Largest subarray with equal number of 0s and 1s

Given an array containing only 0s and 1s, find the largest subarray which contain equal no of 0s and 1s. Expected time complexity is O(n).

Examples:

```
Input: arr[] = {1, 0, 1, 1, 1, 0, 0}
Output: 1 to 6 (Starting and Ending indexes of output subarray)

Input: arr[] = {1, 1, 1, 1}
Output: No such subarray

Input: arr[] = {0, 0, 1, 1, 0}
Output: 0 to 3 Or 1 to 4
```

Source: Largest subarray with equal number of 0s and 1s

Method 1 (Simple)

A simple method is to use two nested loops. The outer loop picks a starting point i. The inner loop considers all subarrays starting from i. If size of a subarray is greater than maximum size so far, then update the maximum size.
In the below code, 0s are considered as -1 and sum of all values from i to j is calculated. If sum becomes 0, then size of this subarray is compared with largest size so far.

C

```
// A simple program to find the largest subarray with equal number of 0s and 1s
#include <stdio.h>

// This function Prints the starting and ending
// indexes of the largest subarray with equal
// number of 0s and 1s. Also returns the size
// of such subarray.
int findSubArray(int arr[], int n)
{
    int sum = 0;
    int maxsize = -1, startindex;

    // Pick a starting point as i
    for (int i = 0; i < n-1; i++)
    {
        sum = (arr[i] == 0)? -1 : 1;

        // Consider all subarrays starting from i
        for (int j = i+1; j < n; j++)
        {
            (arr[j] == 0)? sum += -1: sum += 1;

            // If this is a 0 sum subarray, then
            // compare it with maximum size subarray
            // calculated so far
            if (sum == 0 && maxsize < j-i+1)
            {
                maxsize = j - i + 1;
                startindex = i;
            }
        }
    }
    if (maxsize == -1)
        printf("No such subarray");
    else
        printf("%d to %d", startindex, startindex+maxsize-1);

    return maxsize;
}

/* Driver program to test above functions*/
int main()
{
    int arr[] =  {1, 0, 0, 1, 0, 1, 1};
    int size = sizeof(arr)/sizeof(arr[0]);

    findSubArray(arr, size);
    return 0;
}
```

Java

```
class LargestSubArray
{

    // This function Prints the starting and ending
    // indexes of the largest subarray with equal
    // number of 0s and 1s. Also returns the size
    // of such subarray.
    int findSubArray(int arr[], int n)
    {
        int sum = 0;
        int maxsize = -1, startindex = 0;
        int endindex = 0;

        // Pick a starting point as i
        for (int i = 0; i < n - 1; i++)
        {
            sum = (arr[i] == 0) ? -1 : 1;

            // Consider all subarrays starting from i
            for (int j = i + 1; j < n; j++)
            {
                if(arr[j] == 0)
                    sum += -1;
                else
                    sum += 1;

                // If this is a 0 sum subarray, then
                // compare it with maximum size subarray
                // calculated so far
                if (sum == 0 && maxsize < j - i + 1)
                {
                    maxsize = j - i + 1;
                    startindex = i;
                }
            }
        }
        endindex = startindex+maxsize-1;
        if (maxsize == -1)
            System.out.println("No such subarray");
        else
            System.out.println(startindex+" to "+endindex);

        return maxsize;
    }

    /* Driver program to test the above functions */
    public static void main(String[] args)
    {
        LargestSubArray sub;
        sub = new LargestSubArray();
        int arr[] = {1, 0, 0, 1, 0, 1, 1};
        int size = arr.length;

        sub.findSubArray(arr, size);
    }
}
```

Output:

```
0 to 5
```

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

Method 2 (Tricky)
Following is a solution that uses O(n) extra space and solves the problem in O(n) time complexity.
Let input array be arr[] of size n and maxsize be the size of output subarray.

1) Consider all 0 values as -1. The problem now reduces to find out the maximum length subarray with sum = 0.
2) Create a temporary array sumleft[] of size n. Store the sum of all elements from arr[0] to arr[i] in sumleft[i]. This can be done in O(n) time.
3) There are two cases, the output subarray may start from 0th index or may start from some other index. We will return the max of the values obtained by two cases.
4) To find the maximum length subarray starting from 0th index, scan the sumleft[] and find the maximum i where sumleft[i] = 0.
5) Now, we need to find the subarray where subarray sum is 0 and start index is not 0. This problem is equivalent to finding two indexes i & j in sumleft[] such that sumleft[i] = sumleft[j] and j-i is maximum. To solve this, we can create a hash table with size = max-min+1 where min is the minimum value in the sumleft[] and max is the maximum value in the sumleft[]. The idea is to hash the leftmost occurrences of all different values in sumleft[]. The size of hash is chosen as max-min+1 because there can be these many different possible values in sumleft[]. Initialize all values in hash as -1
6) To fill and use hash[], traverse sumleft[] from 0 to n-1. If a value is not present in hash[], then store its index in hash. If the value is present, then calculate the difference of current index of sumleft[] and previously stored value in hash[]. If this difference is more than maxsize, then update the maxsize.
7) To handle corner cases (all 1s and all 0s), we initialize maxsize as -1. If the maxsize remains -1, then print there is no such subarray.

