

# Problem 3206: Alternating Groups I

## Problem Information

**Difficulty:** [Easy](#)

**Acceptance Rate:** 0.00%

**Paid Only:** No

## Problem Description

There is a circle of red and blue tiles. You are given an array of integers

colors

. The color of tile

i

is represented by

colors[i]

:

colors[i] == 0

means that tile

i

is

red

.

`colors[i] == 1`

means that tile

i

is

blue

Every 3 contiguous tiles in the circle with

alternating

colors (the middle tile has a different color from its

left

and

right

tiles) is called an

alternating

group.

Return the number of

alternating

groups.

Note

that since

colors

represents a

circle

, the

first

and the

last

tiles are considered to be next to each other.

Example 1:

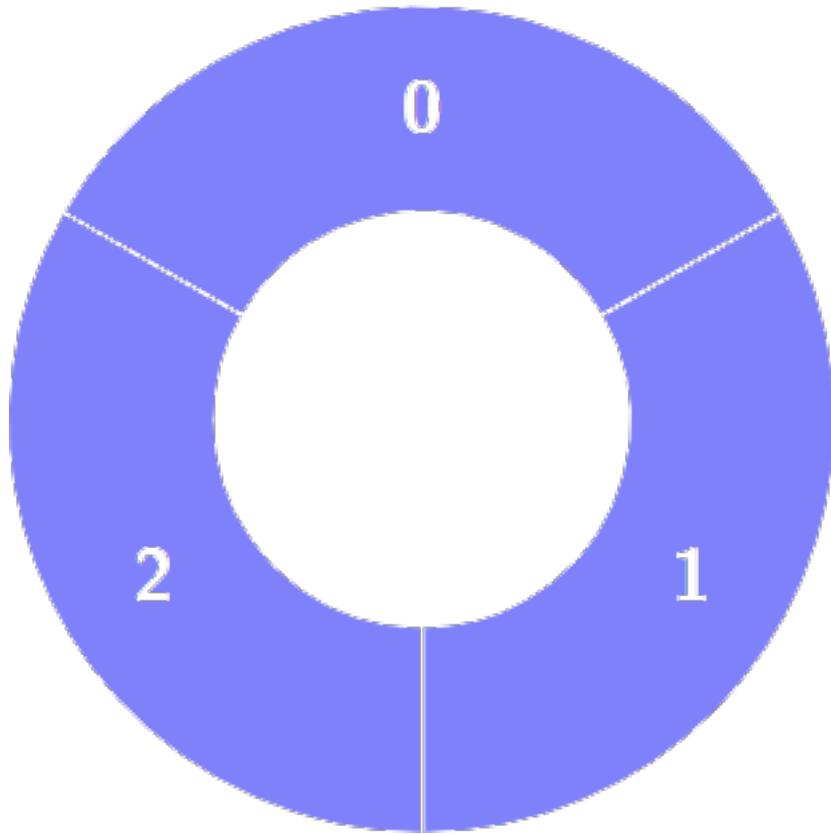
Input:

colors = [1,1,1]

Output:

0

Explanation:



Example 2:

