Longest Subarray with Ones after Replacement (hard)

We'll cover the following

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 c
ontiguous 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 long
est contiguous subarray of 1s having length 9.
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
class ReplacingOnes {
  public static int findLength(int[] arr, int k) {
    // TODO: Write your code here
    return -1;
  }
}

```

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/5668639101419520/5671464854355960 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:

Java | Python3 | C++ | JS

```
      if (arr[windowEnd] == 1)
        maxOnesCount++;

      // current window size is from windowStart to windowEnd, overall we hav
      // repeating a maximum of 'maxOnesCount' times, this means that we can
      // 'maxOnesCount' 1s and the remaining are 0s which should replace with
      // now, if the remaining 0s are more than 'k', it is the time to shrink
      // are not allowed to replace more than 'k' 0s
      if (windowEnd - windowStart + 1 - maxOnesCount > k) {
        if (arr[windowStart] == 1)
          maxOnesCount--;
        windowStart++;
      }

      maxLength = Math.max(maxLength, windowEnd - windowStart + 1);
    }

    return maxLength;
  }

  public static void main(String[] args) {
    System.out.println(ReplacingOnes.findLength(new int[] { 0, 1, 1, 0, 0, 0,
    System.out.println(ReplacingOnes.findLength(new int[] { 0, 1, 0, 0, 1, 1,
  }
}
```

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 $O(1)$.