1929. Concatenation of Array

Easy Array

Leetcode Link

Problem Description

The problem presents us with a challenge to create a new array that is exactly twice the length of the given integer array `nums`. The new array should have a structure such that the first half is a copy of the original `nums` array, and the second half is an exact repeat of the same `nums` array. This means for each element `nums[i]` where `i` is within the bounds of the original array, we want to set `ans[i]` and `ans[i + n]` (where `n` is the length of `nums`) to be equal to `nums[i]`. In essence, we are concatenating `nums` onto itself to create the resultant array `ans`.

Intuition

To arrive at the solution, we can realize that Python's list concatenation operator `+` performs the task of combining two lists end-to-end to form a single list. In this case, since we need to repeat the array `nums` twice, we can make use of this operator to concatenate `nums` with itself - `nums + nums`. This straightforward operation yields a new array which has all elements of `nums` followed by `nums` again, and thus fulfills the problem's requirement with minimal steps and in an easy-to-understand manner.

Solution Approach

The solution makes use of the in-built functionality of Python's list type to solve the problem. Here's a step-by-step walk-through of the implementation:

1. **Concatenation Operation (`+`)**: In Python, the `+` operator is used to concatenate two lists. So, in this case, we use `nums + nums` to create a new list. When the `+` operator is used between two lists, Python creates a new list by taking the elements of the first list (on the left-hand side) and appending the elements of the second list (on the right-hand side) in the same order.
2. **Returning the New List**: The concatenated list is then returned as the output. The beauty of this approach is its simplicity and efficiency; no explicit loops or additional storage is needed, and thus the code is concise and easy to read. Since Python internally manages the creation of the new list during the concatenation, we also do not need to be concerned with the underlying details of how the new memory is allocated and how the elements are copied over.

The Python interpreter handles the concatenation operation efficiently, and since it is a single line of code, this solution is both performant and elegant for this problem.

Here's the code snippet encapsulating this approach:

```
class Solution:
    def getConcatenation(self, nums: List[int]) -> List[int]:
        return nums + nums
```

This code defines a method in a class `Solution`, which takes a list `nums` as input and returns the concatenated list by simply adding `nums` to itself.

Example Walkthrough

Let's consider an example to illustrate the solution approach. Assume we have the following array `nums`:

```
nums = [1, 2, 3]
```

With this array, we want to create a new array `ans` that contains each element of `nums` repeated once, resulting in `ans` being twice as long as `nums`.

Following the solution approach:

1. **Concatenation Operation (`+`)**: We apply the concatenation operator to `nums`, effectively doubling it by appending `nums` to the end of itself.

   Before operation:

   ```
   nums = [1, 2, 3]
   ```

   After operation using `nums + nums`:

   ```
   ans = [1, 2, 3] + [1, 2, 3]
   ```

2. **Returning the New List**: The concatenated list result for `ans` after using the operation `nums + nums` would be:

   ```
   ans = [1, 2, 3, 1, 2, 3]
   ```

The new list `ans` is now `[1, 2, 3, 1, 2, 3]`, which is exactly what is expected according to the problem description. It retains the order of elements as per the original list and successfully repeats the sequence to create a list of twice the length.

This demonstrates that the solution provided is effective for the given problem, producing the desired result with simplicity and efficiency.

Python Solution

```
# Definition of the Solution class
class Solution:
    # Method to get the concatenation of the list 'nums'
    # It takes a list of integers as input and returns a new list
    def getConcatenation(self, nums: List[int]) -> List[int]:
        # The list 'nums' is concatenated with itself
        # and the result is returned
        ans = nums * 2  # replicates 'nums' twice

        # Return the concatenated list
        return ans

```

Java Solution

```
class Solution {
    // Method to concatenate the array with itself
    public int[] getConcatenation(int[] nums) {
        // Get the length of the input array
        int n = nums.length;
        // Create a new array that is twice the length of the input array
        int[] result = new int[n * 2];
        // Loop through the new array to fill it with elements from the input array
        for (int i = 0; i < n * 2; ++i) {
            // Since we are concatenating the input array with itself,
            // we use the modulo operator to wrap around the index for array 'nums'
            // This is because 'nums' has a length of 'n' and 'result' has a length of 'n * 2'
            // Therefore, when 'i' is greater than or equal to 'n',
            // 'i % n' will start from 0 again, essentially repeating the array 'nums'
            result[i] = nums[i % n];
        }
        // Return the concatenated array
        return result;
    }
}

```

C++ Solution

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

class Solution {
public:
    // Function to concatenate an array with itself
    // @param nums: The original array of integers
    // @return: Returns a new vector containing two copies of the original array
    vector<int> getConcatenation(vector<int>& nums) {
        int original_size = nums.size(); // Store the original size of nums
        nums.reserve(2 * original_size); // Reserve total space in advance for efficiency

        // Loop through the original array and append each element to the end of it
        for (int i = 0; i < original_size; ++i) {
            nums.push_back(nums[i]);
        }

        return nums; // Return the modified vector containing two concatenated copies
    }
};

```

Typescript Solution

```
// Function to concatenate a given array with itself
function getConcatenation(nums: number[]): number[] {
    // Use the spread operator to concatenate the array 'nums' with itself
    const concatenatedArray: number[] = [...nums, ...nums];

    // Return the resulting concatenated array
    return concatenatedArray;
}

```

Time and Space Complexity

The time complexity of the code is `O(n)`, where `n` is the length of the input list `nums`. This is because the code concatenates the list `nums` with itself, an operation which requires iterating over the length of `nums` once for the concatenation process.

The space complexity of the code is also `O(n)`. In Python, concatenating two lists using the `+` operator creates a new list with a combined length of the two lists, thus requiring additional space proportional to the sum of the lengths of the two lists. Here, since both lists are the same, the total space required for the new list is proportional to twice the length of the input list `nums`.