Problem 3245: Alternating Groups III

Problem Information

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

Problem Description

There are some red and blue tiles arranged circularly. You are given an array of integers

colors

and a 2D integers array

queries

.

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

.

An

alternating

group is a contiguous subset of tiles in the circle with

alternating

colors (each tile in the group except the first and last one has a different color from its

adjacent

tiles in the group).

You have to process queries of two types:

queries[i] = [1, size

i

]

, determine the count of

alternating

groups with size

size

i

.

queries[i] = [2, index

i

, color

i

]

, change

colors[index

i

]

to

color

i

.

Return an array

answer

containing the results of the queries of the first type

in order

.

Note

that since

colors

represents a

circle

, the

first

and the

last

tiles are considered to be next to each other.

Example 1:

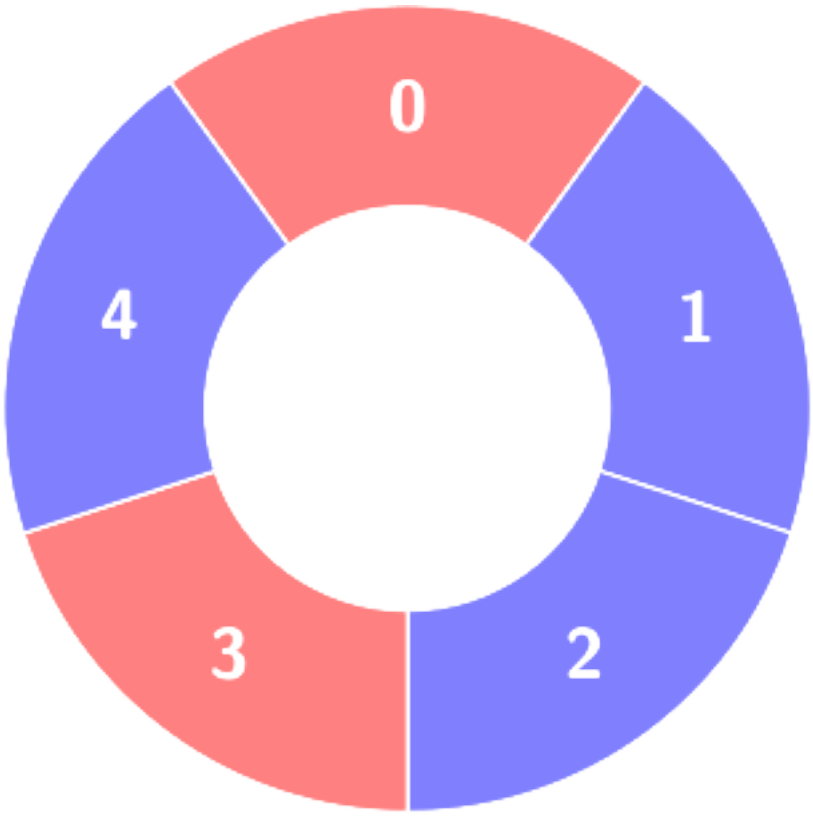
Input:

colors = [0,1,1,0,1], queries = [[2,1,0],[1,4]]

Output:

[2]

Explanation:

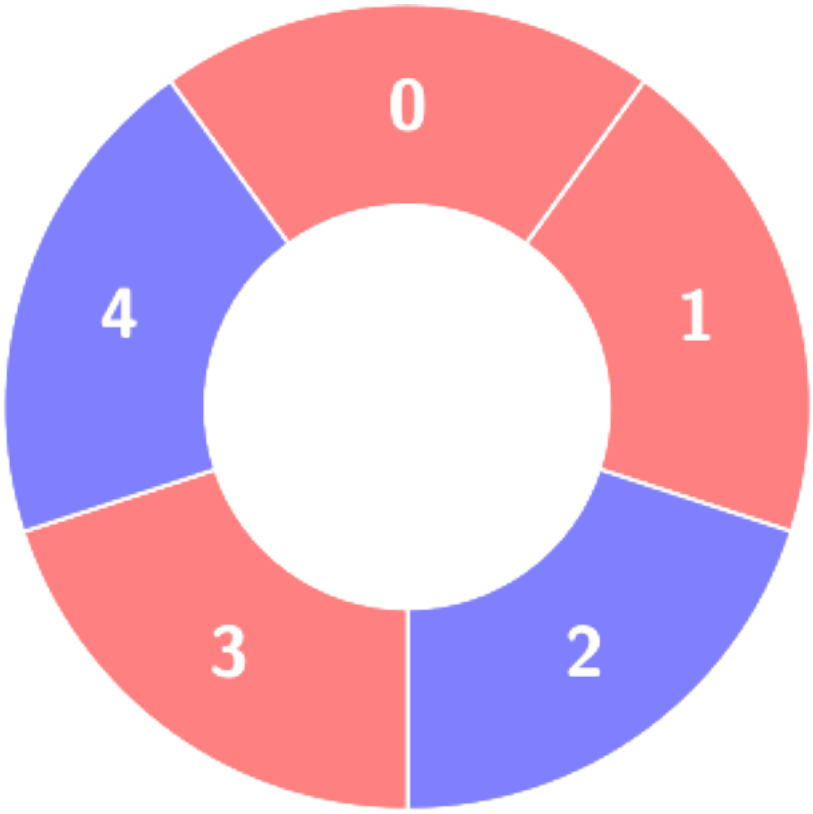


First query:

Change

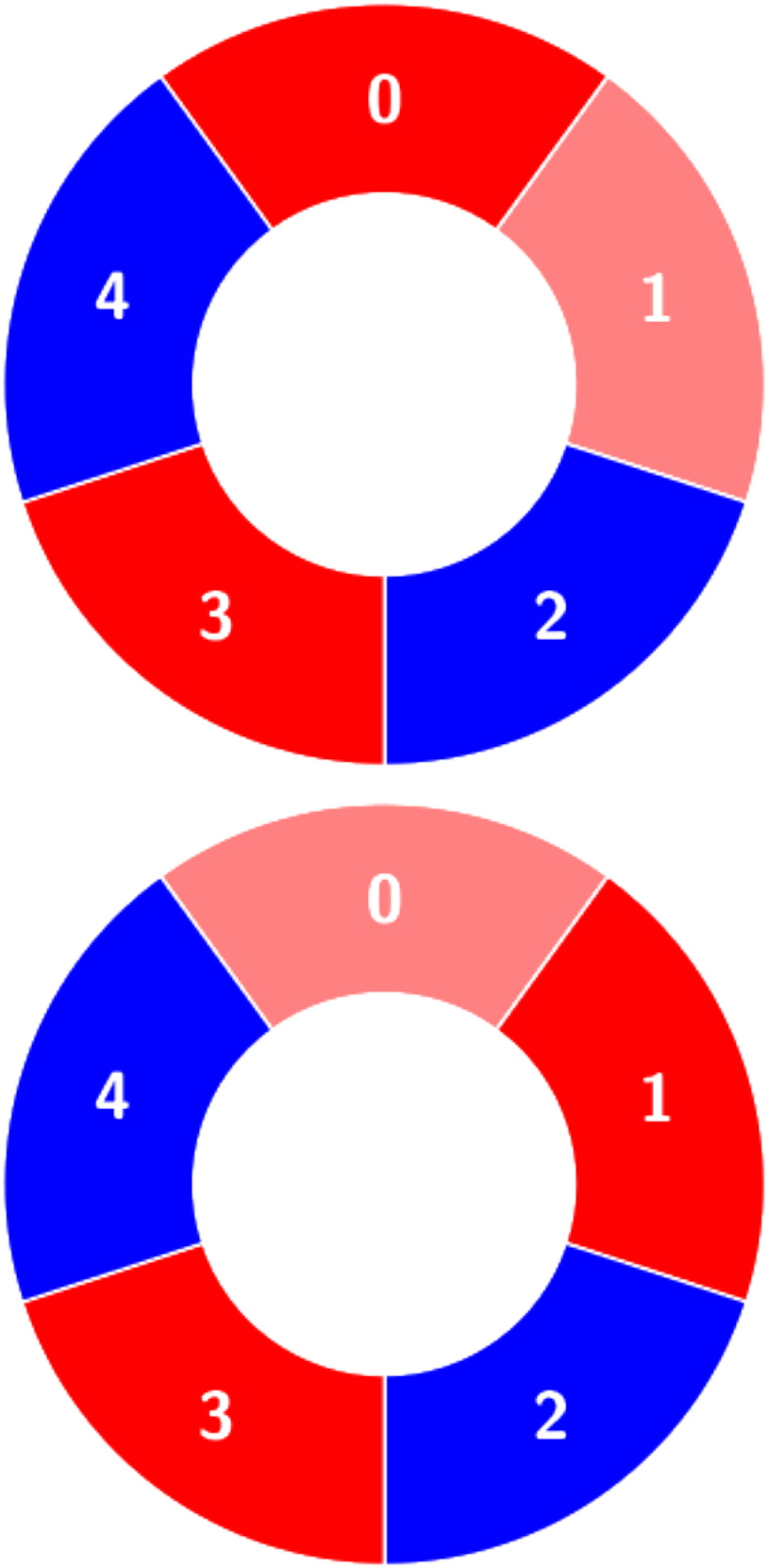
colors[1]

to 0.