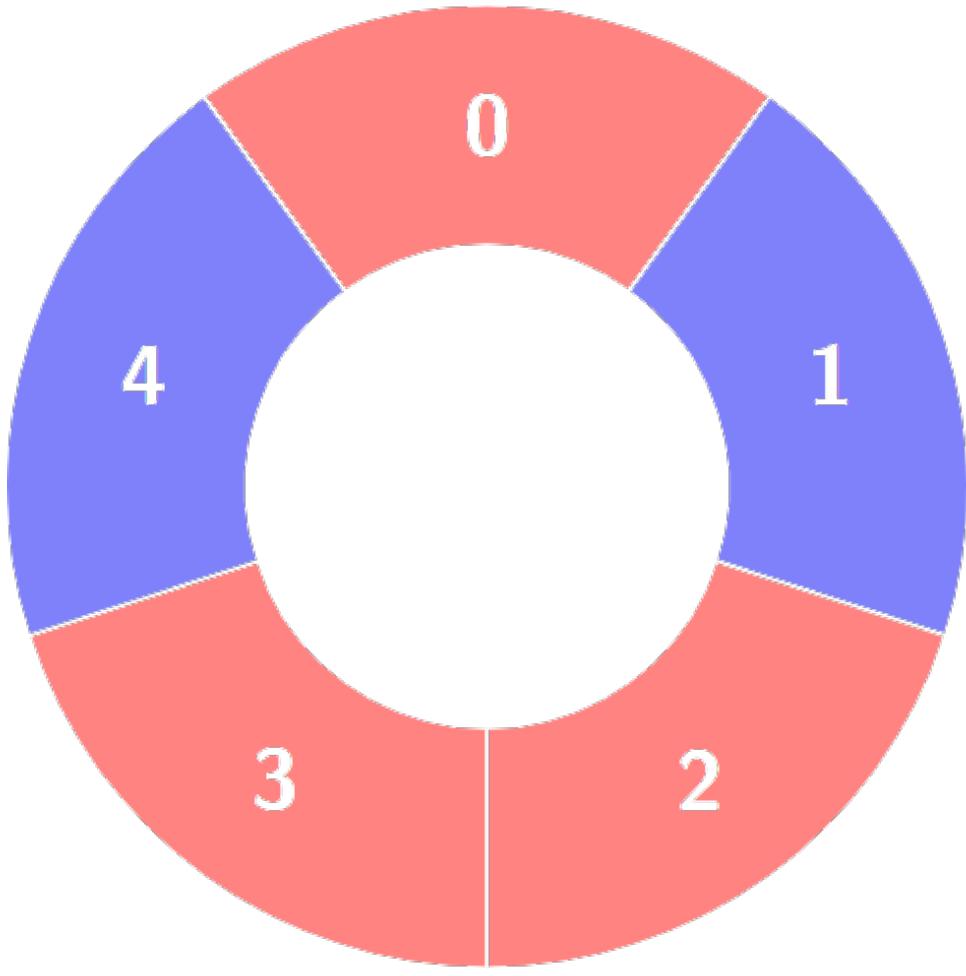
Input:

colors = [0,1,0,0,1]

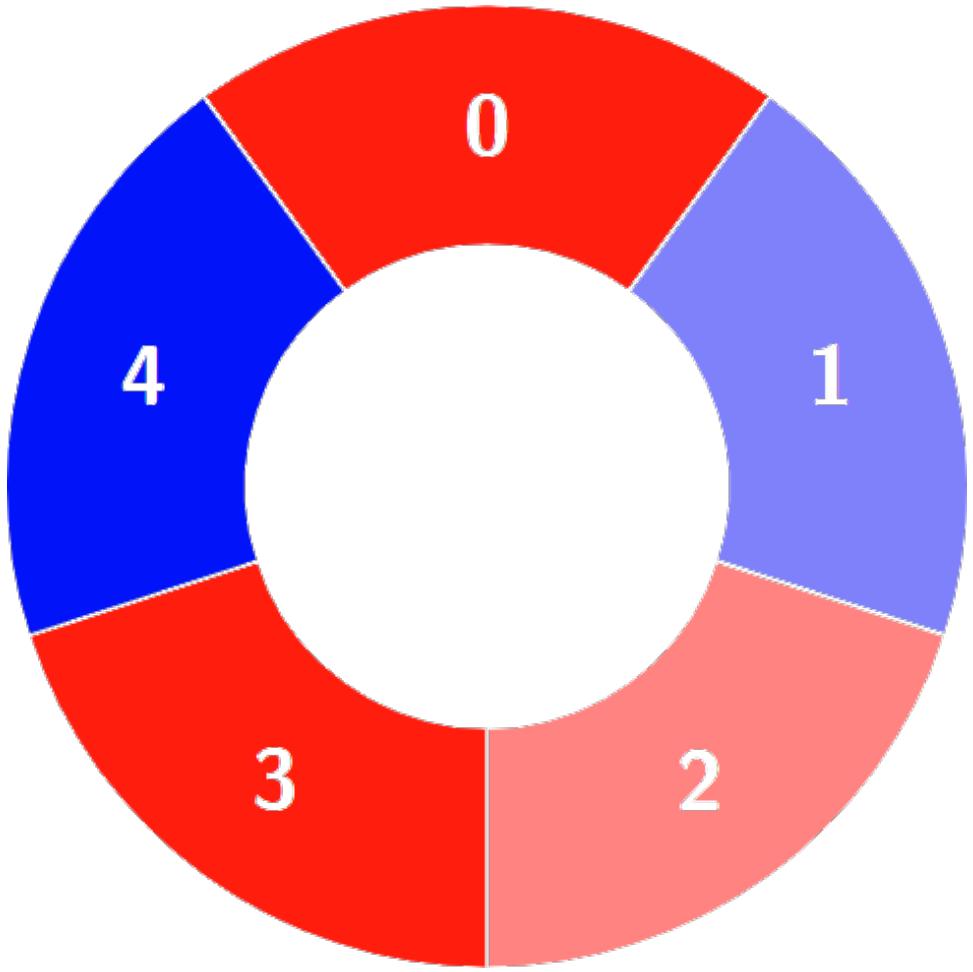
Output:

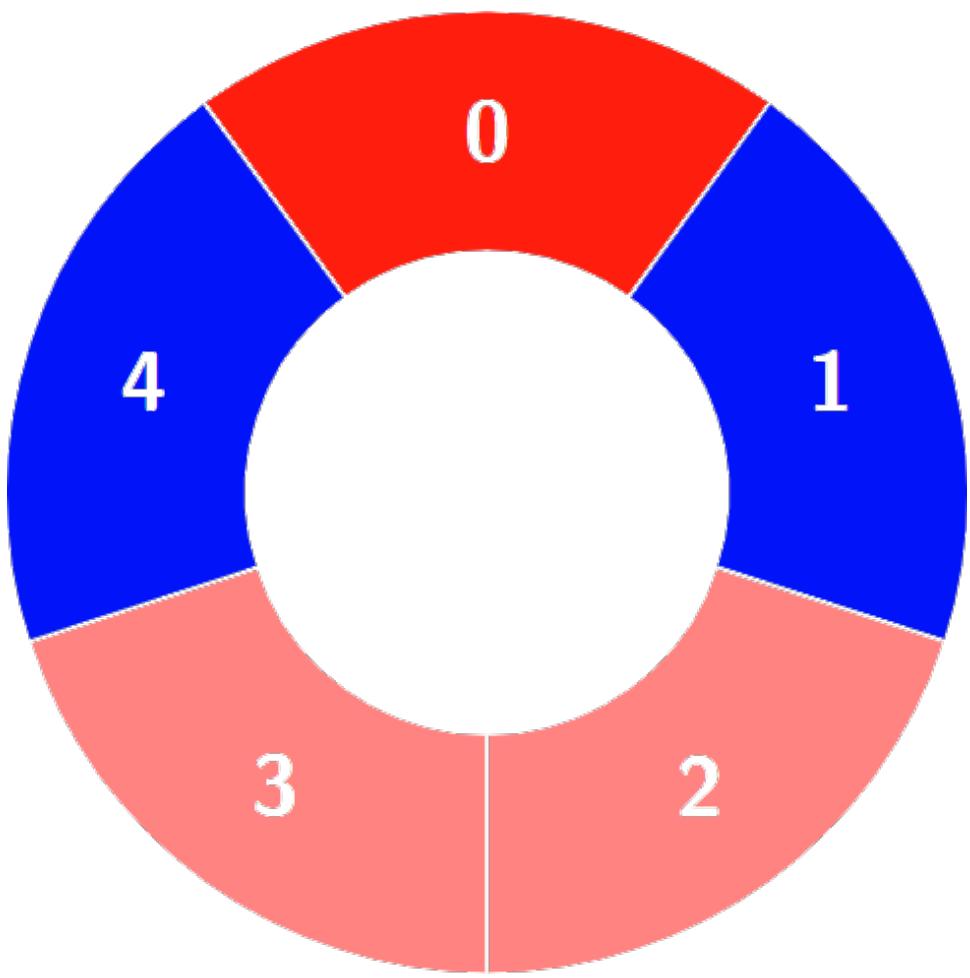
3

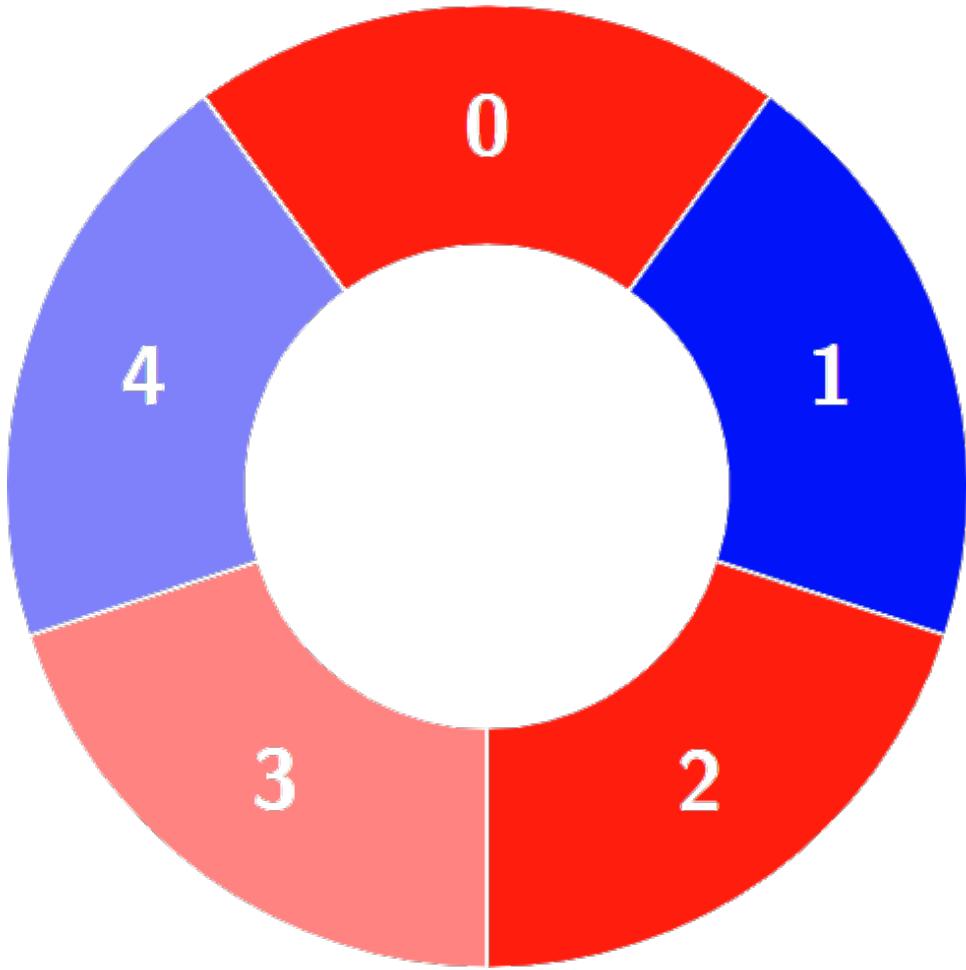
Explanation:



Alternating groups:







Constraints:

$3 \leq \text{colors.length} \leq 100$

$0 \leq \text{colors}[i] \leq 1$

## Code Snippets

C++:

```
class Solution {
public:
    int numberOfAlternatingGroups(vector<int>& colors) {
    }
};
```

**Java:**

```
class Solution {  
    public int numberOfAlternatingGroups(int[] colors) {  
  
    }  
}
```

**Python3:**

```
class Solution:  
    def numberOfAlternatingGroups(self, colors: List[int]) -> int:
```

**Python:**

```
class Solution(object):  
    def numberOfAlternatingGroups(self, colors):  
        """  
        :type colors: List[int]  
        :rtype: int  
        """
```

**JavaScript:**

```
/**  
 * @param {number[]} colors  
 * @return {number}  
 */  
var numberOfAlternatingGroups = function(colors) {  
  
};
```

**TypeScript:**

```
function numberOfAlternatingGroups(colors: number[]): number {  
  
};
```

**C#:**

```
public class Solution {  
    public int NumberOfAlternatingGroups(int[] colors) {
```

```
}
```

```
}
```

**C:**

```
