# 2553. Separate the Digits in an Array

Easy Array Simulation

## Problem Description

The given problem provides an array of positive integers, `nums`. The objective is to produce a new array, `answer`, containing all the individual digits of each number in the original array, with the digits appearing in the same order as they do in those integers. Effectively, the process involves 'separating' the digits of each integer in `nums`. For example, if you have an integer `10921`, you separate its digits to get the sequence `[1,0,9,2,1]`. It is like 'unpacking' each number into its constituent digits and listing them in sequence.

## Intuition

To solve this problem, we can break it down into a few manageable steps. Here's how we can think about the approach:

1. We iterate through each number in the `nums` array since we need to process each number individually.
2. For each integer, we need to separate its digits. A standard way to do this is by continually dividing the number by 10 and collecting the remainders. This process will give us the digits in reverse order.
3. We capture the reverse of the individual digits of an integer in a temporary list to preserve the correct order.
4. After reversing, we append the individual digits into the answer list.
5. We repeat this process for each integer in `nums` until we have processed all integers and collected all their digits, preserving the original order.

## Solution Approach

The solution uses a simple algorithm and basic Python list operations to achieve the desired result. Here's a step-by-step breakdown of the solution implementation:

1. An empty list called `ans` is created. This list will contain the final sequence of all individual digits from each number in `nums`.
2. The process begins by iterating over each number (`x`) in the `nums` array using a `for` loop.
3. Within each iteration, a temporary list called `t` is created to hold the digits of the current number (`x`) in reverse order.
4. A `while` loop runs as long as the current number (`x`) is greater than zero. Inside this loop:
   - The expression `x % 10` is used to get the last digit of `x`.
   - This digit is appended to the temporary list `t`.
   - The number `x` is then divided by 10 (using floor division `x //= 10`) to remove its last digit.
5. After the `while` loop exits (meaning `x` is now zero and all digits have been processed), the list `t` contains the digits of `x` in reverse order. To correct the order, we reverse `t` using `t[::-1]`.
6. The reversed list of digits is then extended into the `ans` list with `ans.extend(t[::-1])`. This means the digits of `x` are now added to `ans` in the correct order.
7. Steps 3 to 6 are repeated for each number in the `nums` array.
8. After the `for` loop completes, the `ans` list, now containing the individual digits of all the numbers in their correct order, is returned as the result.

Notice how the code makes use of modulo and floor division operations to separate the digits, and list operations like append and extend to collect digits in the correct order. Using these operations and control structures effectively, the code walks through each integer, extracts its digits, and assembles the final answer, while maintaining both the inner order of the digits in each number and overall order in which the numbers appear in the input list.

### Example Walkthrough

Let's take a simple example to illustrate the solution approach. Consider an array `nums = [123, 45]`. We want to create an array that 'unpacks' each of these numbers into its individual digits `[1, 2, 3, 4, 5]`.

Here is how the solution will walk through this example:

1. An empty list `ans` is created to store the answer.
2. We start with the first number in the `nums` array, which is `123`.
3. A temporary list `t` is initialized to hold the digits of `123` in reverse order.
4. We enter a `while` loop because `123` is greater than zero. Inside the loop:
   - We calculate `123 % 10` which equals `3`. We append `3` to the list `t`.
   - We then divide `123` by `10` using floor division, so `123` becomes `12`.
5. The loop runs again because `12` is still greater than zero.
   - Calculating `12 % 10` gives us `2`. We append `2` to `t`.
   - Floor division of `12` by `10` reduces it to `1`.
6. The loop runs a final time with the value of `1`.
   - We append `1 % 10` (which is `1`) to `t`.
   - Floor division of `1` by `10` gives us `0`, and the loop exits as `x` is now zero.
7. The list `t` now contains `[3, 2, 1]`. We reverse it to get `[1, 2, 3]` and extend `ans` by this list.
8. Now we move to the second number, `45`, and repeat steps 3 to 7.
   - `t` starts empty, we add `5` then `4` after iterations of the loop.
   - We reverse `[5, 4]` to get `[4, 5]` and extend it to `ans`.
9. At the end of the iteration, `ans` now contains `[1, 2, 3, 4, 5]`.
10. We return the `ans` list as the result.

This walk-through shows how the algorithm correctly takes each integer in the array `nums` and breaks it down into individual digits, preserving the order within and between the numbers in the array.