Second query:

Count of the alternating groups with size 4:



Example 2:

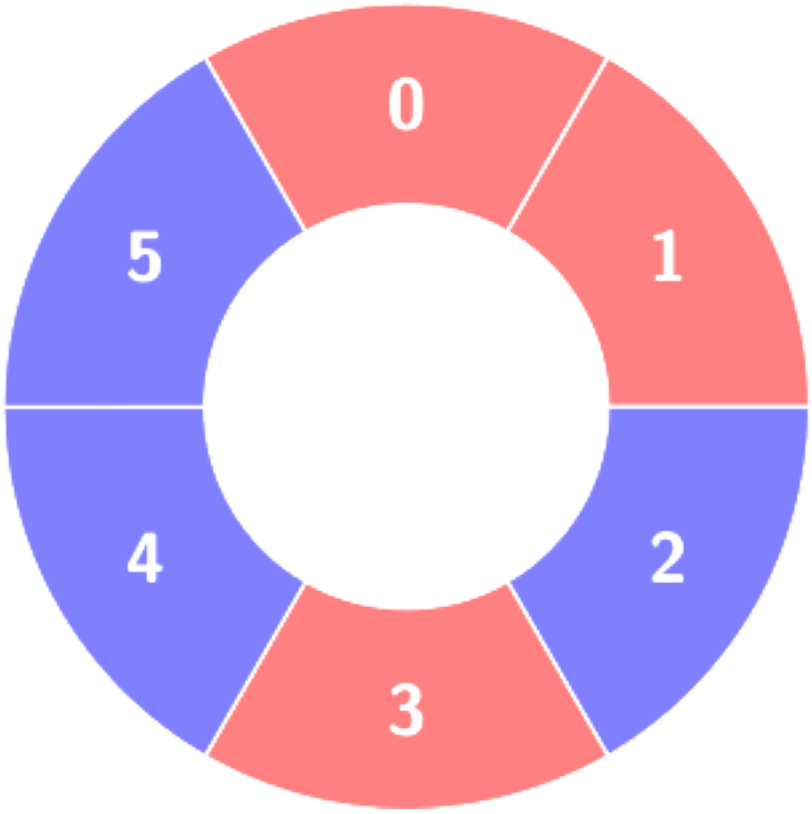
Input:

colors = [0,0,1,0,1,1], queries = [[1,3],[2,3,0],[1,5]]

Output:

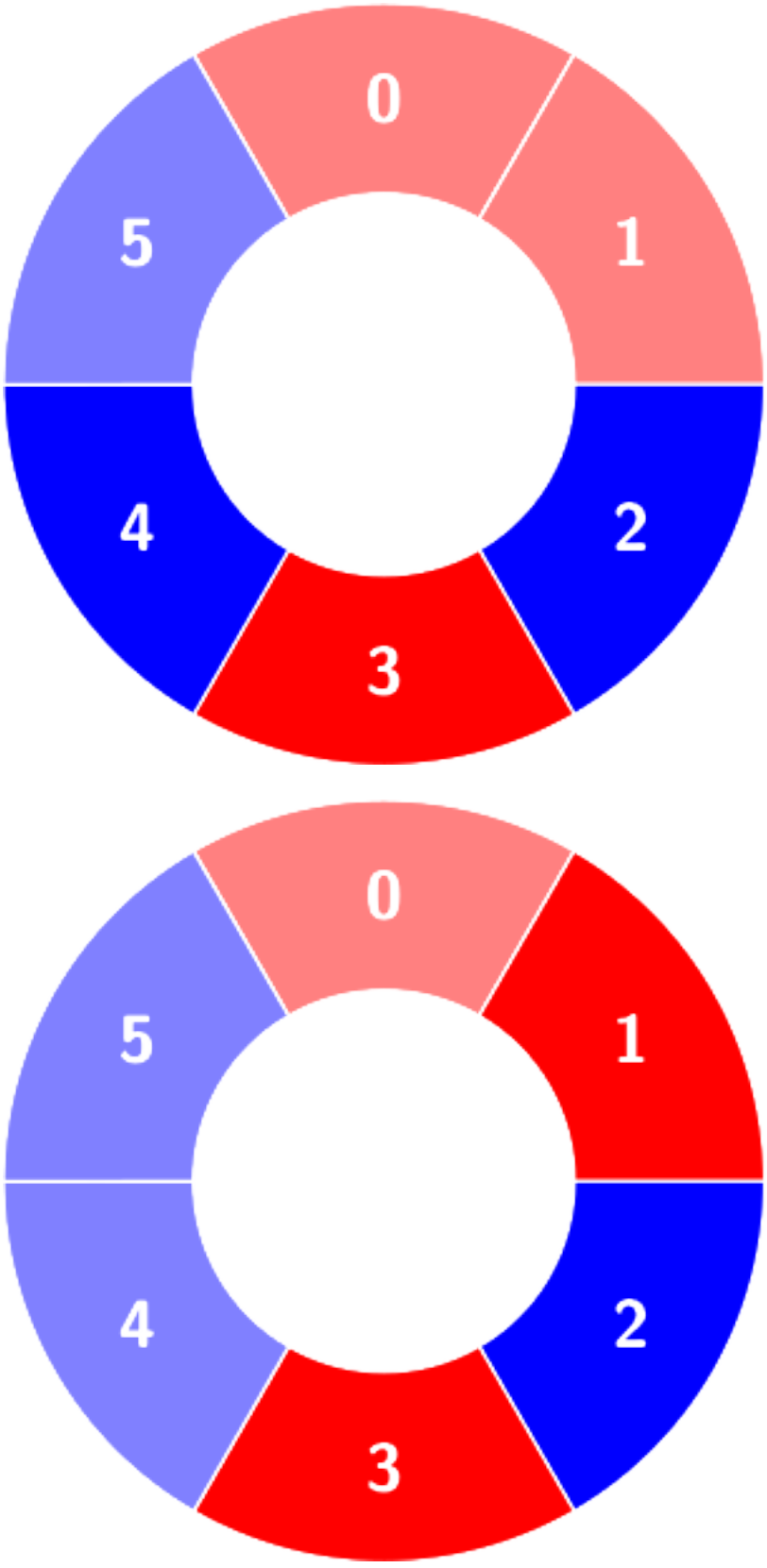
[2,0]

