





Longest Subarray with Ones after Replacement (hard)

We'll cover the following

- Problem Statement
- Try it yourself
- Solution
- Code
 - Time Complexity
 - Space Complexity

Problem Statement

Given an array containing 0s and 1s, if you are allowed to **replace no more than 'k' 0s with 1s**, find the length of the **longest contiguous subarray having all 1s**.

Example 1:

Input: Array=[0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1], k=2

Output: 6

Explanation: Replace the '0' at index 5 and 8 to have the longes

t contiguous subarray of 1s having length 6.

Example 2:

Input: Array=[0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1], k=3 Output: 9

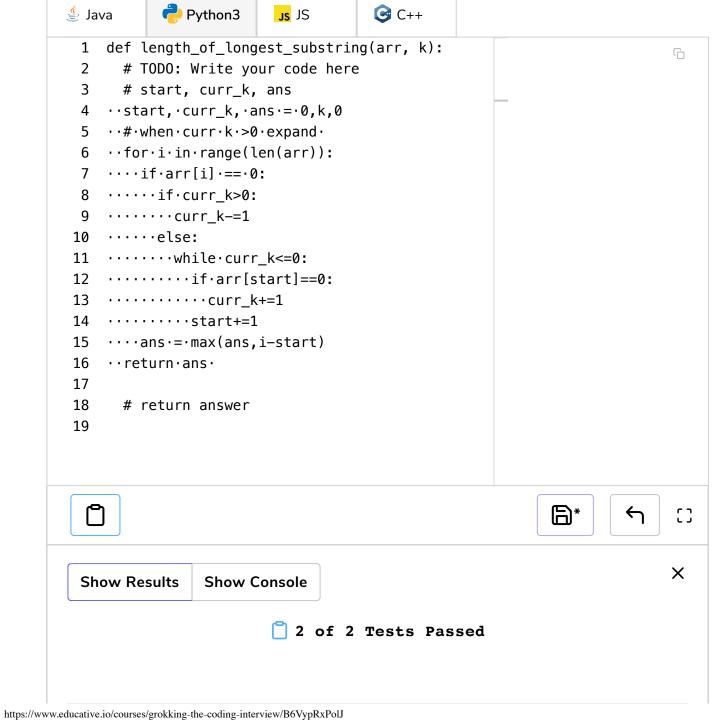




Explanation: Replace the '0' at index 6, 9, and 10 to have the lon gest contiguous subarray of 1s having length 9.

Try it yourself

Try solving this question here:



Result	Input	Expected Output	Actual Output	Reason
✓	<pre>length_of_longest_substring([0,</pre>	6	6	Succeeded
~	<pre>length_of_longest_substring([0,</pre>	9	9	Succeeded
	0.15s			

Solution

This problem follows the **Sliding Window** pattern and is quite similar to Longest Substring with same Letters after Replacement (https://www.educative.io/collection/page/5668639101419520/5671464854355 968/6497958910492672/). The only difference is that, in the problem, we only have two characters (1s and 0s) in the input arrays.

Following a similar approach, we'll iterate through the array to add one number at a time in the window. We'll also keep track of the maximum number of repeating 1s in the current window (let's call it maxOnesCount). So at any time, we know that we can have a window with 1s repeating maxOnesCount time, so we should try to replace the remaining 0s. If we have more than 'k' remaining 0s, we should shrink the window as we are not allowed to replace more than 'k' 0s.

Code

Here is how our algorithm will look like:

```
if arr[window_end] == 1:
 6
 7
                                                                         ₩
          max_ones_count += 1
 8
 9
        # Current window size is from window_start t
        # repeating 'max_ones_count' times, this mea
10
        # and the remaining are 0s which should repl
11
        # now, if the remaining 0s are more than 'k'
12
        # are not allowed to replace more than 'k' (
13
        if (window_end - window_start + 1 - max_ones
14
15
          if arr[window_start] == 1:
16
            max_ones_count -= 1
17
          window_start += 1
18
        max_length = max(max_length, window_end - wi
19
20
      return max_length
21
22
23
    def main():
24
      print(length_of_longest_substring([0, 1, 1, 0,
      print(length_of_longest_substring(
25
26
        [0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1], 3))
27
28
                                                            \triangleright
```

Time Complexity

The above algorithm's time complexity will be O(N), where 'N' is the count of numbers in the input array.

Space Complexity

The algorithm runs in constant space O(1).



? Ask a Question





