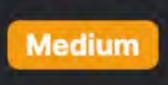
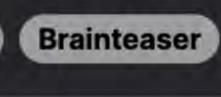
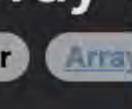
2419. Longest Subarray With Maximum Bitwise AND

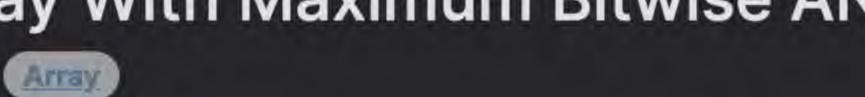


Bit Manipulation









Leetcode Link

Problem Description In this LeetCode problem, you are presented with an integer array named nums, which contains a certain number of integers. Your

objective is to find the maximum possible bitwise AND value from all possible non-empty subarrays of this array. A bitwise AND operates on two numbers bit by bit and only returns 1 for each bit position if both corresponding bits in the two numbers are also 1; otherwise, it returns 0 for that position. Once you've determined this maximum bitwise AND value (let's call it k), you need to consider only the subarrays that produce this

maximum value when performing a bitwise AND on all their elements. Among these subarrays, your goal is to find the length of the longest one. To sum up, you must:

 Calculate the maximum bitwise AND value for all possible subarrays. 2. Identify which subarrays yield this maximum value.

- 3. Determine the maximum length among those subarrays.
- A subarray is defined as a contiguous sequence of elements within the original array. So, the challenge essentially revolves around
- yielding the maximum bitwise AND value.

understanding bit manipulation and being able to efficiently navigate through the array to find the longest contiguous sequence

Intuition The intuition behind the provided solution lies in understanding the properties of the bitwise AND operation. One of the key insights

is that if you perform a bitwise AND on any number with another number that is smaller, the result will always be less than or equal to

the larger number. It means that the maximum bitwise AND value for any subarray in nums can only be obtained if every element in that subarray is at least as large as the maximum value in the entire array.

With this in mind, finding the solution does not require checking every possible subarray. Instead, you can follow a more straightforward approach by first finding the maximum value (mx) in nums. Then, you simply need to look for the longest contiguous sequence of mx within the array. This sequence will guarantee the maximum bitwise AND result because the AND of any number with itself is the number.

Here's how the solution approach is derived: 1. Find the maximum value mx in nums. 2. Initialize a counter (cnt) to track the length of the current sequence of mx elements, and another variable (ans) to keep track of the length of the longest sequence found so far.

3. Iterate through the elements in nums, incrementing cnt if the current element is equal to mx. If an element is not equal to mx, reset

cnt to 0 because the sequence is broken. 4. After each step, update ans with the greater of its current value or cnt, to ensure ans always represents the length of the longest

sequence encountered.

solution to the problem.

1 mx = max(nums)

1 ans = cnt = 0

- 5. At the end of the iteration, ans holds the length of the longest subarray that gives the maximum bitwise AND value which is the
- Solution Approach
- The implementation of the solution uses a straightforward approach without the need for complex algorithms or additional data structures.

1. First, the maximum value (mx) in the array nums is identified using the max() function. This step is crucial because any subarray

whose bitwise AND is to be maximized must contain mx according to the properties of the bitwise AND operation.

2. Two variables are initialized: ans for storing the maximum subarray length found so far, and cnt for counting the length of the

Let's consider the following array nums as an example to illustrate the solution approach:

current sequence of maximum elements when iterating through nums.

Here is the step-by-step execution of the algorithm according to the provided Python code:

3. The function then iterates through every element (v) in nums. For each element, it checks if it equals the maximum value mx. 1 for v in nums:

element. The ans variable is updated with the larger of its current value or cnt to ensure that it always holds the length of the

4. If the current element is equal to mx, then cnt is incremented as we are currently in a subarray consisting of the maximum

5. If the current element is not equal to mx, the cnt is reset to 0 because the sequence of mx is broken, and we need to start

cnt = 0

1 if v == mx:

1 else:

longest subarray found so far.

ans = max(ans, cnt)

counting anew for the next potential sequence.

maximum possible value k. This value of ans is returned as the answer. 1 return ans

The code does not make use of any specific patterns or advanced data structures, relying instead on a simple linear scan of the

input array and basic variables for counting. This type of pattern could be considered a two-pointer approach, where one pointer (or

in this case a counter) keeps track of the current subarray's length and another (implicit pointer) moves through the array elements

via the for-loop. The solution efficiently arrives at the answer in O(n) time, where n is the length of the nums array, since it requires

6. At the end of the iteration, the value of ans will be the length of the longest subarray where the bitwise AND is equal to the

```
only a single pass through the array.
```

Example Walkthrough

1 nums = [2, 2, 1, 2, 2]

Using the solution approach, perform the following steps: 1. Identify the maximum value (mx) in nums: 1 mx = max(nums) # mx = 22. Initialize the required variables to store the length of the current sequence (cnt) and the maximum subarray length found (ans):

3. Iterate through each element (v) in nums and compare it with mx, incrementing cnt if it's the same or resetting cnt if it's different: 1 for v in nums: # Loop starts, iterating over the elements in nums.

