Separate Even and Odd Numbers

Difficulty: Easy
Topics: Array, Two Pointers
Companies: Amazon, Microsoft, Google

Problem Description

Given an array of integers `nums`, separate and return all even numbers followed by all odd numbers. The relative order of the even numbers and odd numbers should be maintained as they appear in the original array.

Examples

Example 1:

```
Input: nums = [7,2,8,3,1,4]
Output: [[2,8,4],[7,3,1]]
Explanation:
- Even numbers in order: [2,8,4]
- Odd numbers in order: [7,3,1]
```

Example 2:

```
Input: nums = [1,3,5]
Output: [[],[1,3,5]]
Explanation:
- No even numbers found
- Odd numbers in order: [1,3,5]
```

Example 3:

```
Input: nums = [2,4,6,8]
Output: [[2,4,6,8],[]]
Explanation:
- Even numbers in order: [2,4,6,8]
- No odd numbers found
```

Constraints

- `1 <= nums.length <= 1000`

- `0 <= nums[i] <= 10^6`

Solution Approach

Approach 1: Two Pass Solution

Intuition: The most straightforward approach is to iterate through the array twice:

1. First pass: collect all even numbers
2. Second pass: collect all odd numbers

Algorithm:

cpp

```cpp
class Solution {
public:
    vector<vector<int>> separateEvenOdd(vector<int>& nums) {
        vector<int> evens, odds;

        // First pass: collect even numbers
        for (int num : nums) {
            if (num % 2 == 0) {
                evens.push_back(num);
            }
        }

        // Second pass: collect odd numbers
        for (int num : nums) {
            if (num % 2 == 1) {
                odds.push_back(num);
            }
        }

        return {evens, odds};
    }
};
```

Complexity Analysis:

- **Time Complexity:** O(n), where n is the length of the input array. We iterate through the array twice.
- **Space Complexity:** O(n) for storing the result arrays.

Approach 2: Single Pass Solution (Optimal)

Intuition: We can optimize by using only one pass through the array and separate numbers into even and odd lists simultaneously.

Algorithm:

cpp

```cpp
class Solution {
public:
    vector<vector<int>> separateEvenOdd(vector<int>& nums) {
        vector<int> evens, odds;

        // Single pass: separate even and odd numbers
        for (int num : nums) {
            if (num % 2 == 0) {
                evens.push_back(num);
            } else {
                odds.push_back(num);
            }
        }

        return {evens, odds};
    }
};
```

Complexity Analysis:

- **Time Complexity:** O(n), where n is the length of the input array
- **Space Complexity:** O(n) for storing the result arrays

Alternative Implementations (Input/Output Format)

C Implementation

c

```c
#include<stdio.h>

int main() {
    int n;
    scanf("%d", &n);
    int a[1000];

    // Read n integers
    for(int i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }

    // Print even numbers
    for(int i = 0; i < n; i++) {
        if(a[i] % 2 == 0) {
            printf("%d ", a[i]);
        }
    }
    printf("\n");

    // Print odd numbers
    for(int i = 0; i < n; i++) {
        if(a[i] % 2 == 1) {
            printf("%d ", a[i]);
        }
    }
    printf("\n");

    return 0;
}
```

C++ Implementation

cpp

```cpp
#include<iostream>
#include<vector>
using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> a(n);

    // Read n integers
    for(int i = 0; i < n; i++) {
        cin >> a[i];
    }

    // Print even numbers
    for(int i = 0; i < n; i++) {
        if(a[i] % 2 == 0) {
            cout << a[i] << " ";
        }
    }
    cout << "\n";

    // Print odd numbers
    for(int i = 0; i < n; i++) {
        if(a[i] % 2 == 1) {
            cout << a[i] << " ";
        }
    }
    cout << "\n";

    return 0;
}
```

Java Implementation

java

```java
import java.util.Scanner;

public class Solution {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] a = new int[n];

        // Read n integers
        for(int i = 0; i < n; i++) {
            a[i] = sc.nextInt();
        }

        // Print even numbers
        for(int i = 0; i < n; i++) {
            if(a[i] % 2 == 0) {
                System.out.print(a[i] + " ");
            }
        }
        System.out.println();

        // Print odd numbers
        for(int i = 0; i < n; i++) {
            if(a[i] % 2 == 1) {
                System.out.print(a[i] + " ");
            }
        }
        System.out.println();

        sc.close();
    }
}
```

C# Implementation

csharp

```
using System;

class Program {
    static void Main() {
        int n = int.Parse(Console.ReadLine());
        string[] input = Console.ReadLine().Split();
        int[] a = new int[n];

        // Read n integers
        for(int i = 0; i < n; i++) {
            a[i] = int.Parse(input[i]);
        }

        // Print even numbers
        for(int i = 0; i < n; i++) {
            if(a[i] % 2 == 0) {
                Console.Write(a[i] + " ");
            }
        }
        Console.WriteLine();

        // Print odd numbers
        for(int i = 0; i < n; i++) {
            if(a[i] % 2 == 1) {
                Console.Write(a[i] + " ");
            }
        }
        Console.WriteLine();
    }
}
```

Rust Implementation

rust

```rust
use std::io;

fn main() {
    let mut input = String::new();
    io::stdin().read_line(&mut input).unwrap();
    let n: usize = input.trim().parse().unwrap();

    input.clear();
    io::stdin().read_line(&mut input).unwrap();
    let a: Vec<i32> = input
        .trim()
        .split_whitespace()
        .map(|s| s.parse().unwrap())
        .collect();

    // Print even numbers
    for &num in &a {
        if num % 2 == 0 {
            print!("{} ", num);
        }
    }
    println!();

    // Print odd numbers
    for &num in &a {
        if num % 2 == 1 {
            print!("{} ", num);
        }
    }
    println!();
}
```

Sample Input/Output for all implementations:

```
Input:
5
1 2 3 4 5

Output:
2 4
1 3 5
```

Follow-up Questions

1. What if we wanted to maintain the original array and just return indices of even and odd numbers?
2. How would you solve this if memory usage was extremely limited?
3. Can you implement this using only constant extra space (not counting output)?

Related Problems

- 905. Sort Array By Parity
- 922. Sort Array By Parity II
- 832. Flipping an Image