Problem 1470: Shuffle the Array

Problem Information

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

Problem Description

Given the array

nums

consisting of

2n

elements in the form

[x

1

,x

2

,...,x

n

,y

1

,y

2

,...,y

n

]

.

Return the array in the form

[x

1

,y

1

,x

2

,y

2

,...,x

n

,y

n

]

.

Example 1:

Input:

nums = [2,5,1,3,4,7], n = 3

Output:

[2,3,5,4,1,7]

Explanation:

Since x

1

=2, x

2

=5, x

3

=1, y

1

=3, y

2

=4, y

3

=7 then the answer is [2,3,5,4,1,7].

Example 2:

Input:

nums = [1,2,3,4,4,3,2,1], n = 4

Output:

[1,4,2,3,3,2,4,1]

Example 3:

Input:

nums = [1,1,2,2], n = 2

Output:

[1,2,1,2]

Constraints:

1 <= n <= 500

nums.length == 2n

1 <= nums[i] <= 10^3

Code Snippets

C++:

```
class Solution {
public:
vector<int> shuffle(vector<int>& nums, int n) {
```

```
}
};
```

Java:

```
class Solution {
public int[] shuffle(int[] nums, int n) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function shuffle(nums: number[], n: number): number[] {

};
```

C#:

```
public class Solution {
public int[] Shuffle(int[] nums, int n) {

}
}
```

C:

```


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

}
```

Go:

```
func shuffle(nums []int, n int) []int {

}
```

Kotlin:

```
class Solution {
fun shuffle(nums: IntArray, n: Int): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} n
# @return {Integer[]}
def shuffle(nums, n)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $n
* @return Integer[]
*/
function shuffle($nums, $n) {

}
}
```

Scala:

```
object Solution {
def shuffle(nums: Array[Int], n: Int): Array[Int] = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Shuffle the 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> shuffle(vector<int>& nums, int n) {

}
};
```

Java Solution:

```
/**
* Problem: Shuffle the 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[] shuffle(int[] nums, int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Shuffle the 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 shuffle(self, nums: List[int], n: int) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Shuffle the 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
* @param {number} n
* @return {number[]}
*/
var shuffle = function(nums, n) {

};
```

TypeScript Solution:

```
/**
* Problem: Shuffle the 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 shuffle(nums: number[], n: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Shuffle the 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[] Shuffle(int[] nums, int n) {

}
}
```

C Solution:

```
/*
* Problem: Shuffle the 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* shuffle(int* nums, int numsSize, int n, int* returnSize){

}
```

Go Solution:

```
// Problem: Shuffle the 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 shuffle(nums []int, n int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun shuffle(nums: IntArray, n: Int): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Shuffle the 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 shuffle(nums: Vec<i32>, n: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} n
# @return {Integer[]}
def shuffle(nums, n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $n
* @return Integer[]
*/
function shuffle($nums, $n) {

}
}
```

Scala Solution:

```
object Solution {
def shuffle(nums: Array[Int], n: Int): Array[Int] = {

}
}
```