Problem 2789: Largest Element in an Array after Merge Operations

Problem Information

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

Problem Description

You are given a

0-indexed

array

nums

consisting of positive integers.

You can do the following operation on the array

any

number of times:

Choose an index

i

such that

0 <= i < nums.length - 1

and

nums[i] <= nums[i + 1]

. Replace the element

nums[i + 1]

with

nums[i] + nums[i + 1]

and delete the element

nums[i]

from the array.

Return

the value of the

largest

element that you can possibly obtain in the final array.

Example 1:

Input:

nums = [2,3,7,9,3]

Output:

21

Explanation:

We can apply the following operations on the array: - Choose i = 0. The resulting array will be nums = [

5

,7,9,3]. - Choose i = 1. The resulting array will be nums = [5,

16

,3]. - Choose i = 0. The resulting array will be nums = [

21

,3]. The largest element in the final array is 21. It can be shown that we cannot obtain a larger element.

Example 2:

Input:

nums = [5,3,3]

Output:

11

Explanation:

We can do the following operations on the array: - Choose i = 1. The resulting array will be nums = [5,

6

]. - Choose i = 0. The resulting array will be nums = [

11

]. There is only one element in the final array, which is 11.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

6

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long maxArrayValue(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long MaxArrayValue(int[] nums) {

}
}
```

C:

```
long long maxArrayValue(int* nums, int numsSize) {

}
```

Go:

```
func maxArrayValue(nums []int) int64 {

}
```

Kotlin:

```
class Solution {
fun maxArrayValue(nums: IntArray): Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn max_array_value(nums: Vec<i32>) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-array-value nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Largest Element in an Array after Merge Operations
* Difficulty: Medium
* Tags: array, greedy
*
* 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:
long long maxArrayValue(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Largest Element in an Array after Merge Operations
* Difficulty: Medium
* Tags: array, greedy
*
* 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 long maxArrayValue(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Largest Element in an Array after Merge Operations
Difficulty: Medium
Tags: array, greedy

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

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Largest Element in an Array after Merge Operations
* Difficulty: Medium
* Tags: array, greedy
*
* 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 maxArrayValue = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Largest Element in an Array after Merge Operations
* Difficulty: Medium
* Tags: array, greedy
*
* 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 maxArrayValue(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Largest Element in an Array after Merge Operations
* Difficulty: Medium
* Tags: array, greedy
*
* 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 long MaxArrayValue(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Largest Element in an Array after Merge Operations
* Difficulty: Medium
* Tags: array, greedy
*
* 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
*/

long long maxArrayValue(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Largest Element in an Array after Merge Operations
// Difficulty: Medium
// Tags: array, greedy
//
// 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 maxArrayValue(nums []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maxArrayValue(nums: IntArray): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Largest Element in an Array after Merge Operations
// Difficulty: Medium
// Tags: array, greedy
//
// 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 max_array_value(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-array-value nums)
(-> (listof exact-integer?) exact-integer?)
)
```