param_2 = obj.Find(value);
*/
```

**C:**

```
typedef struct {
```

```
} TwoSum;


TwoSum* twoSumCreate() {

}

void twoSumAdd(TwoSum* obj, int number) {

}

bool twoSumFind(TwoSum* obj, int value) {

}

void twoSumFree(TwoSum* obj) {

}

/**
* Your TwoSum struct will be instantiated and called as such:
* TwoSum* obj = twoSumCreate();
* twoSumAdd(obj, number);

* bool param_2 = twoSumFind(obj, value);

* twoSumFree(obj);
*/
```

**Go:**

```
type TwoSum struct {

}


func Constructor() TwoSum {

}


func (this *TwoSum) Add(number int) {
```

```
}


func (this *TwoSum) Find(value int) bool {

}


/**
* Your TwoSum object will be instantiated and called as such:
* obj := Constructor();
* obj.Add(number);
* param_2 := obj.Find(value);
*/
```

**Kotlin:**

```
class TwoSum() {

fun add(number: Int) {

}

fun find(value: Int): Boolean {

}

}

/**
* Your TwoSum object will be instantiated and called as such:
* var obj = TwoSum()
* obj.add(number)
* var param_2 = obj.find(value)
*/
```

**Swift:**

```
class TwoSum {
```

```
init() {

}

func add(_ number: Int) {

}

func find(_ value: Int) -> Bool {

}
}

/**
* Your TwoSum object will be instantiated and called as such:
* let obj = TwoSum()
* obj.add(number)
* let ret_2: Bool = obj.find(value)
*/
```

**Rust:**

```
struct TwoSum {

}


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

fn new() -> Self {

}

fn add(&self, number: i32) {

}

fn find(&self, value: i32) -> bool {
```

```
}
}

/**
* Your TwoSum object will be instantiated and called as such:
* let obj = TwoSum::new();
* obj.add(number);
* let ret_2: bool = obj.find(value);
*/
```

**Ruby:**

```
class TwoSum
def initialize()

end


=begin
:type number: Integer
:rtype: Void
=end
def add(number)

end


=begin
:type value: Integer
:rtype: Boolean
=end
def find(value)

end


end

# Your TwoSum object will be instantiated and called as such:
# obj = TwoSum.new()
# obj.add(number)
```

```
# param_2 = obj.find(value)
```

**PHP:**

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

}

/**
* @param Integer $number
* @return NULL
*/
function add($number) {

}

/**
* @param Integer $value
* @return Boolean
*/
function find($value) {

}
}

/**
* Your TwoSum object will be instantiated and called as such:
* $obj = TwoSum();
* $obj->add($number);
* $ret_2 = $obj->find($value);
*/
```

**Dart:**

```
class TwoSum {

TwoSum() {

}
```

```
void add(int number) {

}

bool find(int value) {

}
}

/**
* Your TwoSum object will be instantiated and called as such:
* TwoSum obj = TwoSum();
* obj.add(number);
* bool param2 = obj.find(value);
*/
```

**Scala:**

```
class TwoSum() {

def add(number: Int): Unit = {

}

def find(value: Int): Boolean = {

}

}

/**
* Your TwoSum object will be instantiated and called as such:
* val obj = new TwoSum()
* obj.add(number)
* val param_2 = obj.find(value)
*/
```

**Elixir:**

```
defmodule TwoSum do
@spec init_() :: any
```

```
def init_() do

end

@spec add(number :: integer) :: any
def add(number) do

end

@spec find(value :: integer) :: boolean
def find(value) do

end
end

# Your functions will be called as such:
# TwoSum.init_()
# TwoSum.add(number)
# param_2 = TwoSum.find(value)

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

**Erlang:**

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

-spec two_sum_add(Number :: integer()) -> any().
two_sum_add(Number) ->
.

-spec two_sum_find(Value :: integer()) -> boolean().
two_sum_find(Value) ->
.


%% Your functions will be called as such:
%% two_sum_init_(),
%% two_sum_add(Number),
%% Param_2 = two_sum_find(Value),
```

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

**Racket:**

```
(define two-sum%
(class object%
(super-new)

(init-field)

; add : exact-integer? -> void?
(define/public (add number)
)
; find : exact-integer? -> boolean?
(define/public (find value)
)))

;; Your two-sum% object will be instantiated and called as such:
;; (define obj (new two-sum%))
;; (send obj add number)
;; (define param_2 (send obj find value))
```

## Solutions

### C++ Solution:

```
/*
* Problem: Two Sum III - Data structure design
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class TwoSum {
public:
```

```
TwoSum() {

}

void add(int number) {

}

bool find(int value) {

}
};

/**
* Your TwoSum object will be instantiated and called as such:
* TwoSum* obj = new TwoSum();
* obj->add(number);
* bool param_2 = obj->find(value);
*/
```

**Java Solution:**

```
/**
* Problem: Two Sum III - Data structure design
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class TwoSum {

public TwoSum() {

}

public void add(int number) {

}
```

```
public boolean find(int value) {

}
}

/**
* Your TwoSum object will be instantiated and called as such:
* TwoSum obj = new TwoSum();
* obj.add(number);
* boolean param_2 = obj.find(value);
*/
```

### Python3 Solution:

```
"""
Problem: Two Sum III - Data structure design
Difficulty: Easy
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class TwoSum:

def __init__(self):


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

### Python Solution:

```
class TwoSum(object):

def __init__(self):
```

```
def add(self, number):
"""
:type number: int
:rtype: None
"""


def find(self, value):
"""
:type value: int
:rtype: bool
"""



# Your TwoSum object will be instantiated and called as such:
# obj = TwoSum()
# obj.add(number)
# param_2 = obj.find(value)
```

**JavaScript Solution:**

