## Find zeroes to be flipped so that number of consecutive 1's is maximized

Given a binary array and an integer m, find the position of zeroes flipping which creates maximum number of consecutive 1s in array.

Examples:

```
Input:   arr[] = {1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1}
         m = 2
Output:  5 7
We are allowed to flip maximum 2 zeroes. If we flip
arr[5] and arr[7], we get 8 consecutive 1's which is
maximum possible under given constraints

Input:   arr[] = {1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1}
         m = 1
Output:  7
We are allowed to flip maximum 1 zero. If we flip
arr[7], we get 5 consecutive 1's which is maximum
possible under given constraints.

Input:   arr[] = {0, 0, 0, 1}
         m = 4
Output:  0 1 2
Since m is more than number of zeroes, we can flip
all zeroes.
```

Source: http://www.careercup.com/question?id=5106425965576192

**We strongly recommend that you click here and practice it, before moving on to the solution.**

A **Simple Solution** is to consider every subarray by running two loops. For every subarray, count number of zeroes in it. Return the maximum size subarray with m or less zeroes. Time Complexity of this solution is $O(n^2)$.

A **Better Solution** is to use auxiliary space to solve the problem in O(n) time.

For all positions of 0's calculate left[] and right[] which defines the number of consecutive 1's to the left of i and right of i respectively.

For example, for arr[] = {1, 1, 0, 1, 1, 0, 0, 1, 1, 1} and m = 1, left[2] = 2 and right[2] = 2, left[5] = 2 and right[5] = 0, left[6] = 0 and right[6] = 3.

left[] and right[] can be filled in O(n) time by traversing array once and keeping track of last seen 1 and last seen 0. While filling left[] and right[], we also store indexes of all zeroes in a third array say zeroes[]. For above example, this third array stores {2, 5, 6}

Now traverse zeroes[] and for all consecutive m entries in this array, compute the sum of 1s that can be produced. This step can be done in O(n) using left[] and right[].

An **Efficient Solution** can solve the problem in O(n) time and O(1) space. The idea is to use Sliding Window for the given array. The solution is taken from here.
Let us use a window covering from index wL to index wR. Let the number of zeros inside the window be zeroCount. We maintain the window with at most m zeros inside.

The main steps are:
– While zeroCount is no more than m: expand the window to the right (wR++) and update the count zeroCount.
– While zeroCount exceeds m, shrink the window from left (wL++), update zeroCount;
– Update the widest window along the way. The positions of output zeros are inside the best window.

Below is C++ implementation of the idea.

```
// C++ program to find positions of zeroes flipping which
// produces maximum number of xonsecutive 1's
#include<bits/stdc++.h>
using namespace std;

// m is maximum of number zeroes allowed to flip
// n is size of array
void findZeroes(int arr[], int n, int m)
{
    // Left and right indexes of current window
    int wL = 0, wR = 0;

    // Left index and size of the widest window
    int bestL = 0, bestWindow = 0;

    // Count of zeroes in current window
    int zeroCount = 0;

    // While right boundary of current window doesn't cross
    // right end
    while (wR < n)
    {
        // If zero count of current window is less than m,
        // widen the window toward right
        if (zeroCount <= m)
        {
            if (arr[wR] == 0)
              zeroCount++;
            wR++;
        }

        // If zero count of current window is more than m,
        // reduce the window from left
        if (zeroCount > m)
        {
            if (arr[wL] == 0)
              zeroCount--;
            wL++;
        }

        // Updqate widest window if this window size is more
        if (wR-wL > bestWindow)
        {
            bestWindow = wR-wL;
            bestL = wL;
        }
    }

    // Print positions of zeroes in the widest window
    for (int i=0; i<bestWindow; i++)
    {
        if (arr[bestL+i] == 0)
           cout << bestL+i << " ";
    }
}

// Driver program
int main()
{
   int arr[] = {1, 0, 0, 1, 1, 0, 1, 0, 1, 1};
   int m = 2;
   int n =  sizeof(arr)/sizeof(arr[0]);
   cout << "Indexes of zeroes to be flipped are ";
   findZeroes(arr, n, m);
   return 0;
}
```

Output:

```
Indexes of zeroes to be flipped are 5 7
```