Problem 2680: Maximum OR

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

of length

n

and an integer

k

. In an operation, you can choose an element and multiply it by

2

.

Return

the maximum possible value of

nums[0] | nums[1] | ... | nums[n - 1]

that can be obtained after applying the operation on nums at most

k

times

.

Note that

a | b

denotes the

bitwise or

between two integers

a

and

b

.

Example 1:

Input:

nums = [12,9], k = 1

Output:

30

Explanation:

If we apply the operation to index 1, our new array nums will be equal to [12,18]. Thus, we return the bitwise or of 12 and 18, which is 30.

Example 2:

Input:

nums = [8,1,2], k = 2

Output:

35

Explanation:

If we apply the operation twice on index 0, we yield a new array of [32,1,2]. Thus, we return 32|1|2 = 35.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

1 <= k <= 15

Code Snippets

C++:

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

```
}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
long long maximumOr(int* nums, int numsSize, int k) {

}
```

Go:

```
func maximumOr(nums []int, k int) int64 {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
def maximum_or(nums, k) do

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum OR
* 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 maximumOr(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum OR
* 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 maximumOr(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum OR
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 maximumOr(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum OR
* 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
* @param {number} k
* @return {number}
*/
var maximumOr = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum OR
* 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 maximumOr(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum OR
* 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 MaximumOr(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum OR
* 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 maximumOr(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum OR
// 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 maximumOr(nums []int, k int) int64 {
```

```
}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum OR
// 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 maximum_or(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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