# Problem 1829: Maximum XOR for Each Query

## Problem Information

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

## Problem Description

You are given a

sorted

array

nums

of

n

non-negative integers and an integer

maximumBit

. You want to perform the following query

n

times

:

Find a non-negative integer

k < 2

maximumBit

such that

nums[0] XOR nums[1] XOR ... XOR nums[nums.length-1] XOR k

is

maximized

.

k

is the answer to the

i

th

query.

Remove the

last

element from the current array

nums

.

Return

an array

answer

, where

answer[i]

is the answer to the

i

th

query

.

Example 1:

Input:

nums = [0,1,1,3], maximumBit = 2

Output:

[0,3,2,3]

Explanation

: The queries are answered as follows: 1

st

query: nums = [0,1,1,3], k = 0 since 0 XOR 1 XOR 1 XOR 3 XOR 0 = 3. 2

nd

query: nums = [0,1,1], k = 3 since 0 XOR 1 XOR 1 XOR 3 = 3. 3

rd

query: nums = [0,1], k = 2 since 0 XOR 1 XOR 2 = 3. 4

th

query: nums = [0], k = 3 since 0 XOR 3 = 3.

Example 2:

Input:

nums = [2,3,4,7], maximumBit = 3

Output:

[5,2,6,5]

Explanation

: The queries are answered as follows: 1

st

query: nums = [2,3,4,7], k = 5 since 2 XOR 3 XOR 4 XOR 7 XOR 5 = 7. 2

nd

query: nums = [2,3,4], k = 2 since 2 XOR 3 XOR 4 XOR 2 = 7. 3

rd

query: nums = [2,3], k = 6 since 2 XOR 3 XOR 6 = 7. 4

th

query: nums = [2], k = 5 since 2 XOR 5 = 7.

Example 3:

Input:

nums = [0,1,2,2,5,7], maximumBit = 3

Output:

[4,3,6,4,6,7]

Constraints:

nums.length == n

1 <= n <= 10

5

1 <= maximumBit <= 20

0 <= nums[i] < 2

maximumBit

nums

is sorted in

ascending

order.

## Code Snippets

**C++:**

```
class Solution {
public:
vector<int> getMaximumXor(vector<int>& nums, int maximumBit) {

}
};
```

**Java:**

```
class Solution {
public int[] getMaximumXor(int[] nums, int maximumBit) {

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

```
function getMaximumXor(nums: number[], maximumBit: number): number[] {

};
```

**C#:**

```
public class Solution {
public int[] GetMaximumXor(int[] nums, int maximumBit) {

}
}
```

**C:**

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* getMaximumXor(int* nums, int numsSize, int maximumBit, int* returnSize)
{

}
```

**Go:**

```
func getMaximumXor(nums []int, maximumBit int) []int {

}
```

**Kotlin:**

```
class Solution {
fun getMaximumXor(nums: IntArray, maximumBit: Int): IntArray {

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

```
}
}
```

**Ruby:**

```
# @param {Integer[]} nums
# @param {Integer} maximum_bit
# @return {Integer[]}
def get_maximum_xor(nums, maximum_bit)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $maximumBit
* @return Integer[]
*/
function getMaximumXor($nums, $maximumBit) {

}
}
```

**Dart:**

```
class Solution {
List<int> getMaximumXor(List<int> nums, int maximumBit) {

}
}
```

**Scala:**

```
object Solution {
def getMaximumXor(nums: Array[Int], maximumBit: Int): Array[Int] = {

}
}
```

### Elixir:

```
defmodule Solution do
@spec get_maximum_xor(nums :: [integer], maximum_bit :: integer) :: [integer]
def get_maximum_xor(nums, maximum_bit) do

end
end
```

### Erlang:

```
-spec get_maximum_xor(Nums :: [integer()], MaximumBit :: integer()) ->
[integer()].
get_maximum_xor(Nums, MaximumBit) ->
.
```

### Racket:

```
(define/contract (get-maximum-xor nums maximumBit)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Maximum XOR for Each Query
* Difficulty: Medium
* Tags: array, sort
*
* 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:
vector<int> getMaximumXor(vector<int>& nums, int maximumBit) {

}
};
```

### Java Solution:

```
/**
* Problem: Maximum XOR for Each Query
* Difficulty: Medium
* Tags: array, sort
*
* 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[] getMaximumXor(int[] nums, int maximumBit) {

}
}
```

### Python3 Solution:

```
"""
Problem: Maximum XOR for Each Query
Difficulty: Medium
Tags: array, sort

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

### Python Solution:

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

```
:rtype: List[int]
"""
```

### JavaScript Solution:

```
/**
* Problem: Maximum XOR for Each Query
* Difficulty: Medium
* Tags: array, sort
*
* 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} maximumBit
* @return {number[]}
*/
var getMaximumXor = function(nums, maximumBit) {

};
```

### TypeScript Solution:

```
/**
* Problem: Maximum XOR for Each Query
* Difficulty: Medium
* Tags: array, sort
*
* 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 getMaximumXor(nums: number[], maximumBit: number): number[] {

};
```

### C# Solution:

```
/*
* Problem: Maximum XOR for Each Query
* Difficulty: Medium
* Tags: array, sort
*
* 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[] GetMaximumXor(int[] nums, int maximumBit) {

}
}
```

### C Solution:

```
/*
* Problem: Maximum XOR for Each Query
* Difficulty: Medium
* Tags: array, sort
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* getMaximumXor(int* nums, int numsSize, int maximumBit, int* returnSize)
{

}
```

### Go Solution:

```
// Problem: Maximum XOR for Each Query
// Difficulty: Medium
// Tags: array, sort
//
```

```
// 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 getMaximumXor(nums []int, maximumBit int) []int {

}
```

**Kotlin Solution:**

```
class Solution {
fun getMaximumXor(nums: IntArray, maximumBit: Int): IntArray {

}
}
```

**Swift Solution:**

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

}
}
```

**Rust Solution:**

```
// Problem: Maximum XOR for Each Query
// Difficulty: Medium
// Tags: array, sort
//
// 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 get_maximum_xor(nums: Vec<i32>, maximum_bit: i32) -> Vec<i32> {

}
}
```

**Ruby Solution:**

```
# @param {Integer[]} nums
# @param {Integer} maximum_bit
# @return {Integer[]}
def get_maximum_xor(nums, maximum_bit)

end
```

**PHP Solution:**

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $maximumBit
* @return Integer[]
*/
function getMaximumXor($nums, $maximumBit) {

}
}
```

**Dart Solution:**

```
class Solution {
List<int> getMaximumXor(List<int> nums, int maximumBit) {

}
}
```

**Scala Solution:**

```
object Solution {
def getMaximumXor(nums: Array[Int], maximumBit: Int): Array[Int] = {

}
}
```

**Elixir Solution:**

```
defmodule Solution do
@spec get_maximum_xor(nums :: [integer], maximum_bit :: integer) :: [integer]
def get_maximum_xor(nums, maximum_bit) do
```

```
end
end
```

### Erlang Solution:

```
-spec get_maximum_xor(Nums :: [integer()], MaximumBit :: integer()) ->
[integer()].
get_maximum_xor(Nums, MaximumBit) ->
.
```

### Racket Solution:

```
(define/contract (get-maximum-xor nums maximumBit)
(-> (listof exact-integer?) exact-integer? (listof exact-integer?))
)
```