Problem 335: Self Crossing

Problem Information

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

Problem Description

You are given an array of integers

distance

.

You start at the point

(0, 0)

on an

X-Y plane,

and you move

distance[0]

meters to the north, then

distance[1]

meters to the west,

distance[2]

meters to the south,

distance[3]

meters to the east, and so on. In other words, after each move, your direction changes counter-clockwise.

Return

true

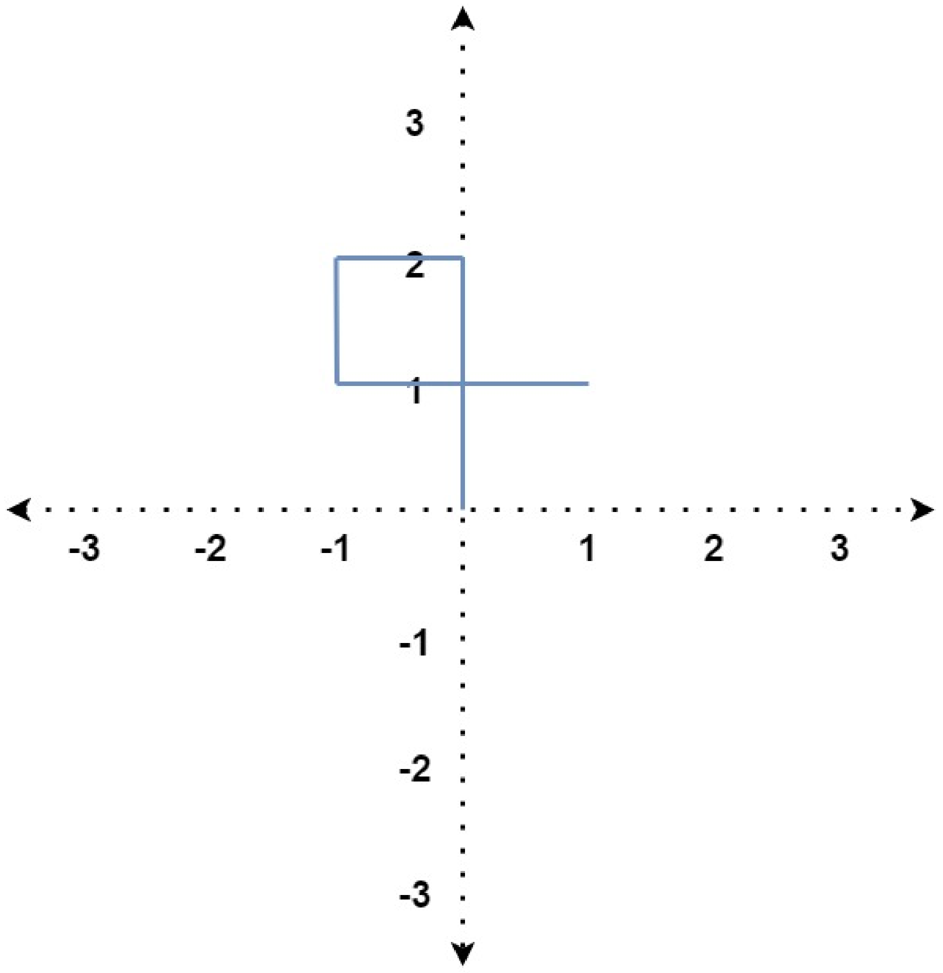
if your path crosses itself or

false

if it does not

.

Example 1:



Input:

distance = [2,1,1,2]

