1550. Three Consecutive Odds

Easy Array Leetcode Link

Problem Description

In this problem, we are given an array of integers, named `arr`. Our objective is to determine whether the array contains a sequence of three consecutive odd numbers. If such a sequence exists, we should return `true`; if not, we should return `false`.

To be classified as odd, a number must have a remainder of 1 when divided by 2. A sequence of three consecutive odd numbers is a subsequence in the array where each of the three numbers appears one after another and each is odd.

Intuition

To solve this problem, we scan through the array, starting from the first element and continuing until the third-to-last element (because we are checking sequences of three). For each element `i`, we need to check that element `i`, `i+1` (the next element), and `i+2` (the element after the next) are all odd.

The simplest way to do this is to check the remainder when each number is divided by 2. This is achieved using the modulo operator `%`. If a number is odd, `number % 2` will equal 1, and if the number is even, it will equal 0. By summing up the remainders of the current element and the next two (`arr[i] % 2 + arr[i + 1] % 2 + arr[i + 2] % 2`), we can quickly verify if all three are odd. If the sum equals 3, it means that each term in the sum must have been 1, confirming that all three numbers are odd. If we find such a sequence, we immediately return `true`. If we reach the end of our checks without finding a sequence, we return `false`.

Iterating through the array and checking each trio of consecutive elements provides us with a clear way to solve this problem.

Solution Approach

The algorithm we use to solve this problem is straightforward linear iteration over the array. We don't need any complex data structures—just the ability to access elements in the array by their index.

The main pattern we utilize is a simple for-loop, which iterates over each element in the array up to the third-to-last element. This is because we need to check each group of three consecutive numbers, and beyond the third-to-last element there are not enough numbers to make up a group of three.

The core logic of the solution is encapsulated in the for-loop:

```python
for i in range(len(arr) - 2):
    if arr[i] % 2 + arr[i + 1] % 2 + arr[i + 2] % 2 == 3:
        return True
```

As we iterate over the array, we check each group of three consecutive elements. With the expression `arr[i] % 2 + arr[i + 1] % 2 + arr[i + 2] % 2`, we are essentially summing up the remainders of dividing each of the three consecutive numbers by 2.

If the sum is equal to 3, we hit our target condition: all three numbers must be odd (since the sum of their remainders when divided by 2 is the maximum it could possibly be for three numbers). When this condition is met, we immediately exit the loop and return `True`.

If we complete the loop without finding three consecutive odd numbers, we exit the loop normally and return `False`. This happens after the loop has checked up to the third-to-last element without success.

No additional space is required other than the input array, making the space complexity of this algorithm O(1), or constant space. The time complexity is O(n), where n is the length of the array, because we may need to check every element in the array in the worst-case scenario.

Example Walkthrough

Let's consider a small example where the given array is `arr = [2, 1, 3, 5, 1]`. We're going to walk through this array to look for three consecutive odd numbers.

Starting with the index i = 0, we check the elements at positions 0, 1, and 2. Arr[i] is 2, arr[i+1] is 1, and arr[i+2] is 3. We calculate the remainder of each when divided by 2, which gives us:

- arr[0] % 2 = 2 % 2 = 0
- arr[1] % 2 = 1 % 2 = 1
- arr[2] % 2 = 3 % 2 = 1

Adding these up, we get: `0 + 1 + 1 = 2`. Since the sum is not 3, not all of these numbers are odd.

Next, we increment i to 1 and perform the sum of remainders for elements at index 1, 2, and 3. Arr[i] is now 1, arr[i+1] is 3, and arr[i+2] is 5. The remainders are:

- arr[1] % 2 = 1 % 2 = 1
- arr[2] % 2 = 3 % 2 = 1
- arr[3] % 2 = 5 % 2 = 1

The sum this time is `1 + 1 + 1 = 3`. Each of the terms in the sum is 1, which indicates that all three numbers (1, 3, 5) are odd, and they are consecutive in the array. Therefore, we have found a sequence that meets the criteria and we return `True`.

