1929. Concatenation of Array

<https://leetcode.com/problems/concatenation-of-array/>

1. **Listen**

**Problem Statement:**

Given an integer array nums of length n, you want to create an array ans of length 2n where ans[i] == nums[i] and ans[i + n] == nums[i] for 0 <= i < n(**0-indexed**).

Specifically, ans is the **concatenation** of two numsarrays.

Return *the array*ans.

**Input:**

Integer array **nums** of **length n**

**Goal:**

Create an array **ans** of **length 2n** where

ans[i] == nums[i]

and

ans[i + n] == nums[i]

for

0 <= i < n(**0-indexed**).

**Return:**

Return *the array*ans.

1. **Examples**

**Example 1:**

**Input:** nums = [1,2,1]

**Output:** [1,2,1,1,2,1]

**Explanation:** The array ans is formed as follows:

- ans = [nums[0],nums[1],nums[2],nums[0],nums[1],nums[2]]

- ans = [1,2,1,1,2,1]

**Example 2:**

**Input:** nums = [1,3,2,1]

**Output:** [1,3,2,1,1,3,2,1]

**Explanation:** The array ans is formed as follows:

- ans = [nums[0],nums[1],nums[2],nums[3],nums[0],nums[1],nums[2],nums[3]]

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

**Constraints:**

* n == nums.length
* 1 <= n <= 1000
* 1 <= nums[i] <= 1000

**Test Cases:**

* Array of length 1
* Array of length 1000
* Array of length 10

**Questions:**

* Are we allowed in edit the input array (give the answer in-place?)

1. **Brute Force**

**Solution 1:**

**Walkthrough**

We can simply create a new array named ans that is double the size of the input array nums.

Then, we can iterate over nums and set the value of ans[i] and ans[i+nums.length] equal to ans[i] because the constraints call for ans[i] == nums[i] and ans[i + n] == nums[i].

**Algorithm**

initialize ans array double the length of nums

for every element in nums

ans[i] = nums[i]

ans[i+n] = nums[i]

**Big O**

Time Complexity: O(N)

Space Complexity: O(2N) = O(N)

1. **Optimize**

If this were a different data structure, we could perhaps edit the input collection in-place. However, since it is an array, we would have to create an array double the size of the input array, because the size of arrays are not dynamic.

1. **Implement**

public int[] getConcatenation(int[] nums) {

int[] res = new int[nums.length \* 2];

for(int i = 0; i < nums.length; i++)

{

res[i] = nums[i];

res[i + nums.length] = nums[i];

}

return res;

}

1. **Test**

**Conceptual Test**

The potential for errors comes from these main two calculations inside the loop.

res[i] = nums[i];

res[i + nums.length] = nums[i];

If we use the following example:

Input: [1,2,1]

Output: [1,2,1,1,2,1]

Then we can see test the values to see if we go beyond the bounds of the array.

We just need to make sure the final calculation does not go beyond the size of the res array.

nums[0] = 1;

nums[0+3] = 1;

nums[1] = 2;

nums[1+3] = 1;

nums[2] = 1;

nums[2+3] = 1;

We now break out of the loop because we have visited all elements in the nums input array.

Since we have not gone beyond the bounds the res array, and since the output array is correct, it seems as though this code is working.