Problem 473: Matchsticks to Square

Problem Information

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

Problem Description

You are given an integer array

matchsticks

where

matchsticks[i]

is the length of the

i

th

matchstick. You want to use

all the matchsticks

to make one square. You

should not break

any stick, but you can link them up, and each matchstick must be used

exactly one time

.

Return

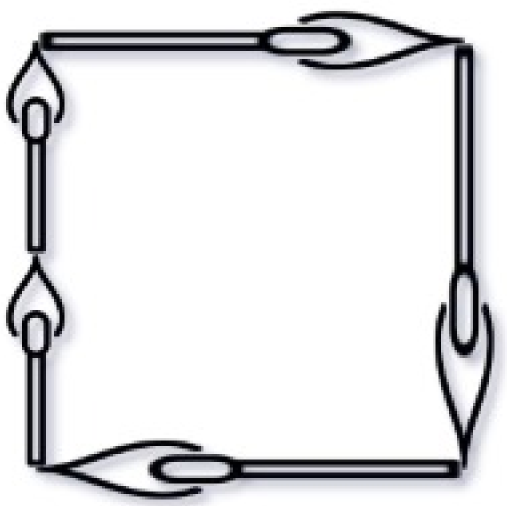
true

if you can make this square and

false

otherwise.

Example 1:



Input:

matchsticks = [1,1,2,2,2]

Output:

true

Explanation:

You can form a square with length 2, one side of the square came two sticks with length 1.

Example 2:

Input:

matchsticks = [3,3,3,3,4]

Output:

false

Explanation:

You cannot find a way to form a square with all the matchsticks.

Constraints:

1 <= matchsticks.length <= 15

1 <= matchsticks[i] <= 10

8

Code Snippets

C++:

```
class Solution {
public:
bool makesquare(vector<int>& matchsticks) {

}
};
```

Java:

```
class Solution {
public boolean makesquare(int[] matchsticks) {
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function makesquare(matchsticks: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool Makesquare(int[] matchsticks) {

}
}
```

C:

```
bool makesquare(int* matchsticks, int matchsticksSize) {

}
```

Go:

```
func makesquare(matchsticks []int) bool {

}
```

Kotlin:

```
class Solution {
fun makesquare(matchsticks: IntArray): Boolean {

}
}
```

Swift:

```
class Solution {
func makesquare(_ matchsticks: [Int]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn makesquare(matchsticks: Vec<i32>) -> bool {

}
}
```

Ruby:

```
# @param {Integer[]} matchsticks
# @return {Boolean}
def makesquare(matchsticks)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $matchsticks
* @return Boolean
*/
function makesquare($matchsticks) {

}
}
```

Dart:

```
class Solution {
bool makesquare(List<int> matchsticks) {

}
}
```

Scala:

```
object Solution {
def makesquare(matchsticks: Array[Int]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec makesquare(matchsticks :: [integer]) :: boolean
def makesquare(matchsticks) do

end
end
```

Erlang:

```
-spec makesquare(Matchsticks :: [integer()]) -> boolean().
makesquare(Matchsticks) ->
.
```

Racket:

```
(define/contract (makesquare matchsticks)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Matchsticks to Square
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
bool makesquare(vector<int>& matchsticks) {

}
};
```

Java Solution:

```
/**
* Problem: Matchsticks to Square
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public boolean makesquare(int[] matchsticks) {
}
}
```

Python3 Solution:

```
"""
Problem: Matchsticks to Square
Difficulty: Medium
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def makesquare(self, matchsticks: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Matchsticks to Square
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Matchsticks to Square
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function makesquare(matchsticks: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Matchsticks to Square
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public bool Makesquare(int[] matchsticks) {

}

}
```

C Solution:

```
/*
* Problem: Matchsticks to Square
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

bool makesquare(int* matchsticks, int matchsticksSize) {

}
```

Go Solution:

```
// Problem: Matchsticks to Square
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func makesquare(matchsticks []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun makesquare(matchsticks: IntArray): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func makesquare(_ matchsticks: [Int]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Matchsticks to Square
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn makesquare(matchsticks: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[]} matchsticks
# @return {Boolean}
def makesquare(matchsticks)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $matchsticks
* @return Boolean
*/
function makesquare($matchsticks) {

}
}
```

Dart Solution:

```
class Solution {
bool makesquare(List<int> matchsticks) {

}
}
```

Scala Solution:

```
object Solution {
def makesquare(matchsticks: Array[Int]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec makesquare(matchsticks :: [integer]) :: boolean
def makesquare(matchsticks) do

end
end
```

Erlang Solution:

```
-spec makesquare(Matchsticks :: [integer()]) -> boolean().
makesquare(Matchsticks) ->
.
```

Racket Solution:

```
(define/contract (makesquare matchsticks)
(-> (listof exact-integer?) boolean?)
)
```