```
/**
* Problem: Two Sum III - Data structure design
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/


var TwoSum = function() {

};

/**
* @param {number} number
* @return {void}
*/
```

```
TwoSum.prototype.add = function(number) {

};

/**
* @param {number} value
* @return {boolean}
*/
TwoSum.prototype.find = function(value) {

};

/**
* Your TwoSum object will be instantiated and called as such:
* var obj = new TwoSum()
* obj.add(number)
* var param_2 = obj.find(value)
*/
```

**TypeScript Solution:**

```
/**
* Problem: Two Sum III - Data structure design
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class TwoSum {
constructor() {

}

add(number: number): void {

}

find(value: number): boolean {
```

```
}
}

/**
* Your TwoSum object will be instantiated and called as such:
* var obj = new TwoSum()
* obj.add(number)
* var param_2 = obj.find(value)
*/
```

**C# Solution:**

```
/*
* Problem: Two Sum III - Data structure design
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class TwoSum {

public TwoSum() {

}

public void Add(int number) {

}

public bool Find(int value) {

}
}

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

```
* obj.Add(number);
* bool param_2 = obj.Find(value);
*/
```

### C Solution:

```
/*
* Problem: Two Sum III - Data structure design
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/




typedef struct {

} TwoSum;


TwoSum* twoSumCreate() {

}

void twoSumAdd(TwoSum* obj, int number) {

}

bool twoSumFind(TwoSum* obj, int value) {

}

void twoSumFree(TwoSum* obj) {

}

/**
```

```
* Your TwoSum struct will be instantiated and called as such:
* TwoSum* obj = twoSumCreate();
* twoSumAdd(obj, number);

* bool param_2 = twoSumFind(obj, value);

* twoSumFree(obj);
*/
```

### Go Solution:

```
// Problem: Two Sum III - Data structure design
// Difficulty: Easy
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

type TwoSum struct {

}


func Constructor() TwoSum {

}


func (this *TwoSum) Add(number int) {

}


func (this *TwoSum) Find(value int) bool {

}


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

```
* obj := Constructor();
* obj.Add(number);
* param_2 := obj.Find(value);
*/
```

**Kotlin Solution:**

```
class TwoSum() {

fun add(number: Int) {

}

fun find(value: Int): Boolean {

}

}

/**
* Your TwoSum object will be instantiated and called as such:
* var obj = TwoSum()
* obj.add(number)
* var param_2 = obj.find(value)
*/
```

**Swift Solution:**

```
class TwoSum {

init() {

}

func add(_ number: Int) {

}

func find(_ value: Int) -> Bool {
```

```
}
}

/**
* Your TwoSum object will be instantiated and called as such:
* let obj = TwoSum()
* obj.add(number)
* let ret_2: Bool = obj.find(value)
*/
```

**Rust Solution:**

```
// Problem: Two Sum III - Data structure design
// Difficulty: Easy
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

struct TwoSum {

}



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

fn new() -> Self {

}

fn add(&self, number: i32) {

}

fn find(&self, value: i32) -> bool {
```

```
}
}

/**
* Your TwoSum object will be instantiated and called as such:
* let obj = TwoSum::new();
* obj.add(number);
* let ret_2: bool = obj.find(value);
*/
```

### Ruby Solution:

```
class TwoSum
def initialize()

end


=begin
:type number: Integer
:rtype: Void
=end
def add(number)

end


=begin
:type value: Integer
:rtype: Boolean
=end
def find(value)

end


end

# Your TwoSum object will be instantiated and called as such:
# obj = TwoSum.new()
# obj.add(number)
```

```
# param_2 = obj.find(value)
```

**PHP Solution:**

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

}

/**
* @param Integer $number
* @return NULL
*/
function add($number) {

}

/**
* @param Integer $value
* @return Boolean
*/
function find($value) {

}
}

/**
* Your TwoSum object will be instantiated and called as such:
* $obj = TwoSum();
* $obj->add($number);
* $ret_2 = $obj->find($value);
*/
```

**Dart Solution:**

```
class TwoSum {

TwoSum() {
```

```
}

void add(int number) {

}

bool find(int value) {

}
}

/**
* Your TwoSum object will be instantiated and called as such:
* TwoSum obj = TwoSum();
* obj.add(number);
* bool param2 = obj.find(value);
*/
```

**Scala Solution:**

```
class TwoSum() {

def add(number: Int): Unit = {

}

def find(value: Int): Boolean = {

}

}

/**
* Your TwoSum object will be instantiated and called as such:
* val obj = new TwoSum()
* obj.add(number)
* val param_2 = obj.find(value)
*/
```

**Elixir Solution:**

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

end

@spec add(number :: integer) :: any
def add(number) do

end

@spec find(value :: integer) :: boolean
def find(value) do

end
end

# Your functions will be called as such:
# TwoSum.init_()
# TwoSum.add(number)
# param_2 = TwoSum.find(value)

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

### Erlang Solution:

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

-spec two_sum_add(Number :: integer()) -> any().
two_sum_add(Number) ->
.

-spec two_sum_find(Value :: integer()) -> boolean().
two_sum_find(Value) ->
.


%% Your functions will be called as such:
%% two_sum_init_(),
```

```
%% two_sum_add(Number),
%% Param_2 = two_sum_find(Value),

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

### Racket Solution:

```
(define two-sum%
(class object%
(super-new)

(init-field)

; add : exact-integer? -> void?
(define/public (add number)
)
; find : exact-integer? -> boolean?
(define/public (find value)
)))

;; Your two-sum% object will be instantiated and called as such:
;; (define obj (new two-sum%))
;; (send obj add number)
;; (define param_2 (send obj find value))
```