Problem 2917: Find the K-or of an Array

Problem Information

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

Problem Description

You are given an integer array

nums

, and an integer

k

. Let's introduce

K-or

operation by extending the standard bitwise OR. In K-or, a bit position in the result is set to

1

if at least

k

numbers in

nums

have a

1

in that position.

Return

the K-or of

nums

.

Example 1:

Input:

nums = [7,12,9,8,9,15], k = 4

Output:

9

Explanation:

Represent numbers in binary:

Number

Bit 3

Bit 2

Bit 1

Bit 0

7

0

1

1

1

12

1

1

0

0

9

1

0

0

1

8

1

0

0

0

9

1

0

0

1

15

1

1

1

1

Result = 9

1

0

0

1

Bit 0 is set in 7, 9, 9, and 15. Bit 3 is set in 12, 9, 8, 9, and 15.

Only bits 0 and 3 qualify. The result is

(1001)

2

= 9

.

Example 2:

Input:

nums = [2,12,1,11,4,5], k = 6

Output:

0

Explanation:

No bit appears as 1 in all six array numbers, as required for K-or with

k = 6

. Thus, the result is 0.

Example 3:

Input:

nums = [10,8,5,9,11,6,8], k = 1

Output:

15

Explanation:

Since

k == 1

, the 1-or of the array is equal to the bitwise OR of all its elements. Hence, the answer is

10 OR 8 OR 5 OR 9 OR 11 OR 6 OR 8 = 15

.

Constraints:

1 <= nums.length <= 50

0 <= nums[i] < 2

31

1 <= k <= nums.length

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int findKOr(int[] nums, int k) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def findKOr(self, nums, k):
```

```
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function findKOr(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int FindKOr(int[] nums, int k) {

}
}
```

C:

```
int findKOr(int* nums, int numsSize, int k) {

}
```

Go:

```
func findKOr(nums []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun findKOr(nums: IntArray, k: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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



}
}
```

Dart:

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

}
}
```

Scala:

```
object Solution {
def findKOr(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (find-k-or nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the K-or of an Array
* 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 findKOr(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Find the K-or of an Array
* 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 findKOr(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the K-or of an Array
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the K-or of an Array
* 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
* @param {number} k
* @return {number}
*/
var findKOr = function(nums, k) {
};
```

TypeScript Solution:

```
/**
* Problem: Find the K-or of an Array
* 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 findKOr(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Find the K-or of an Array
* 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 FindKOr(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Find the K-or of an Array
* 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 findKOr(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Find the K-or of an Array
// 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 findKOr(nums []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findKOr(nums: IntArray, k: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find the K-or of an Array
// 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 find_k_or(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

```
object Solution {
def findKOr(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-k-or nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```