## Solution Implementation

### Python

```python
from typing import List

class Solution:
    def separateDigits(self, nums: List[int]) -> List[int]:
        # Initialize an empty list to store the result
        result = []

        # Iterate over each number in the input list
        for number in nums:
            # Initialize a temporary list to store the digits of the current number
            temp = []

            # Loop to separate out each digit of the current number
            while number:
                # Append the last digit to the temporary list
                temp.append(number % 10)
                # Remove the last digit from the current number
                number //= 10

            # Reverse the temporary list because digits are stored from least significant to most significant
            # Then extend the result list with the reversed list of digits
            result.extend(temp[::-1])

        # Return the result list containing all digits in order
        return result

# Example usage:
# solution = Solution()
# print(solution.separateDigits([123, 456])) # Output would be [1, 2, 3, 4, 5, 6]
```

### Java

```java
class Solution {

    // Method to separate digits of each number in an array and return a new array with all the digits
    public int[] separateDigits(int[] nums) {

        // Initialize a list to hold individual digits
        List<Integer> result = new ArrayList<>();

        // Iterate over each number in the input array
        for (int number : nums) {

            // List to temporarily hold the digits of the current number
            List<Integer> digits = new ArrayList<>();

            // Extract digits from the number and add them to the temporary list
            while (number > 0) {
                int digit = number % 10; // get the last digit
                digits.add(digit); // add digit to the list
                number /= 10; // remove the last digit from the number
            }

            // Since digits are collected in reverse order, reverse the list to correct the order
            Collections.reverse(digits);

            // Add all the digits to the result list
            result.addAll(digits);
        }

        // Convert the List<Integer> to an int array
        int[] answer = new int[result.size()];
        for (int i = 0; i < answer.length; i++) {
            answer[i] = result.get(i); // Retrieve each integer from result list and store it in the array
        }

        // Return the array with separated digits
        return answer;
    }
}
```

### C++

```cpp
#include <vector> // Include the necessary header for std::vector

class Solution {
public:
    // Function to separate digits of numbers in a vector and return them as a new vector
    vector<int> separateDigits(vector<int>& nums) {
        vector<int> result; // This will store the final sequence of digits

        // Loop through all numbers in the input vector
        for (int number : nums) {
            vector<int> temp; // Temporary vector to store the digits of the current number

            // Extract digits of the current number from the end to the start
            for (; number != 0; number /= 10) { // Continue until the number is 0
                temp.push_back(number % 10); // Get the last digit and push it into temp
            }

            // While there are still digits in the temp vector
            while (!temp.empty()) {
                result.push_back(temp.back()); // Add the last digit from temp to the result vector
                temp.pop_back(); // Remove the last element from temp
            }
        }

        return result; // Return the final digit sequence
    }
};
```

### TypeScript

```typescript
function separateDigits(nums: number[]): number[] {
    // We will store the final array of separated digits here
    const separatedDigits: number[] = [];

    // Iterate over each number in the array
    for (let num of nums) {
        // Temporary array to store the digits of the current number
        const digits: number[] = [];

        // Extract digits of the current number and add them to the 'digits' array
        while (num > 0) {
            // Get the last digit of 'num' by modulo 10 (num % 10)
            digits.push(num % 10);
            // Remove the last digit from 'num'
            num = Math.floor(num / 10);
        }

        // 'digits' array is in reverse order, so we reverse it to maintain the original order
        separatedDigits.push(...digits.reverse());
    }

    // Return the array containing all the separated digits in correct order
    return separatedDigits;
}
```

```python
from typing import List

class Solution:
    def separateDigits(self, nums: List[int]) -> List[int]:
        # Initialize an empty list to store the result
        result = []

        # Iterate over each number in the input list
        for number in nums:
            # Initialize a temporary list to store the digits of the current number
            temp = []

            # Loop to separate out each digit of the current number
            while number:
                # Append the last digit to the temporary list
                temp.append(number % 10)
                # Remove the last digit from the current number
                number //= 10

            # Reverse the temporary list because digits are stored from least significant to most significant
            # Then extend the result list with the reversed list of digits
            result.extend(temp[::-1])

        # Return the result list containing all digits in order
        return result

# Example usage:
# solution = Solution()
# print(solution.separateDigits([123, 456])) # Output would be [1, 2, 3, 4, 5, 6]
```

## Time and Space Complexity

The time complexity of the given code can be analyzed as follows:

- There is a loop that iterates over each number in the input list `nums`.
- For each number, the inner while loop executes once for each digit in the number. So if a number `x` has `k` digits, the while loop iterates `k` times.

Considering an input list with `n` numbers, and each number has an average of `d` digits, the total operations for separating digits of all numbers would be `O(n * d)`. Therefore, the time complexity is `O(n * d)`.

The space complexity is determined by:

- The list `ans` that holds the individual digits of all numbers. In the worst case, it will hold all `n * d` digits from all numbers in the input list.
- The temporary list `t` that stores the digits of a single number in reverse. Since it's reused for each number and extends the `ans` list immediately after, it doesn't increase the maximal memory footprint with respect to the number of total digits.

Given space is generally calculated in terms of the additional space required by the algorithm, not including the space for the input itself. The space complexity of the given algorithm is `O(n * d)` as the `ans` list may hold all digits of all numbers, though in practice, only the maximum number of digits in a single number is simultaneously held in the temporary list `t`.

In conclusion, the time complexity is `O(n * d)` and the space complexity is `O(n * d)`.