Problem 2974: Minimum Number Game

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

of

even

length and there is also an empty array

arr

. Alice and Bob decided to play a game where in every round Alice and Bob will do one move. The rules of the game are as follows:

Every round, first Alice will remove the

minimum

element from

nums

, and then Bob does the same.

Now, first Bob will append the removed element in the array

arr

, and then Alice does the same.

The game continues until

nums

becomes empty.

Return

the resulting array

arr

.

Example 1:

Input:

nums = [5,4,2,3]

Output:

[3,2,5,4]

Explanation:

In round one, first Alice removes 2 and then Bob removes 3. Then in arr firstly Bob appends 3 and then Alice appends 2. So arr = [3,2]. At the begining of round two, nums = [5,4]. Now, first Alice removes 4 and then Bob removes 5. Then both append in arr which becomes [3,2,5,4].

Example 2:

Input:

nums = [2,5]

Output:

[5,2]

Explanation:

In round one, first Alice removes 2 and then Bob removes 5. Then in arr firstly Bob appends and then Alice appends. So arr = [5,2].

Constraints:

2 <= nums.length <= 100

1 <= nums[i] <= 100

nums.length % 2 == 0

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
```

```
int* numberGame(int* nums, int numsSize, int* returnSize) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (number-game nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number Game
* Difficulty: Easy
* Tags: array, sort, queue, heap
*
* 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> numberGame(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number Game
* Difficulty: Easy
* Tags: array, sort, queue, heap
*
* 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[] numberGame(int[] nums) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number Game
Difficulty: Easy
Tags: array, sort, queue, heap

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 numberGame(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Number Game
* Difficulty: Easy
* Tags: array, sort, queue, heap
*
* 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 numberGame = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number Game
* Difficulty: Easy
* Tags: array, sort, queue, heap
*
* 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 numberGame(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Minimum Number Game
* Difficulty: Easy
* Tags: array, sort, queue, heap
*
* 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[] NumberGame(int[] nums) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Minimum Number Game
* Difficulty: Easy
* Tags: array, sort, queue, heap
*
* 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* numberGame(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Minimum Number Game
// Difficulty: Easy
// Tags: array, sort, queue, heap
//
// 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 numberGame(nums []int) []int {

}
```

Kotlin Solution:

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

}
```

```
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Number Game
// Difficulty: Easy
// Tags: array, sort, queue, heap
//
// 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 number_game(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer[]
```

```
*/
function numberGame($nums) {

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (number-game nums)
(-> (listof exact-integer?) (listof exact-integer?))
```

```
)
```