C

```c
// A O(n) program to find the largest subarray
// with equal number of 0s and 1s
#include <stdio.h>
#include <stdlib.h>

// A utility function to get maximum of two
// integers
int max(int a, int b) { return a>b? a: b; }

// This function Prints the starting and ending
// indexes of the largest subarray with equal
// number of 0s and 1s. Also returns the size
// of such subarray.
int findSubArray(int arr[], int n)
{
    // variables to store result values
    int maxsize = -1, startindex;

    // Create an auxiliary array sunmleft[].
    // sumleft[i] will be sum of array
    // elements from arr[0] to arr[i]
    int sumleft[n];

    // For min and max values in sumleft[]
    int min, max;
    int i;

    // Fill sumleft array and get min and max
    // values in it.  Consider 0 values in arr[]
    // as -1
    sumleft[0] = ((arr[0] == 0)? -1: 1);
    min = arr[0]; max = arr[0];
    for (i=1; i<n; i++)
    {
        sumleft[i] = sumleft[i-1] + ((arr[i] == 0)?
                     -1: 1);
        if (sumleft[i] < min)
            min = sumleft[i];
        if (sumleft[i] > max)
            max = sumleft[i];
    }

    // Now calculate the max value of j - i such
    // that sumleft[i] = sumleft[j]. The idea is
    // to create a hash table to store indexes of all
    // visited values.
    // If you see a value again, that it is a case of
    // sumleft[i] = sumleft[j]. Check if this j-i is
    // more than maxsize.
    // The optimum size of hash will be max-min+1 as
    // these many different values of sumleft[i] are
    // these many different values of sumleft[i] are
    // possible. Since we use optimum size, we need
    // to shift all values in sumleft[] by min before
    // using them as an index in hash[].
    int hash[max-min+1];

    // Initialize hash table
    for (i=0; i<max-min+1; i++)
        hash[i] = -1;

    for (i=0; i<n; i++)
    {
        // Case 1: when the subarray starts from
        //         index 0
        if (sumleft[i] == 0)
        {
           maxsize = i+1;
           startindex = 0;
        }

        // Case 2: fill hash table value. If already
        //         filled, then use it
        if (hash[sumleft[i]-min] == -1)
            hash[sumleft[i]-min] = i;
        else
        {
            if ((i - hash[sumleft[i]-min]) > maxsize)
            {
                maxsize = i - hash[sumleft[i]-min];
                startindex = hash[sumleft[i]-min] + 1;
            }
        }
    }
    if (maxsize == -1)
        printf("No such subarray");
    else
        printf("%d to %d", startindex, startindex+maxsize-1);

    return maxsize;
}

/* Driver program to test above functions */
int main()
{
    int arr[] =  {1, 0, 0, 1, 0, 1, 1};
    int size = sizeof(arr)/sizeof(arr[0]);

    findSubArray(arr, size);
    return 0;
}
```

C++/STL

```cpp
// C++ program to find largest subarray with equal number of
// 0's and 1's.
#include <bits/stdc++.h>
using namespace std;

// Returns largest subarray with equal number of 0s and 1s
int maxLen(int arr[], int n)
{
    // Creates an empty hashMap hM
    unordered_map<int, int> hM;

    int sum = 0;     // Initialize sum of elements
    int max_len = 0; // Initialize result
    int ending_index = -1;

    for (int i = 0; i < n; i++)
        arr[i] = (arr[i] == 0)? -1: 1;

    // Traverse through the given array
    for (int i = 0; i < n; i++)
    {
        // Add current element to sum
        sum += arr[i];

        // To handle sum=0 at last index
        if (sum == 0)
        {
            max_len = i + 1;
            ending_index = i;
        }

        // If this sum is seen before, then update max_len
        // if required
        if (hM.find(sum + n) != hM.end())
        {
            if (max_len < i - hM[sum + n])
            {
                max_len = i - hM[sum + n];
                ending_index = i;
            }
        }
        else // Else put this sum in hash table
            hM[sum + n] = i;
    }

    for (int i = 0; i < n; i++)
        arr[i] = (arr[i] == -1)? 0: 1;

    printf("%d to %d\n", ending_index-max_len+1, ending_index);

    return max_len;
}

// Driver method
int main()
{
    int arr[] = {1, 0, 0, 1, 0, 1, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

    maxLen(arr, n);
    return 0;
}
// This code is contributed by Aditya Goel
```

Java

```java
import java.util.HashMap;

class LargestSubArray1
{

    // Returns largest subarray with equal number of 0s and 1s
    int maxLen(int arr[], int n)
    {
        // Creates an empty hashMap hM
        HashMap<Integer, Integer> hM = new HashMap<Integer, Integer>();

        int sum = 0;     // Initialize sum of elements
        int max_len = 0; // Initialize result
        int ending_index = -1;
        int start_index = 0;

        for (int i = 0; i < n; i++)
        {
            arr[i] = (arr[i] == 0) ? -1 : 1;
        }

        // Traverse through the given array
        for (int i = 0; i < n; i++)
        {
            // Add current element to sum
            sum += arr[i];

            // To handle sum=0 at last index
            if (sum == 0)
            {
                max_len = i + 1;
                ending_index = i;
            }

            // If this sum is seen before, then update max_len
            // if required
            if (hM.containsKey(sum))
            {
                if (max_len < i - hM.get(sum + n))
                {
                    max_len = i - hM.get(sum + n);
                    ending_index = i;
                }
            }
            else // Else put this sum in hash table
                hM.put(sum + n, i);
        }

        for (int i = 0; i < n; i++)
        {
            arr[i] = (arr[i] == -1) ? 0 : 1;
        }

        int end = ending_index - max_len + 1;
        System.out.println(end + " to " + ending_index);

        return max_len;
    }

    /* Driver program to test the above functions */
    public static void main(String[] args)
    {
        LargestSubArray1 sub = new LargestSubArray1();
        int arr[] = {1, 0, 0, 1, 0, 1, 1};
        int n = arr.length;

        sub.maxLen(arr, n);
    }
}

// This code has been by Mayank Jaiswal(mayank_24)
```

Output:

```
0 to 5
```

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