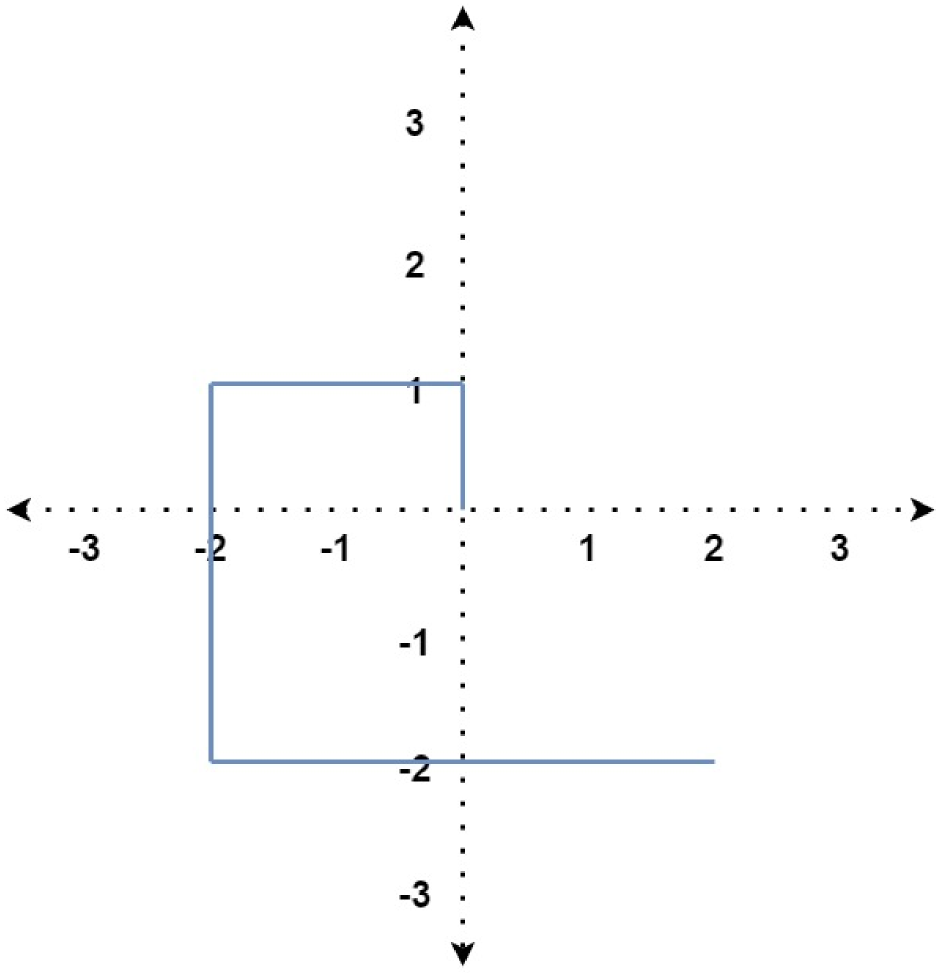
Output:

true

Explanation:

The path crosses itself at the point (0, 1).

Example 2:



Input:

distance = [1,2,3,4]

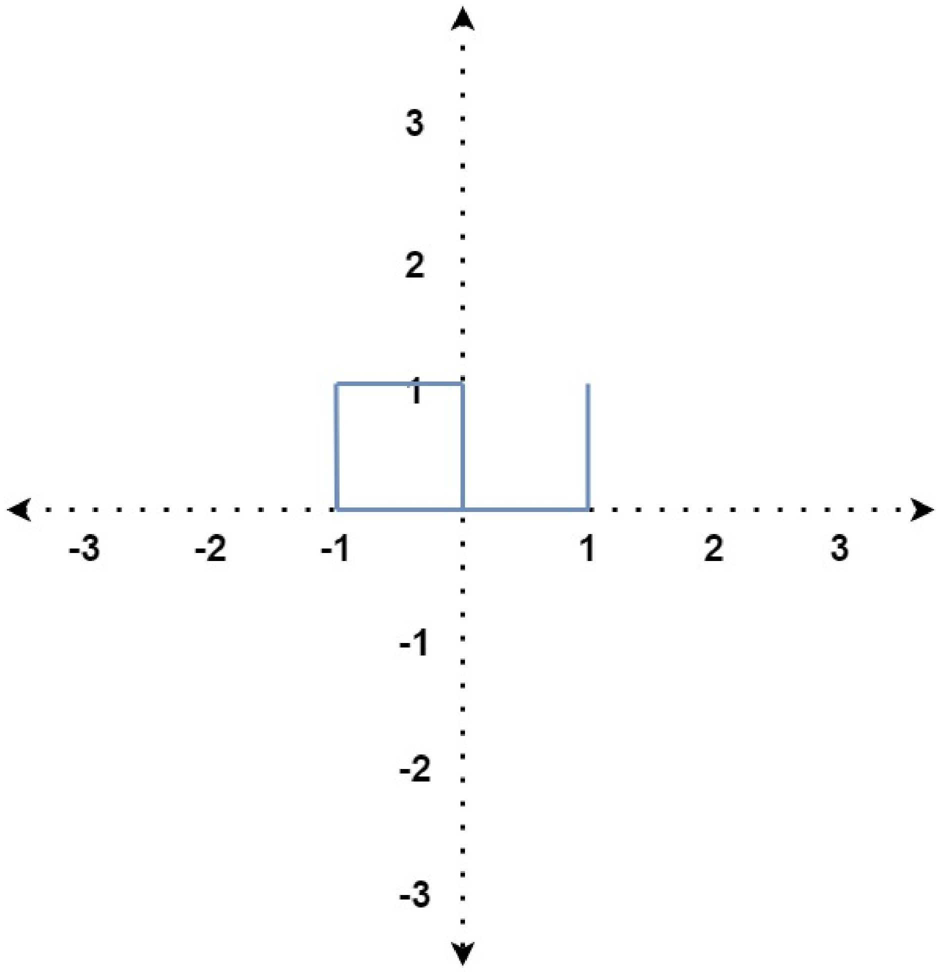
Output:

false

Explanation:

The path does not cross itself at any point.

