Problem 1550: Three Consecutive Odds

Problem Information

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

Problem Description

Given an integer array

arr

, return

true

if there are three consecutive odd numbers in the array. Otherwise, return

false

.

Example 1:

Input:

arr = [2,6,4,1]

Output:

false

Explanation:

There are no three consecutive odds.

Example 2:

Input:

arr = [1,2,34,3,4,5,7,23,12]

Output:

true

Explanation:

[5,7,23] are three consecutive odds.

Constraints:

1 <= arr.length <= 1000

1 <= arr[i] <= 1000

Code Snippets

C++:

```
class Solution {
public:
bool threeConsecutiveOdds(vector<int>& arr) {

}
};
```

Java:

```
class Solution {
public boolean threeConsecutiveOdds(int[] arr) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function threeConsecutiveOdds(arr: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool ThreeConsecutiveOdds(int[] arr) {

}
}
```

C:

```
bool threeConsecutiveOdds(int* arr, int arrSize) {

}
```

Go:

```
func threeConsecutiveOdds(arr []int) bool {

}
```

Kotlin:

```
class Solution {
fun threeConsecutiveOdds(arr: IntArray): Boolean {

}
}
```

Swift:

```
class Solution {
func threeConsecutiveOdds(_ arr: [Int]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn three_consecutive_odds(arr: Vec<i32>) -> bool {

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @return {Boolean}
def three_consecutive_odds(arr)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @return Boolean
*/
function threeConsecutiveOdds($arr) {

}
}
```

Dart:

```
class Solution {
bool threeConsecutiveOdds(List<int> arr) {

}
}
```

Scala:

```
object Solution {
def threeConsecutiveOdds(arr: Array[Int]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec three_consecutive_odds(arr :: [integer]) :: boolean
def three_consecutive_odds(arr) do

end
end
```

Erlang:

```
-spec three_consecutive_odds(Arr :: [integer()]) -> boolean().
three_consecutive_odds(Arr) ->
.
```

Racket:

```
(define/contract (three-consecutive-odds arr)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Three Consecutive Odds
* 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:
bool threeConsecutiveOdds(vector<int>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Three Consecutive Odds
* 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 boolean threeConsecutiveOdds(int[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Three Consecutive Odds
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 threeConsecutiveOdds(self, arr: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Three Consecutive Odds
* 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[]} arr
* @return {boolean}
*/
var threeConsecutiveOdds = function(arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Three Consecutive Odds
* 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 threeConsecutiveOdds(arr: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Three Consecutive Odds
* 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
*/

public class Solution {
public bool ThreeConsecutiveOdds(int[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Three Consecutive Odds
* 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
*/

bool threeConsecutiveOdds(int* arr, int arrSize) {

}
```

Go Solution:

```
// Problem: Three Consecutive Odds
// 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

func threeConsecutiveOdds(arr []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun threeConsecutiveOdds(arr: IntArray): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func threeConsecutiveOdds(_ arr: [Int]) -> Bool {


}
}
```

Rust Solution:

```rust
// Problem: Three Consecutive Odds
// 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 three_consecutive_odds(arr: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```ruby
# @param {Integer[]} arr
# @return {Boolean}
def three_consecutive_odds(arr)

end
```

PHP Solution:

```php
class Solution {

/**
* @param Integer[] $arr
* @return Boolean
*/
function threeConsecutiveOdds($arr) {

}
}
```

Dart Solution:

```
class Solution {
bool threeConsecutiveOdds(List<int> arr) {

}
}
```

Scala Solution:

```
object Solution {
def threeConsecutiveOdds(arr: Array[Int]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec three_consecutive_odds(arr :: [integer]) :: boolean
def three_consecutive_odds(arr) do

end
end
```

Erlang Solution:

```
-spec three_consecutive_odds(Arr :: [integer()]) -> boolean().
three_consecutive_odds(Arr) ->
.
```

Racket Solution:

```
(define/contract (three-consecutive-odds arr)
(-> (listof exact-integer?) boolean?)
)
```