int numberOfAlternatingGroups(int* colors, int colorsSize) {  
  
}
```

**Go:**

```
func numberOfAlternatingGroups(colors []int) int {  
  
}
```

**Kotlin:**

```
class Solution {  
    fun numberOfAlternatingGroups(colors: IntArray): Int {  
  
    }  
}
```

**Swift:**

```
class Solution {  
    func numberOfAlternatingGroups(_ colors: [Int]) -> Int {  
  
    }  
}
```

**Rust:**

```
impl Solution {  
    pub fn number_of_alternating_groups(colors: Vec<i32>) -> i32 {  
  
    }  
}
```

**Ruby:**

```
# @param {Integer[]} colors
# @return {Integer}
def number_of_alternating_groups(colors)

end
```

### PHP:

```
class Solution {

    /**
     * @param Integer[] $colors
     * @return Integer
     */
    function numberOfAlternatingGroups($colors) {

    }
}
```

### Dart:

```
class Solution {
int numberOfAlternatingGroups(List<int> colors) {

}
```

### Scala:

```
object Solution {
def numberOfAlternatingGroups(colors: Array[Int]): Int = {

}
```

### Elixir:

```
defmodule Solution do
@spec number_of_alternating_groups(colors :: [integer]) :: integer
def number_of_alternating_groups(colors) do

end
end
```

