

# Problem 2681: Power of Heroes

## Problem Information

**Difficulty:** Hard

**Acceptance Rate:** 0.00%

**Paid Only:** No

## Problem Description

You are given a

0-indexed

integer array

nums

representing the strength of some heroes. The

power

of a group of heroes is defined as follows:

Let

i

0

,

i

1

, ...,

i

k

be the indices of the heroes in a group. Then, the power of this group is

$\max(\text{nums}[i]$

0

$], \text{nums}[i$

1

$], \dots, \text{nums}[i$

k

)

2

$* \min(\text{nums}[i]$

0

$], \text{nums}[i$

1

$], \dots, \text{nums}[i$

k

)

.

Return

the sum of the

power

of all

non-empty

groups of heroes possible.

Since the sum could be very large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

nums = [2,1,4]

Output:

141

Explanation:

1

st

group: [2] has power = 2

2

\* 2 = 8. 2

nd

group: [1] has power = 1

2

\* 1 = 1. 3

rd

group: [4] has power = 4

2

\* 4 = 64. 4

th

group: [2,1] has power = 2

2

\* 1 = 4. 5

th

group: [2,4] has power = 4

2

\* 2 = 32. 6

th

group: [1,4] has power = 4

2

\* 1 = 16. 7

th

group: [2,1,4] has power = 4

2

\* 1 = 16. The sum of powers of all groups is  $8 + 1 + 64 + 4 + 32 + 16 + 16 = 141$ .

Example 2:

Input:

nums = [1,1,1]

Output:

7

Explanation:

A total of 7 groups are possible, and the power of each group will be 1. Therefore, the sum of the powers of all groups is 7.

Constraints:

$1 \leq \text{nums.length} \leq 10$

5

$1 \leq \text{nums}[i] \leq 10$

## Code Snippets

### C++:

```
class Solution {
public:
    int sumOfPower(vector<int>& nums) {
        ...
    }
};
```

### Java:

```
class Solution {
    public int sumOfPower(int[] nums) {
        ...
    }
}
```

### Python3:

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

### Python:

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

### JavaScript:

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

```
var sumOfPower = function(nums) {  
};
```

### TypeScript:

```
function sumOfPower(nums: number[]): number {  
};
```

### C#:

```
public class Solution {  
    public int SumOfPower(int[] nums) {  
          
    }  
}
```

### C:

```
int sumOfPower(int* nums, int numsSize) {  
}
```

### Go:

```
func sumOfPower(nums []int) int {  
}
```

### Kotlin:

```
class Solution {  
    fun sumOfPower(nums: IntArray): Int {  
          
    }  
}
```

### Swift:

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

```
}
```

```
}
```

### Rust:

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

### Ruby:

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

end
```

### PHP:

```
class Solution {

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

    }
}
```

### Dart:

```
class Solution {
    int sumOfPower(List<int> nums) {
        }
    }
```

### Scala:

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

### Elixir:

```
defmodule Solution do  
  @spec sum_of_power(list(integer)) :: integer  
  def sum_of_power(nums) do  
  
  end  
  end
```

### Erlang:

```
-spec sum_of_power(list(integer)) -> integer().  
sum_of_power(Nums) ->  
.
```

### Racket:

```
(define/contract (sum-of-power nums)  
  (-> (listof exact-integer?) exact-integer?)  
)
```

## Solutions

### C++ Solution:

```
/*  
 * Problem: Power of Heroes  
 * Difficulty: Hard  
 * Tags: array, dp, math, sort  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(n) or O(n * m) for DP table  
 */
```

```
class Solution {  
public:  
    int sumOfPower(vector<int>& nums) {  
  
    }  
};
```

### Java Solution:

```
/**  
 * Problem: Power of Heroes  
 * Difficulty: Hard  
 * Tags: array, dp, math, sort  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(n) or O(n * m) for DP table  
 */  
  
class Solution {  
public int sumOfPower(int[] nums) {  
  
}  
}
```

### Python3 Solution:

```
"""  
Problem: Power of Heroes  
Difficulty: Hard  
Tags: array, dp, math, sort  
  
Approach: Use two pointers or sliding window technique  
Time Complexity: O(n) or O(n log n)  
Space Complexity: O(n) or O(n * m) for DP table  
"""  
  
class Solution:  
    def sumOfPower(self, nums: List[int]) -> int:  
        # TODO: Implement optimized solution  
        pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
 * Problem: Power of Heroes
 * Difficulty: Hard
 * Tags: array, dp, math, sort
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

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

};
```

### TypeScript Solution:

```
/**
 * Problem: Power of Heroes
 * Difficulty: Hard
 * Tags: array, dp, math, sort
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

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

```
};
```

### C# Solution:

```
/*
 * Problem: Power of Heroes
 * Difficulty: Hard
 * Tags: array, dp, math, sort
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

public class Solution {
    public int SumOfPower(int[] nums) {

    }
}
```

### C Solution:

```
/*
 * Problem: Power of Heroes
 * Difficulty: Hard
 * Tags: array, dp, math, sort
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

int sumOfPower(int* nums, int numsSize) {

}
```

### Go Solution:

```
// Problem: Power of Heroes
// Difficulty: Hard
```

```

// Tags: array, dp, math, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func sumOfPower(nums []int) int {

}

```

### Kotlin Solution:

```

class Solution {
    fun sumOfPower(nums: IntArray): Int {
        return 0
    }
}

```

### Swift Solution:

```

class Solution {
    func sumOfPower(_ nums: [Int]) -> Int {
        return 0
    }
}

```

### Rust Solution:

```

// Problem: Power of Heroes
// Difficulty: Hard
// Tags: array, dp, math, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
    pub fn sum_of_power(nums: Vec<i32>) -> i32 {
        return 0
    }
}

```

### Ruby Solution:

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

end
```

### PHP Solution:

```
class Solution {

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

    }
}
```

### Dart Solution:

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

}
```

### Scala Solution:

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

}
```

### Elixir Solution:

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

```
end  
end
```

### Erlang Solution:

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

### Racket Solution:

```
(define/contract (sum-of-power nums)  
(-> (listof exact-integer?) exact-integer?)  
)
```