Explanation:



First query:

Count of the alternating groups with size 3:



Second query:

colors

will not change.

Third query: There is no alternating group with size 5.

Constraints:

4 <= colors.length <= 5 * 10

4

0 <= colors[i] <= 1

1 <= queries.length <= 5 * 10

4

queries[i][0] == 1

or

queries[i][0] == 2

For all

i

that:

queries[i][0] == 1

:

queries[i].length == 2

,

3 <= queries[i][1] <= colors.length - 1

queries[i][0] == 2

:

queries[i].length == 3

,

0 <= queries[i][1] <= colors.length - 1

,

0 <= queries[i][2] <= 1

Code Snippets

C++:

```
class Solution {
public:
vector<int> numberOfAlternatingGroups(vector<int>& colors,
vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public List<Integer> numberOfAlternatingGroups(int[] colors, int[][] queries)
{

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function numberOfAlternatingGroups(colors: number[], queries: number[][]):
number[] {

};
```

C#:

```
public class Solution {
public IList<int> NumberOfAlternatingGroups(int[] colors, int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* numberOfAlternatingGroups(int* colors, int colorsSize, int** queries,
int queriesSize, int* queriesColSize, int* returnSize) {
```

```
}
```

Go:

```
func numberOfAlternatingGroups(colors []int, queries [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun numberOfAlternatingGroups(colors: IntArray, queries: Array<IntArray>):
List<Int> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

```
end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Alternating Groups III
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
vector<int> numberOfAlternatingGroups(vector<int>& colors,
vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Alternating Groups III
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public List<Integer> numberOfAlternatingGroups(int[] colors, int[][] queries)
{

}
}
```

Python3 Solution:

```
"""
Problem: Alternating Groups III
Difficulty: Hard
Tags: array, tree

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def numberOfAlternatingGroups(self, colors: List[int], queries:
List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Alternating Groups III
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Alternating Groups III
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function numberOfAlternatingGroups(colors: number[], queries: number[][]):
number[] {

};
```

C# Solution:

```
/*
* Problem: Alternating Groups III
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public IList<int> NumberOfAlternatingGroups(int[] colors, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Alternating Groups III
* Difficulty: Hard
* Tags: array, tree
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* numberOfAlternatingGroups(int* colors, int colorsSize, int** queries,
int queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Alternating Groups III
// Difficulty: Hard
// Tags: array, tree
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func numberOfAlternatingGroups(colors []int, queries [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun numberOfAlternatingGroups(colors: IntArray, queries: Array<IntArray>):
List<Int> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Alternating Groups III
// Difficulty: Hard
// Tags: array, tree
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

```
}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

```
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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