Problem 1929: Concatenation of Array

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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

nums

arrays.

Return

the array

ans

.

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

Code Snippets

C++:

```
class Solution {
public:
vector<int> getConcatenation(vector<int>& nums) {

}
```

```
};
```

Java:

```
class Solution {
public int[] getConcatenation(int[] nums) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def getConcatenation(self, nums):
"""
:type nums: List[int]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @return {number[]}
*/
var getConcatenation = function(nums) {

};
```

TypeScript:

```
function getConcatenation(nums: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] GetConcatenation(int[] nums) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* getConcatenation(int* nums, int numsSize, int* returnSize) {

}
```

Go:

```
func getConcatenation(nums []int) []int {

}
```

Kotlin:

```
class Solution {
fun getConcatenation(nums: IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func getConcatenation(_ nums: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn get_concatenation(nums: Vec<i32>) -> Vec<i32> {

}
```

```
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer[]}
def get_concatenation(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer[]
*/
function getConcatenation($nums) {

}
}
```

Dart:

```
class Solution {
List<int> getConcatenation(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def getConcatenation(nums: Array[Int]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec get_concatenation(nums :: [integer]) :: [integer]
def get_concatenation(nums) do

end
end
```

Erlang:

```
-spec get_concatenation(Nums :: [integer()]) -> [integer()].
get_concatenation(Nums) ->
.
```

Racket:

```
(define/contract (get-concatenation nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Concatenation of Array
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<int> getConcatenation(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Concatenation of Array
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Concatenation of Array
Difficulty: Easy
Tags: array

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def getConcatenation(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def getConcatenation(self, nums):
"""
:type nums: List[int]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Concatenation of Array
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[]} nums
* @return {number[]}
*/
var getConcatenation = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Concatenation of Array
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function getConcatenation(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Concatenation of Array
* Difficulty: Easy
* Tags: array
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int[] GetConcatenation(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Concatenation of Array
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* getConcatenation(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Concatenation of Array
// Difficulty: Easy
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func getConcatenation(nums []int) []int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun getConcatenation(nums: IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func getConcatenation(_ nums: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Concatenation of Array
// Difficulty: Easy
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn get_concatenation(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer[]}
def get_concatenation(nums)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer[]
*/
function getConcatenation($nums) {

}
}
```

Dart Solution:

```
class Solution {
List<int> getConcatenation(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def getConcatenation(nums: Array[Int]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec get_concatenation(nums :: [integer]) :: [integer]
def get_concatenation(nums) do

end
end
```

Erlang Solution:

```
-spec get_concatenation(Nums :: [integer()]) -> [integer()].
get_concatenation(Nums) ->
.
```

Racket Solution:

```
(define/contract (get-concatenation nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```