

# Longest Subarray with Ones after Replacement (hard)

## 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 longest 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 longest contiguous subarray of 1s having length 9.


## Try it yourself #

Try solving this question here:

 Java




 Python3

 JS

 C++

```
1 using namespace std;
2
3 #include <iostream>
4 #include <vector>
5
6 class ReplacingOnes {
7 public:
8     static int findLength(const vector<int>& arr, int k) {
9         int maxLength = 0;
10        // TODO: Write your code here
11        return maxLength;
12    }
13 };

```



Show Results

Show Console

×

0 of 2 Tests Passed

Result	Input	Expected Output	Actual Output	Reason
✗	findLength([0, 1, 1, 0, 0, 0, 1, 1, 0, 1 ...	6	-1	Incorrect Output
✗	findLength([0, 1, 0, 0, 1, 1, 0, 1, 1, 0 ...	9	-1	Incorrect Output

3.860s

## Solution #

This problem follows the **Sliding Window** pattern and is quite similar to [Longest Substring with same Letters after Replacement](#). 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 which has 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:

--	--	--	--

JavaPython3C++JS

```
1 using namespace std;
2
3 #include <iostream>
4 #include <vector>
5
6 class ReplacingOnes {
7 public:
8     static int findLength(const vector<int>& arr, int k) {
9         int windowStart = 0, maxLength = 0, maxOnesCount = 0;
10        // try to extend the range [windowStart, windowEnd]
11        for (int windowEnd = 0; windowEnd < arr.size(); windowEnd++) {
12            if (arr[windowEnd] == 1) {
13                maxOnesCount++;
14            }
15
16            // current window size is from windowStart to windowEnd, overall we have a maximum of 1s
17            // repeating a maximum of 'maxOnesCount' times, this means that we can have a window with
18            // 'maxOnesCount' 1s and the remaining are 0s which should replace with 1s.
19            // now, if the remaining 0s are more than 'k', it is the time to shrink the window as we
20            // are not allowed to replace more than 'k' 0s
21            if (windowEnd - windowStart + 1 - maxOnesCount > k) {
22                if (arr[windowStart] == 1) {
23                    maxOnesCount--;
24                }
25                windowStart++;
26            }
27
28            maxLength = max(maxLength, windowEnd - windowStart + 1);
29        }
30
31        return maxLength;
32    }
33 };
34
35 int main(int argc, char* argv[]) {
36     cout << ReplacingOnes::findLength(vector<int>{0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1}, 2) << endl;
37     cout << ReplacingOnes::findLength(vector<int>{0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1}, 3) << endl;
38 }
39
```

Output1.743s

69

### Time Complexity #

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

### Space Complexity #

The algorithm runs in constant space  $O(1)$ .