In this case, our search would terminate early, and we would not need to continue iterating to the end of the array. If, however, no such sequence of three consecutive odd numbers existed, we would finish the for-loop without returning `True` and would instead return `False` at the end.

This walkthrough demonstrates the described solution approach: linear iteration and conditional checking without any need for complex algorithms or data structures.

Python Solution

```python
from typing import List

class Solution:
    def threeConsecutiveOdds(self, arr: List[int]) -> bool:
        # Loop through the array until the third-last element
        for i in range(len(arr) - 2):
            # Check if the current element and the next two are all odd
            # An odd number % 2 equals 1, so the sum of three odd numbers % 2 equals 3
            if arr[i] % 2 + arr[i + 1] % 2 + arr[i + 2] % 2 == 3:
                return True  # Return True if three consecutive odds are found
        return False  # Return False if no three consecutive odds are found

# Example usage:
# sol = Solution()
# result = sol.threeConsecutiveOdds([2, 6, 4, 1])
# print(result)  # Output will be False

```

Java Solution

```java
class Solution {
    // Method to check if there are any three consecutive odd numbers in the array
    public boolean threeConsecutiveOdds(int[] arr) {
        // Initialize a counter for consecutive odd numbers
        int consecutiveOddsCount = 0;

        // Loop through each value in the array
        for (int value : arr) {
            // If the value is odd (remainder of division by 2 is 1), increment the counter
            if (value % 2 == 1) {
                consecutiveOddsCount++;
            } else {
                // If the value is not odd, reset the counter to 0
                consecutiveOddsCount = 0;
            }

            // If we have found 3 consecutive odd numbers, return true
            if (consecutiveOddsCount == 3) {
                return true;
            }
        }
        // If no three consecutive odd numbers were found, return false
        return false;
    }
}

```

C++ Solution

```cpp
#include <vector> // Include the header for using the vector class

class Solution {
public:
    // Function to check if there are three consecutive odd numbers in the array
    bool threeConsecutiveOdds(vector<int>& arr) {
        int oddCount = 0; // Initialize a counter for consecutive odd numbers

        // Iterate through each element in the array
        for (int value : arr) {
            // If the current value is odd (bitwise AND with 1)
            if (value & 1) {
                ++oddCount; // Increment the odd counter
            } else {
                oddCount = 0; // Reset the counter if the number is even
            }

            // Check if we have found 3 consecutive odds
            if (oddCount == 3) return true;
        }

        // Return false if we finish iterating without finding 3 consecutive odds
        return false;
    }
};

```

Typescript Solution

```typescript
function threeConsecutiveOdds(arr: number[]): boolean {
    // Initialize a counter for consecutive odd numbers
    let consecutiveOddsCount = 0;

    // Iterate over each number in the array
    for (const value of arr) {
        // Check if the current number is odd using bitwise AND with 1
        if (value & 1) {
            // If odd, increment the consecutive odd counter
            ++consecutiveOddsCount;
        } else {
            // If not odd, reset the counter to 0
            consecutiveOddsCount = 0;
        }

        // If we find three consecutive odd numbers, return true
        if (consecutiveOddsCount == 3) {
            return true;
        }
    }

    // If no three consecutive odds found, return false
    return false;
}

```

Time and Space Complexity

The time complexity of the code is `O(n)`, where `n` is the length of the input array. This complexity arises because the code uses a single loop that iterates through the array from the beginning to `n-3`. The loop has a constant time check `arr[i] % 2 + arr[i + 1] % 2 + arr[i + 2] % 2 == 3` inside it, which does not depend on the size of the input, therefore not affecting the overall linear time complexity.

The space complexity of the code is `O(1)`, indicating that the space required is constant and does not depend on the input array size. This is due to the fact that the method only uses a fixed number of single-value variables (`i`) and does not allocate any additional memory that scales with the input size.