# Problem 914: X of a Kind in a Deck of Cards

## Problem Information

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

## Problem Description

You are given an integer array

deck

where

deck[i]

represents the number written on the

i

th

card.

Partition the cards into

one or more groups

such that:

Each group has

exactly

x

cards where

x > 1

, and

All the cards in one group have the same integer written on them.

Return

true

if such partition is possible, or

false

otherwise

.

Example 1:

Input:

deck = [1,2,3,4,4,3,2,1]

Output:

true

Explanation

: Possible partition [1,1],[2,2],[3,3],[4,4].

Example 2:

Input:

deck = [1,1,1,2,2,2,3,3]

Output:

false

Explanation

: No possible partition.

Constraints:

1 <= deck.length <= 10

4

0 <= deck[i] < 10

4

## Code Snippets

**C++:**

```
class Solution {
public:
bool hasGroupsSizeX(vector<int>& deck) {

}
};
```

**Java:**

```
class Solution {
public boolean hasGroupsSizeX(int[] deck) {

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

```
function hasGroupsSizeX(deck: number[]): boolean {

};
```

**C#:**

```
public class Solution {
public bool HasGroupsSizeX(int[] deck) {

}
}
```

**C:**

```
bool hasGroupsSizeX(int* deck, int deckSize) {

}
```

**Go:**

```
func hasGroupsSizeX(deck []int) bool {

}
```

**Kotlin:**

```
class Solution {
fun hasGroupsSizeX(deck: IntArray): Boolean {

}
}
```

**Swift:**

```
class Solution {
func hasGroupsSizeX(_ deck: [Int]) -> Bool {

}
}
```

**Rust:**

```
impl Solution {
pub fn has_groups_size_x(deck: Vec<i32>) -> bool {

}
}
```

**Ruby:**

```
# @param {Integer[]} deck
# @return {Boolean}
def has_groups_size_x(deck)

end
```

**PHP:**

```
class Solution {

/**
```

```
* @param Integer[] $deck
* @return Boolean
*/
function hasGroupsSizeX($deck) {

}
}
```

**Dart:**

```
class Solution {
bool hasGroupsSizeX(List<int> deck) {

}
}
```

**Scala:**

```
object Solution {
def hasGroupsSizeX(deck: Array[Int]): Boolean = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec has_groups_size_x(deck :: [integer]) :: boolean
def has_groups_size_x(deck) do

end
end
```

**Erlang:**

```
-spec has_groups_size_x(Deck :: [integer()]) -> boolean().
has_groups_size_x(Deck) ->
.
```

**Racket:**

```
(define/contract (has-groups-size-x deck)
(-> (listof exact-integer?) boolean?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: X of a Kind in a Deck of Cards
* Difficulty: Easy
* Tags: array, math, 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 Solution {
public:
bool hasGroupsSizeX(vector<int>& deck) {

}
};
```

### Java Solution:

```
/**
* Problem: X of a Kind in a Deck of Cards
* Difficulty: Easy
* Tags: array, math, 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 Solution {
public boolean hasGroupsSizeX(int[] deck) {

}
```

```
}
```

### Python3 Solution:

```
"""
Problem: X of a Kind in a Deck of Cards
Difficulty: Easy
Tags: array, math, 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 Solution:
def hasGroupsSizeX(self, deck: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: X of a Kind in a Deck of Cards
* Difficulty: Easy
* Tags: array, math, 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
*/

/**
```

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

};
```

### TypeScript Solution:

```
/**
* Problem: X of a Kind in a Deck of Cards
* Difficulty: Easy
* Tags: array, math, 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
*/

function hasGroupsSizeX(deck: number[]): boolean {

};
```

### C# Solution:

```
/*
* Problem: X of a Kind in a Deck of Cards
* Difficulty: Easy
* Tags: array, math, 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 Solution {
public bool HasGroupsSizeX(int[] deck) {

}
}
```

### C Solution:

```
/*
* Problem: X of a Kind in a Deck of Cards
* Difficulty: Easy
* Tags: array, math, 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
*/

bool hasGroupsSizeX(int* deck, int deckSize) {

}
```

### Go Solution:

```
// Problem: X of a Kind in a Deck of Cards
// Difficulty: Easy
// Tags: array, math, 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

func hasGroupsSizeX(deck []int) bool {

}
```

### Kotlin Solution:

```
class Solution {
fun hasGroupsSizeX(deck: IntArray): Boolean {

}
}
```

### Swift Solution:

```
class Solution {
func hasGroupsSizeX(_ deck: [Int]) -> Bool {
```

```
}
}
```

### Rust Solution:

```
// Problem: X of a Kind in a Deck of Cards
// Difficulty: Easy
// Tags: array, math, 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

impl Solution {
pub fn has_groups_size_x(deck: Vec<i32>) -> bool {

}
}
```

### Ruby Solution:

```
# @param {Integer[]} deck
# @return {Boolean}
def has_groups_size_x(deck)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[] $deck
* @return Boolean
*/
function hasGroupsSizeX($deck) {

}
}
```

### Dart Solution:

```
class Solution {
bool hasGroupsSizeX(List<int> deck) {

}
}
```

### Scala Solution:

```
object Solution {
def hasGroupsSizeX(deck: Array[Int]): Boolean = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec has_groups_size_x(deck :: [integer]) :: boolean
def has_groups_size_x(deck) do

end
end
```

### Erlang Solution:

```
-spec has_groups_size_x(Deck :: [integer()]) -> boolean().
has_groups_size_x(Deck) ->
.
```

### Racket Solution:

```
(define/contract (has-groups-size-x deck)
(-> (listof exact-integer?) boolean?)
)
```