### Erlang:

```
-spec number_of_alternating_groups(Colors :: [integer()]) -> integer().  
number_of_alternating_groups(Colors) ->  
.
```

### Racket:

```
(define/contract (number-of-alternating-groups colors)  
  (-> (listof exact-integer?) exact-integer?)  
)
```

## Solutions

### C++ Solution:

```
/*  
 * Problem: Alternating Groups I  
 * Difficulty: Easy  
 * Tags: array  
 *  
 * 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:  
    int numberOfAlternatingGroups(vector<int>& colors) {  
  
    }  
};
```

### Java Solution:

```
/**  
 * Problem: Alternating Groups I  
 * Difficulty: Easy  
 * Tags: array  
 *  
 * 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 int numberOfAlternatingGroups(int[] colors) {
        return 0;
    }
}

```

### Python3 Solution:

```

"""
Problem: Alternating Groups I
Difficulty: Easy
Tags: array

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 numberOfAlternatingGroups(self, colors: List[int]) -> int:
        # TODO: Implement optimized solution
        pass

```

### Python Solution:

```

class Solution(object):
    def numberOfAlternatingGroups(self, colors):
        """
        :type colors: List[int]
        :rtype: int
        """

```

### JavaScript Solution:

```

/**
 * Problem: Alternating Groups I
 * Difficulty: Easy
 */

```

```

* Tags: array
*
* 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[]} colors
* @return {number}
*/
var numberOfAlternatingGroups = function(colors) {
};

```

### TypeScript Solution:

```

/**
* Problem: Alternating Groups I
* Difficulty: Easy
* Tags: array
*
* 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 numberOfAlternatingGroups(colors: number[]): number {
}

```

### C# Solution:

```

/*
* Problem: Alternating Groups I
* Difficulty: Easy
* Tags: array
*
* 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

```

```
*/\n\npublic class Solution {\n    public int NumberOfAlternatingGroups(int[] colors) {\n        }\n    }\n}
```

### C Solution:

```
/*\n * Problem: Alternating Groups I\n * Difficulty: Easy\n * Tags: array\n *\n * Approach: Use two pointers or sliding window technique\n * Time Complexity: O(n) or O(n log n)\n * Space Complexity: O(1) to O(n) depending on approach\n */\n\nint numberOfAlternatingGroups(int* colors, int colorsSize) {\n}\n
```

### Go Solution:

```
// Problem: Alternating Groups I\n// Difficulty: Easy\n// Tags: array\n//\n// Approach: Use two pointers or sliding window technique\n// Time Complexity: O(n) or O(n log n)\n// Space Complexity: O(1) to O(n) depending on approach\n\nfunc numberOfAlternatingGroups(colors []int) int {\n}
```

### Kotlin Solution:

```
class Solution {  
    fun numberOfAlternatingGroups(colors: IntArray): Int {  
        }  
        }  
}
```

### Swift Solution:

```
class Solution {  
    func numberOfAlternatingGroups(_ colors: [Int]) -> Int {  
        }  
        }  
}
```

### Rust Solution:

```
// Problem: Alternating Groups I  
// Difficulty: Easy  
// Tags: array  
//  
// 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 number_of_alternating_groups(colors: Vec<i32>) -> i32 {  
        }  
        }  
}
```

### Ruby Solution:

```
# @param {Integer[]} colors  
# @return {Integer}  
def number_of_alternating_groups(colors)  
  
end
```

### PHP Solution:

```
class Solution {
```

```
/**
 * @param Integer[] $colors
 * @return Integer
 */
function numberOfAlternatingGroups($colors) {

}

}
```

### Dart Solution:

```
class Solution {
int numberOfAlternatingGroups(List<int> colors) {

}
```

### Scala Solution:

```
object Solution {
def numberOfAlternatingGroups(colors: Array[Int]): Int = {

}
```

### Elixir Solution:

```
defmodule Solution do
@spec number_of_alternating_groups(colors :: [integer]) :: integer
def number_of_alternating_groups(colors) do

end
end
```

### Erlang Solution:

```
-spec number_of_alternating_groups(Colors :: [integer()]) -> integer().
number_of_alternating_groups(Colors) ->
.
```

### Racket Solution:

```
(define/contract (number-of-alternating-groups colors)
  (-> (listof exact-integer?) exact-integer?))
)
```