Longest Span with same Sum in two Binary arrays

Given two binary arrays arr1[] and arr2[] of same size n. Find length of the longest common span (i, j) where j >= i such that arr1[i] + arr1[i+1] + …. + arr1[j] = arr2[i] + arr2[i+1] + …. + arr2[j].

Expected time complexity is Θ(n).

Examples:

```
Input: arr1[] = {0, 1, 0, 0, 0, 0};
       arr2[] = {1, 0, 1, 0, 0, 1};
Output: 4
The longest span with same sum is from index 1 to 4.

Input: arr1[] = {0, 1, 0, 1, 1, 1, 1};
       arr2[] = {1, 1, 1, 1, 1, 0, 1};
Output: 6
The longest span with same sum is from index 1 to 6.

Input: arr1[] = {0, 0, 0};
       arr2[] = {1, 1, 1};
Output: 0

Input: arr1[] = {0, 0, 1, 0};
       arr2[] = {1, 1, 1, 1};
Output: 1
```

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

Method 1 (Simple Solution)

One by one by consider same subarrays of both arrays. For all subarrays, compute sums and if sums are same and current length is more than max length, then update max length. Below is C++ implementation of simple approach.

```
// A Simple C++ program to find longest common
// subarray of two binary arrays with same sum
#include<bits/stdc++.h>
using namespace std;

// Returns length of the longest common subarray
// with same sum
int longestCommonSum(bool arr1[], bool arr2[], int n)
{
   // Initialize result
   int maxLen = 0;

   // One by one pick all possible starting points
   // of subarrays
   for (int i=0; i<n; i++)
   {
     // Initialize sums of current subarrays
     int sum1 = 0, sum2 = 0;

     // Conider all points for starting with arr[i]
     for (int j=i; j<n; j++)
     {
        // Update sums
        sum1 += arr1[j];
        sum2 += arr2[j];

        // If sums are same and current length is
        // more than maxLen, update maxLen
        if (sum1 == sum2)
        {
          int len = j-i+1;
          if (len > maxLen)
            maxLen = len;
        }
     }
   }
   return maxLen;
}

// Driver progra+m to test above function
int main()
{
   bool  arr1[] = {0, 1, 0, 1, 1, 1, 1};
   bool  arr2[] = {1, 1, 1, 1, 1, 0, 1};
   int n = sizeof(arr1)/sizeof(arr1[0]);
   cout << "Length of the longest common span with same "
        "sum is "<< longestCommonSum(arr1, arr2, n);
   return 0;
}
```

Output:

```
Length of the longest common span with same sum is 6
```

Time Complexity: $O(n^2)$
Auxiliary Space: O(1)

Method 2 (Using Auxiliary Array)

The idea is based on below observations.

1. Since there are total n elements, maximum sum is n for both arrays.
2. Difference between two sums varies from **-n** to **n**. So there are total 2n + 1 possible values of difference.
3. If differences between prefix sums of two arrays become same at two points, then subarrays between these two points have same sum.

Below is Complete Algorithm.

1. Create an auxiliary array of size 2n+1 to store starting points of all possible values of differences (Note that possible values of

differences vary from -n to n, i.e., there are total 2n+1 possible values)

2. Initialize starting points of all differences as -1.
3. Initialize **maxLen** as 0 and prefix sums of both arrays as 0, **preSum1** = 0, **preSum2** = 0
4. Travers both arrays from i = 0 to n-1.
    a. Update prefix sums: preSum1 += arr1[i], preSum2 += arr2[i]
    b. Compute difference of current prefix sums: **curr_diff** = preSum1 – preSum2
    c. Find index in diff array: **diffIndex** = n + curr_diff // curr_diff can be negative and can go till -n
    d. **If** curr_diff is 0, then i+1 is maxLen so far
    e. **Else If** curr_diff is seen first time, i.e., starting point of current diff is -1, then update starting point as i
    f. **Else** (curr_diff is NOT seen first time), then consider i as ending point and find length of current same sum span. If this length is more, then update maxLen
5. Return maxLen

Below is C++ implementation of above algorithm.

```
// A O(n) and O(n) extra space C++ program to find
// longest common subarray of two binary arrays with
// same sum
#include<bits/stdc++.h>
using namespace std;

// Returns length of the longest common sum in arr1[]
// and arr2[]. Both are of same size n.
int longestCommonSum(bool arr1[], bool arr2[], int n)
{
  // Initialize result
  int maxLen = 0;

  // Initialize prefix sums of two arrays
  int preSum1 = 0, preSum2 = 0;

  // Create an array to store staring and ending
  // indexes of all possible diff values. diff[i]
  // would store starting and ending points for
  // difference "i-n"
  int diff[2*n+1];

  // Initialize all starting and ending values as -1.
  memset(diff, -1, sizeof(diff));

  // Traverse both arrays
  for (int i=0; i<n; i++)
  {
    // Update prefix sums
    preSum1 += arr1[i];
    preSum2 += arr2[i];

    // Comput current diff and index to be used
    // in diff array. Note that diff can be negative
    // and can have minimum value as -1.
    int curr_diff = preSum1 - preSum2;
    int diffIndex = n + curr_diff;

    // If current diff is 0, then there are same number
    // of 1's so far in both arrays, i.e., (i+1) is
    // maximum length.
    if (curr_diff == 0)
      maxLen = i+1;

    // If current diff is seen first time, then update
    // starting index of diff.
    else if ( diff[diffIndex] == -1)
      diff[diffIndex] = i;

    // Current diff is already seen
    else
    {
      // Find lenght of this same sum common span
      int len = i - diff[diffIndex];
```

```
            // Update max len if needed
            if (len > maxLen)
                maxLen = len;
        }
    }
    return maxLen;
}

// Driver progra+m to test above function
int main()
{
    bool  arr1[] = {0, 1, 0, 1, 1, 1, 1};
    bool  arr2[] = {1, 1, 1, 1, 1, 0, 1};
    int n = sizeof(arr1)/sizeof(arr1[0]);
    cout << "Length of the longest common span with same "
          "sum is "<< longestCommonSum(arr1, arr2, n);
    return 0;
}
```

Output:

```
Length of the longest common span with same sum is 6
```

Time Complexity: Θ(n)
Auxiliary Space: Θ(n)