1 else:

For the first element, v = 2:

1 if v == mx: # True, as 2 == 2

1 if v == mx: # True, as 2 == 2

For the third element, v = 1:

 \circ For the fourth element, v = 2:

1 if v == mx: # True, as 2 == 2

cnt += 1 # cnt becomes 1 again

cnt += 1 # cnt becomes 1

cnt += 1 # cnt becomes 2

ans = max(ans, cnt) # ans becomes max(0, 1) which is 1

ans = max(ans, cnt) # ans becomes max(1, 2) which is 2

ans = max(ans, cnt) # ans remains 2, as max(2, 1) is 2

cnt = 0 # cnt is reset since 1 is not equal to mx (2)

1 ans = cnt = 0

For the second element, v = 2:

```
 For the fifth element, v = 2:

         1 if v == mx: # True, as 2 == 2
               cnt += 1  # cnt becomes 2 (as we had 1 from the previous step)
               ans = max(ans, cnt) # ans becomes max(2, 2) which is 2
 4. After completing the iteration, the ans variable contains the length of the longest subarray where the bitwise AND is equal to the
   maximum possible value k (which is 2 in this case), and ans is 2:
    1 return ans # Returns 2 as the answer.
In this example, the longest continuous subarray with elements that have the maximum value mx (2) consists of 2 elements, so ans is
2. This is the length of the longest subarray that when bitwise AND-ed together would give the maximum possible value.
The algorithm successfully finds this subarray length in one pass, which makes it a very efficient solution.
Python Solution
   from typing import List
   class Solution:
       def longestSubarray(self, nums: List[int]) -> int:
           # Find the maximum value in the array nums.
           max_value = max(nums)
           # Initialize the longest length and the current length counter to zero.
           longest_length = current_length = 0
           # Iterate through each element in the list.
           for number in nums:
               # If the current element is equal to the maximum value...
               if number == max_value:
                   # Increment the current length counter.
                   current_length += 1
                   # Update the longest length if the current length is greater.
```

longest_length = max(longest_length, current_length)

int maxNum = 0; // variable to store the maximum value in the array

int maxLength = 0; // variable to store the length of the longest subarray

// Iterate through the array to find the length of the longest subarray

// Function to find the length of the longest subarray consisting of the maximum element

int currentLength = 0; // variable to track the length of the current subarray

// If the current element is the max, increment the current length

// Iterate through the array to find the maximum value

// where all elements are equal to the maximum value

// Return the length of the longest subarray

If the current element is not equal to the max value, reset current length to 0.

After iterating through the list, return the longest length of the subarray with max values.

currentLength++; 18 // Update the maxLength if the current subarray is longer 19 maxLength = Math.max(maxLength, currentLength); 20 } else { // Reset the current length if the current element is not max 21 currentLength = 0;

C++ Solution

Java Solution

1 class Solution {

9

10

11

12

13

14

15

16

17

18

20

21

22

23

24

9

11

12

13

14

15

16

23

24

25

26

27

28

30

else:

return longest_length

for (int num : nums) {

for (int num : nums) {

return maxLength;

if (num == maxNum) {

current_length = 0

public int longestSubarray(int[] nums) {

maxNum = Math.max(maxNum, num);

```
1 #include <vector>
   #include <algorithm>
   class Solution {
  public:
       // Method to find the length of the longest subarray consisting of the maximum element.
 6
       int longestSubarray(std::vector<int>& nums) {
           // Get the maximum value in the array.
 8
           int maxValue = *std::max_element(nums.begin(), nums.end());
           // Initialize answer (longest subarray length) and counter for current subarray length.
10
           int longestSubarrayLength = 0, currentSubarrayLength = 0;
11
12
13
           // Iterate over each element in the array.
           for (int value : nums) {
14
               // Check if the current element equals the maximum value.
16
               if (value == maxValue) {
17
                   // Increment the current subarray length as it is part of a subarray containing max elements.
18
                   ++currentSubarrayLength;
                   // Update the answer with the maximum subarray length found so far.
19
                   longestSubarrayLength = std::max(longestSubarrayLength, currentSubarrayLength);
20
               } else {
                   // Reset current subarray length if the current element is not the maximum value.
23
                   currentSubarrayLength = 0;
24
25
26
           // Return the length of the longest subarray found.
           return longestSubarrayLength;
28
29 };
30
Typescript Solution
```

// Importing the `max` function from Lodash for finding maximum element in array import max from 'lodash/max';

1 npm install lodash

```
function longestSubarray(nums: number[]): number {
       // Get the maximum value in the array using Lodash max function
       const maxValue: number = max(nums);
       // Initialize longestSubarrayLength for keeping track of the longest subarray length and currentSubarrayLength for the current se
       let longestSubarrayLength: number = 0;
       let currentSubarrayLength: number = 0;
10
11
12
       // Iterate over each element in the array
       nums.forEach((value: number) => {
           // If the current element equals the maximum value, it's part of a max-element subarray
           if (value === maxValue) {
15
               // Increment the current subarray length counter
16
               currentSubarrayLength++;
               // Update the longest subarray length if the current one is longer
18
               longestSubarrayLength = Math.max(longestSubarrayLength, currentSubarrayLength);
19
20
21
               // If the current element is not the max value, reset current subarray length counter
22
               currentSubarrayLength = 0;
23
24
       });
25
26
       // Return the length of the longest subarray found
27
       return longestSubarrayLength;
28 }
Please note that the method names haven't been changed as per the instruction. The given logic and functionality are equivalent to
the original C++ code, using TypeScript syntax and naming conventions. In TypeScript, there's no need for explicit imports like in
C++, as one can use the corresponding functions directly or, as in this case, I used lodash for getting the maximum element in an
array for illustrative purposes.
```

Time and Space Complexity

Due to TypeScript's reliance on npm packages, you'd need to install lodash to use it:

The time complexity of the given code snippet is O(n) where n is the length of the input list nums. This is because the code iterates through the list once with a single for-loop, performing constant-time operations within the loop.

The space complexity of the code is 0(1) as it uses a fixed amount of additional space (variables mx, ans, cnt, and v) that does not depend on the input size n.