Problem 1389: Create Target Array in the Given Order

Problem Information

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

Problem Description

Given two arrays of integers

nums

and

index

. Your task is to create

target

array under the following rules:

Initially

target

array is empty.

From left to right read nums[i] and index[i], insert at index

index[i]

the value

nums[i]

in

target

array.

Repeat the previous step until there are no elements to read in

nums

and

index.

Return the

target

array.

It is guaranteed that the insertion operations will be valid.

Example 1:

Input:

nums = [0,1,2,3,4], index = [0,1,2,2,1]

Output:

[0,4,1,3,2]

Explanation:

nums index target 0 0 [0] 1 1 [0,1] 2 2 [0,1,2] 3 2 [0,1,3,2] 4 1 [0,4,1,3,2]

Example 2:

Input:

nums = [1,2,3,4,0], index = [0,1,2,3,0]

Output:

[0,1,2,3,4]

Explanation:

nums index target 1 0 [1] 2 1 [1,2] 3 2 [1,2,3] 4 3 [1,2,3,4] 0 0 [0,1,2,3,4]

Example 3:

Input:

nums = [1], index = [0]

Output:

[1]

Constraints:

1 <= nums.length, index.length <= 100

nums.length == index.length

0 <= nums[i] <= 100

0 <= index[i] <= i

Code Snippets

C++:

```
class Solution {
public:
vector<int> createTargetArray(vector<int>& nums, vector<int>& index) {

}
};
```

Java:

```
class Solution {
public int[] createTargetArray(int[] nums, int[] index) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function createTargetArray(nums: number[], index: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] CreateTargetArray(int[] nums, int[] index) {

}
}
```

C:

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

}
```

Go:

```
func createTargetArray(nums []int, index []int) []int {

}
```

Kotlin:

```
class Solution {
fun createTargetArray(nums: IntArray, index: IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func createTargetArray(_ nums: [Int], _ index: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn create_target_array(nums: Vec<i32>, index: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[]} index
# @return {Integer[]}
def create_target_array(nums, index)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $index
* @return Integer[]
*/
function createTargetArray($nums, $index) {

}
}
```

Dart:

```
class Solution {
List<int> createTargetArray(List<int> nums, List<int> index) {

}
}
```

Scala:

```
object Solution {
def createTargetArray(nums: Array[Int], index: Array[Int]): Array[Int] = {
```

```
}
}
```

Elixir:

```
defmodule Solution do
@spec create_target_array(nums :: [integer], index :: [integer]) :: [integer]
def create_target_array(nums, index) do

end
end
```

Erlang:

```
-spec create_target_array(Nums :: [integer()], Index :: [integer()]) ->
[integer()].
create_target_array(Nums, Index) ->
.
```

Racket:

```
(define/contract (create-target-array nums index)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Create Target Array in the Given Order
* 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> createTargetArray(vector<int>& nums, vector<int>& index) {

}
};
```

Java Solution:

```
/**
* Problem: Create Target Array in the Given Order
* 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[] createTargetArray(int[] nums, int[] index) {

}
}
```

Python3 Solution:

```
"""
Problem: Create Target Array in the Given Order
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 createTargetArray(self, nums: List[int], index: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Create Target Array in the Given Order
* 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
* @param {number[]} index
* @return {number[]}
*/
var createTargetArray = function(nums, index) {

};
```

TypeScript Solution:

```
/**
* Problem: Create Target Array in the Given Order
* 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 createTargetArray(nums: number[], index: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Create Target Array in the Given Order
* 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[] CreateTargetArray(int[] nums, int[] index) {

}
}
```

C Solution:

```
/*
* Problem: Create Target Array in the Given Order
* 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* createTargetArray(int* nums, int numsSize, int* index, int indexSize,
int* returnSize) {
```

```
}
```

Go Solution:

```
// Problem: Create Target Array in the Given Order
// 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 createTargetArray(nums []int, index []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun createTargetArray(nums: IntArray, index: IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func createTargetArray(_ nums: [Int], _ index: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Create Target Array in the Given Order
// 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 create_target_array(nums: Vec<i32>, index: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[]} index
# @return {Integer[]}
def create_target_array(nums, index)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[] $index
* @return Integer[]
*/
function createTargetArray($nums, $index) {

}
}
```

Dart Solution:

```
class Solution {
List<int> createTargetArray(List<int> nums, List<int> index) {

}
}
```

Scala Solution:

```
object Solution {
def createTargetArray(nums: Array[Int], index: Array[Int]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec create_target_array(nums :: [integer], index :: [integer]) :: [integer]
def create_target_array(nums, index) do

end
end
```

Erlang Solution:

```
-spec create_target_array(Nums :: [integer()], Index :: [integer()]) ->
[integer()].
create_target_array(Nums, Index) ->
.
```

Racket Solution:

```
(define/contract (create-target-array nums index)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```