

Problem 2897: Apply Operations on Array to Maximize Sum of Squares

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a

0-indexed

integer array

nums

and a

positive

integer

k

.

You can do the following operation on the array

any

number of times:

Choose any two distinct indices

i

and

j

and

simultaneously

update the values of

$\text{nums}[i]$

to

$(\text{nums}[i] \text{ AND } \text{nums}[j])$

and

$\text{nums}[j]$

to

$(\text{nums}[i] \text{ OR } \text{nums}[j])$

. Here,

OR

denotes the bitwise

OR

operation, and

AND

denotes the bitwise

AND

operation.

You have to choose

k

elements from the final array and calculate the sum of their

squares

Return

the

maximum

sum of squares you can achieve

Since the answer can be very large, return it

modulo

10

9

+ 7

Example 1:

Input:

nums = [2,6,5,8], k = 2

Output:

261

Explanation:

We can do the following operations on the array: - Choose $i = 0$ and $j = 3$, then change $\text{nums}[0]$ to $(2 \text{ AND } 8) = 0$ and $\text{nums}[3]$ to $(2 \text{ OR } 8) = 10$. The resulting array is $\text{nums} = [0,6,5,10]$. - Choose $i = 2$ and $j = 3$, then change $\text{nums}[2]$ to $(5 \text{ AND } 10) = 0$ and $\text{nums}[3]$ to $(5 \text{ OR } 10) = 15$. The resulting array is $\text{nums} = [0,6,0,15]$. We can choose the elements 15 and 6 from the final array. The sum of squares is 15

2

+ 6

2

= 261. It can be shown that this is the maximum value we can get.

Example 2:

Input:

nums = [4,5,4,7], k = 3

Output:

90

Explanation:

We do not need to apply any operations. We can choose the elements 7, 5, and 4 with a sum of squares: 7

2

+ 5

2

+ 4

2

= 90. It can be shown that this is the maximum value we can get.

Constraints:

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

5

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

9

Code Snippets

C++:

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

Java:

```
class Solution {
public int maxSum(List<Integer> nums, int k) {
    }
```

```
}
```

Python3:

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

Python:

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

JavaScript:

```
/**  
 * @param {number[]} nums  
 * @param {number} k  
 * @return {number}  
 */  
var maxSum = function(nums, k) {  
  
};
```

TypeScript:

```
function maxSum(nums: number[], k: number): number {  
  
};
```

C#:

```
public class Solution {  
    public int MaxSum(IList<int> nums, int k) {  
  
    }  
}
```

C:

```
int maxSum(int* nums, int numsSize, int k) {  
  
}
```

Go:

```
func maxSum(nums []int, k int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun maxSum(nums: List<Int>, k: Int): Int {  
  
    }  
}
```

Swift:

```
class Solution {  
    func maxSum(_ nums: [Int], _ k: Int) -> Int {  
  
    }  
}
```

Rust:

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

Ruby:

```
# @param {Integer[]} nums  
# @param {Integer} k  
# @return {Integer}  
def max_sum(nums, k)
```

```
end
```

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums  
     * @param Integer $k  
     * @return Integer  
     */  
    function maxSum($nums, $k) {  
  
    }  
}
```

Dart:

```
class Solution {  
int maxSum(List<int> nums, int k) {  
  
}  
}
```

Scala:

```
object Solution {  
def maxSum(nums: List[Int], k: Int): Int = {  
  
}  
}
```

Elixir:

```
defmodule Solution do  
@spec max_sum(nums :: [integer], k :: integer) :: integer  
def max_sum(nums, k) do  
  
end  
end
```

Erlang:

```
-spec max_sum(Nums :: [integer()]), K :: integer() -> integer().  
max_sum(Nums, K) ->  
. .
```

Racket:

```
(define/contract (max-sum nums k)  
(-> (listof exact-integer?) exact-integer? exact-integer?)  
)
```

Solutions

C++ Solution:

```
/*  
 * Problem: Apply Operations on Array to Maximize Sum of Squares  
 * Difficulty: Hard  
 * Tags: array, greedy, hash  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(n) for hash map  
 */  
  
class Solution {  
public:  
    int maxSum(vector<int>& nums, int k) {  
  
    }  
};
```

Java Solution:

