Problem 2293: Min Max Game

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

whose length is a power of

2

.

Apply the following algorithm on

nums

:

Let

n

be the length of

nums

. If

n == 1

,

end

the process. Otherwise,

create

a new

0-indexed

integer array

newNums

of length

n / 2

.

For every

even

index

i

where

0 <= i < n / 2

,

assign

the value of

newNums[i]

as

min(nums[2 * i], nums[2 * i + 1])

.

For every

odd

index

i

where

0 <= i < n / 2

,

assign

the value of

newNums[i]

as

max(nums[2 * i], nums[2 * i + 1])

.

Replace

the array

nums

with

newNums

.

Repeat

the entire process starting from step 1.

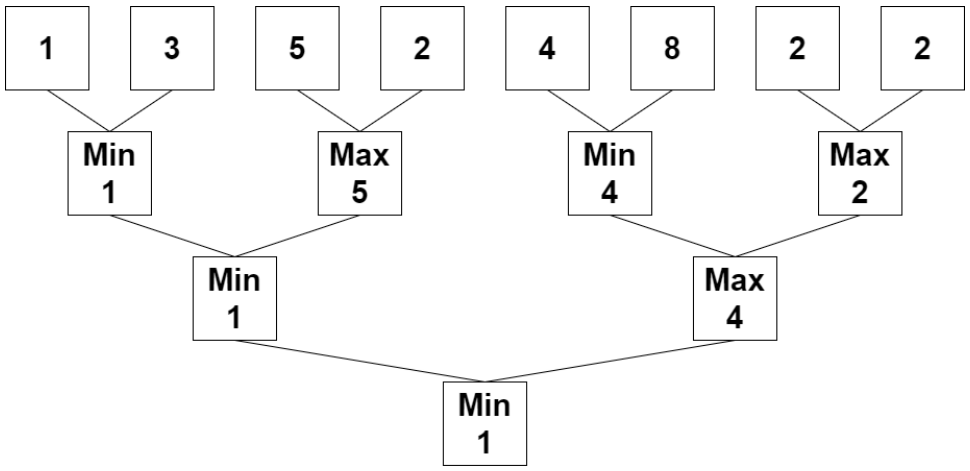
Return

the last number that remains in

nums

after applying the algorithm.

Example 1:



Input:

nums = [1,3,5,2,4,8,2,2]

Output:

1

Explanation:

The following arrays are the results of applying the algorithm repeatedly. First: nums = [1,5,4,2] Second: nums = [1,4] Third: nums = [1] 1 is the last remaining number, so we return 1.

Example 2:

Input:

nums = [3]

Output:

3

Explanation:

3 is already the last remaining number, so we return 3.

Constraints:

1 <= nums.length <= 1024

1 <= nums[i] <= 10

9

nums.length

is a power of

2

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
```

```
 * @return {number}
 */
var minMaxGame = function(nums) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int minMaxGame(int* nums, int numsSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun minMaxGame(nums: IntArray): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Min Max Game
* 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 minMaxGame(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Min Max Game
* 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 minMaxGame(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Min Max Game
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 minMaxGame(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Min Max Game
* 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 minMaxGame = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Min Max Game
* 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 minMaxGame(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Min Max Game
* 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 MinMaxGame(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Min Max Game
* 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
*/

int minMaxGame(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Min Max Game
// 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 minMaxGame(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minMaxGame(nums: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Min Max Game
// 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 min_max_game(nums: Vec<i32>) -> i32 {

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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