Example 3:



Input:

distance = [1,1,1,2,1]

Output:

true

Explanation:

The path crosses itself at the point (0, 0).

Constraints:

1 <= distance.length <= 10

1 <= distance[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
bool isSelfCrossing(vector<int>& distance) {

}
};
```

Java:

```
class Solution {
public boolean isSelfCrossing(int[] distance) {

}
}
```

Python3:

```
class Solution:
def isSelfCrossing(self, distance: List[int]) -> bool:
```

Python:

```
class Solution(object):
def isSelfCrossing(self, distance):
"""
:type distance: List[int]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number[]} distance
* @return {boolean}
*/
var isSelfCrossing = function(distance) {

};
```

TypeScript:

```
function isSelfCrossing(distance: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool IsSelfCrossing(int[] distance) {

}
}
```

C:

```
bool isSelfCrossing(int* distance, int distanceSize) {

}
```

Go:

```
func isSelfCrossing(distance []int) bool {

}
```

Kotlin:

```
class Solution {
fun isSelfCrossing(distance: IntArray): Boolean {

}
}
```

Swift:

```
class Solution {
func isSelfCrossing(_ distance: [Int]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn is_self_crossing(distance: Vec<i32>) -> bool {

}
}
```

Ruby:

```
# @param {Integer[]} distance
# @return {Boolean}
def is_self_crossing(distance)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $distance
* @return Boolean
*/
function isSelfCrossing($distance) {

}
}
```

Dart:

```
class Solution {
bool isSelfCrossing(List<int> distance) {

}
}
```

Scala:

```
object Solution {
def isSelfCrossing(distance: Array[Int]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec is_self_crossing(distance :: [integer]) :: boolean
def is_self_crossing(distance) do

end
end
```

Erlang:

```
-spec is_self_crossing(Distance :: [integer()]) -> boolean().
is_self_crossing(Distance) ->
.
```

Racket:

```
(define/contract (is-self-crossing distance)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Self Crossing
* Difficulty: Hard
* Tags: array, math
*
* 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 Solution {
public:
bool isSelfCrossing(vector<int>& distance) {

}
};
```

Java Solution:

```
/**
* Problem: Self Crossing
* Difficulty: Hard
* Tags: array, math
*
* 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 Solution {
public boolean isSelfCrossing(int[] distance) {

}
}
```

Python3 Solution:

```
"""
Problem: Self Crossing
Difficulty: Hard
Tags: array, math

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 Solution:
def isSelfCrossing(self, distance: List[int]) -> bool:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def isSelfCrossing(self, distance):
"""
:type distance: List[int]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Self Crossing
* Difficulty: Hard
* Tags: array, math
*
* 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[]} distance
* @return {boolean}
*/
var isSelfCrossing = function(distance) {

};
```

TypeScript Solution:

```
/**
* Problem: Self Crossing
* Difficulty: Hard
* Tags: array, math
*
* 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
```

```
*/

function isSelfCrossing(distance: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Self Crossing
* Difficulty: Hard
* Tags: array, math
*
* 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 Solution {
public bool IsSelfCrossing(int[] distance) {

}
}
```

C Solution:

```
/*
* Problem: Self Crossing
* Difficulty: Hard
* Tags: array, math
*
* 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
*/

bool isSelfCrossing(int* distance, int distanceSize) {

}
```

Go Solution:

```
// Problem: Self Crossing
// Difficulty: Hard
// Tags: array, math
//
// 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

func isSelfCrossing(distance []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun isSelfCrossing(distance: IntArray): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func isSelfCrossing(_ distance: [Int]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Self Crossing
// Difficulty: Hard
// Tags: array, math
//
// 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

impl Solution {
pub fn is_self_crossing(distance: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[]} distance
# @return {Boolean}
def is_self_crossing(distance)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $distance
* @return Boolean
*/
function isSelfCrossing($distance) {

}
}
```

Dart Solution:

```
class Solution {
bool isSelfCrossing(List<int> distance) {

}
}
```

Scala Solution:

```
object Solution {
def isSelfCrossing(distance: Array[Int]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec is_self_crossing(distance :: [integer]) :: boolean
def is_self_crossing(distance) do

end
end
```

Erlang Solution:

```
-spec is_self_crossing(Distance :: [integer()]) -> boolean().
is_self_crossing(Distance) ->
.
```

Racket Solution:

```
(define/contract (is-self-crossing distance)
(-> (listof exact-integer?) boolean?)
)
```