```
/**  
 * Problem: Apply Operations on Array to Maximize Sum of Squares  
 * Difficulty: Hard  
 * Tags: array, greedy, hash  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(n) for hash map
```

```
*/\n\n\nclass Solution {\n    public int maxSum(List<Integer> nums, int k) {\n\n        }\n    }\n}
```

Python3 Solution:

```
'''\n\nProblem: Apply Operations on Array to Maximize Sum of Squares\nDifficulty: Hard\nTags: array, greedy, hash\n\nApproach: Use two pointers or sliding window technique\nTime Complexity: O(n) or O(n log n)\nSpace Complexity: O(n) for hash map\n'''
```

```
class Solution:\n    def maxSum(self, nums: List[int], k: int) -> int:\n        # TODO: Implement optimized solution\n        pass
```

Python Solution:

```
class Solution(object):\n    def maxSum(self, nums, k):\n\n        '''\n        :type nums: List[int]\n        :type k: int\n        :rtype: int\n        '''
```

JavaScript Solution:

```
/**\n * Problem: Apply Operations on Array to Maximize Sum of Squares\n * Difficulty: Hard\n * Tags: array, greedy, hash
```

```

/*
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

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

};

```

TypeScript Solution:

```

/**
 * Problem: Apply Operations on Array to Maximize Sum of Squares
 * Difficulty: Hard
 * Tags: array, greedy, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

function maxSum(nums: number[], k: number): number {

};

```

C# Solution:

```

/*
 * Problem: Apply Operations on Array to Maximize Sum of Squares
 * Difficulty: Hard
 * Tags: array, greedy, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map

```

```
*/\n\npublic class Solution {\n    public int MaxSum(IList<int> nums, int k) {\n\n        }\n    }\n}
```

C Solution:

```
/*\n * Problem: Apply Operations on Array to Maximize Sum of Squares\n * Difficulty: Hard\n * Tags: array, greedy, hash\n *\n * Approach: Use two pointers or sliding window technique\n * Time Complexity: O(n) or O(n log n)\n * Space Complexity: O(n) for hash map\n */\n\nint maxSum(int* nums, int numsSize, int k) {\n\n}
```

Go Solution:

```
// Problem: Apply Operations on Array to Maximize Sum of Squares\n// Difficulty: Hard\n// Tags: array, greedy, hash\n//\n// Approach: Use two pointers or sliding window technique\n// Time Complexity: O(n) or O(n log n)\n// Space Complexity: O(n) for hash map\n\nfunc maxSum(nums []int, k int) int {\n\n}
```

Kotlin Solution:

```
class Solution {  
    fun maxSum(nums: List<Int>, k: Int): Int {  
        }  
        }  
}
```

Swift Solution:

```
class Solution {  
    func maxSum(_ nums: [Int], _ k: Int) -> Int {  
        }  
        }  
}
```

Rust Solution:

```
// Problem: Apply Operations on Array to Maximize Sum of Squares  
// Difficulty: Hard  
// Tags: array, greedy, hash  
//  
// Approach: Use two pointers or sliding window technique  
// Time Complexity: O(n) or O(n log n)  
// Space Complexity: O(n) for hash map  
  
impl Solution {  
    pub fn max_sum(nums: Vec<i32>, k: i32) -> i32 {  
        }  
        }  
}
```

Ruby Solution:

```
# @param {Integer[]} nums  
# @param {Integer} k  
# @return {Integer}  
def max_sum(nums, k)  
  
end
```

PHP Solution:

```
class Solution {

    /**
     * @param Integer[] $nums
     * @param Integer $k
     * @return Integer
     */
    function maxSum($nums, $k) {

    }
}
```

Dart Solution:

```
class Solution {
    int maxSum(List<int> nums, int k) {

    }
}
```

Scala Solution:

```
object Solution {
    def maxSum(nums: List[Int], k: Int): Int = {

    }
}
```

Elixir Solution:

```
defmodule Solution do
    @spec max_sum(nums :: [integer], k :: integer) :: integer
    def max_sum(nums, k) do
        end
    end
```

Erlang Solution:

```
-spec max_sum(Nums :: [integer()], K :: integer()) -> integer().
max_sum(Nums, K) ->
    .
```

Racket Solution:

```
(define/contract (max-sum nums k)
  (-> (listof exact-integer?) exact-integer? exact-integer?))
)
```