# Problem 679: 24 Game

## Problem Information

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

## Problem Description

You are given an integer array

cards

of length

4

. You have four cards, each containing a number in the range

[1, 9]

. You should arrange the numbers on these cards in a mathematical expression using the operators

['+', '-', '*', '/']

and the parentheses

'('

and

')'

to get the value 24.

You are restricted with the following rules:

The division operator

'/'

represents real division, not integer division.

For example,

4 / (1 - 2 / 3) = 4 / (1 / 3) = 12

.

Every operation done is between two numbers. In particular, we cannot use

'-'

as a unary operator.

For example, if

cards = [1, 1, 1, 1]

, the expression

"-1 - 1 - 1 - 1"

is

not allowed

.

You cannot concatenate numbers together

For example, if

cards = [1, 2, 1, 2]

, the expression

"12 + 12"

is not valid.

Return

true

if you can get such expression that evaluates to

24

, and

false

otherwise.

Example 1:

Input:

cards = [4,1,8,7]

Output:

true

Explanation:

(8-4) * (7-1) = 24

Example 2:

Input:

cards = [1,2,1,2]

Output:

false

Constraints:

cards.length == 4

1 <= cards[i] <= 9

## Code Snippets

### C++:

```
class Solution {
public:
bool judgePoint24(vector<int>& cards) {

}
};
```

### Java:

```
class Solution {
public boolean judgePoint24(int[] cards) {

}
}
```

### Python3:

```
class Solution:
def judgePoint24(self, cards: List[int]) -> bool:
```

### Python:

```
class Solution(object):
def judgePoint24(self, cards):
```

```
"""
:type cards: List[int]
:rtype: bool
"""
```

**JavaScript:**

```
/**
* @param {number[]} cards
* @return {boolean}
*/
var judgePoint24 = function(cards) {

};
```

**TypeScript:**

```
function judgePoint24(cards: number[]): boolean {

};
```

**C#:**

```
public class Solution {
public bool JudgePoint24(int[] cards) {

}
}
```

**C:**

```
bool judgePoint24(int* cards, int cardsSize) {

}
```

**Go:**

```
func judgePoint24(cards []int) bool {

}
```

**Kotlin:**

```
class Solution {
fun judgePoint24(cards: IntArray): Boolean {

}
}
```

**Swift:**

```
class Solution {
func judgePoint24(_ cards: [Int]) -> Bool {

}
}
```

**Rust:**

```
impl Solution {
pub fn judge_point24(cards: Vec<i32>) -> bool {

}
}
```

**Ruby:**

```
# @param {Integer[]} cards
# @return {Boolean}
def judge_point24(cards)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $cards
* @return Boolean
*/
function judgePoint24($cards) {

}
}
```

**Dart:**

```
class Solution {
bool judgePoint24(List<int> cards) {

}
}
```

**Scala:**

```
object Solution {
def judgePoint24(cards: Array[Int]): Boolean = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec judge_point24(cards :: [integer]) :: boolean
def judge_point24(cards) do

end
end
```

**Erlang:**

```
-spec judge_point24(Cards :: [integer()]) -> boolean().
judge_point24(Cards) ->
.
```

**Racket:**

```
(define/contract (judge-point24 cards)
(-> (listof exact-integer?) boolean?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: 24 Game
* 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 judgePoint24(vector<int>& cards) {

}
};
```

### Java Solution:

```
/**
* Problem: 24 Game
* 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 judgePoint24(int[] cards) {

}
}
```

### Python3 Solution:

```
"""
Problem: 24 Game
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 judgePoint24(self, cards: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def judgePoint24(self, cards):
"""
:type cards: List[int]
:rtype: bool
"""
```

### JavaScript Solution:

```
/**
* Problem: 24 Game
* 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[]} cards
* @return {boolean}
*/
var judgePoint24 = function(cards) {

};
```

### TypeScript Solution:

```
/**
* Problem: 24 Game
* 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 judgePoint24(cards: number[]): boolean {

};
```

### C# Solution:

```
/*
* Problem: 24 Game
* 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 JudgePoint24(int[] cards) {

}
}
```

### C Solution:

```
/*
* Problem: 24 Game
* 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 judgePoint24(int* cards, int cardsSize) {

}
```

**Go Solution:**

```
// Problem: 24 Game
// 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 judgePoint24(cards []int) bool {

}
```

**Kotlin Solution:**

```
class Solution {
fun judgePoint24(cards: IntArray): Boolean {

}
}
```

**Swift Solution:**

```
class Solution {
func judgePoint24(_ cards: [Int]) -> Bool {

}
}
```

**Rust Solution:**

```
// Problem: 24 Game
// 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 judge_point24(cards: Vec<i32>) -> bool {

}
}
```

### Ruby Solution:

```
# @param {Integer[]} cards
# @return {Boolean}
def judge_point24(cards)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[] $cards
* @return Boolean
*/
function judgePoint24($cards) {

}
}
```

### Dart Solution:

```
class Solution {
bool judgePoint24(List<int> cards) {

}
}
```

### Scala Solution:

```
object Solution {
def judgePoint24(cards: Array[Int]): Boolean = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec judge_point24(cards :: [integer]) :: boolean
def judge_point24(cards) do

end
end
```

### Erlang Solution:

```
-spec judge_point24(Cards :: [integer()]) -> boolean().
judge_point24(Cards) ->
.
```

### Racket Solution:

```
(define/contract (judge-point24 cards)
(-> (listof exact-integer?) boolean?)
)
```