Find the largest subarray with 0 sum

Given an array of integers, find length of the largest subarray with sum equals to 0.

Examples:

```
Input: arr[] = {15, -2, 2, -8, 1, 7, 10, 23};
Output: 5
The largest subarray with 0 sum is -2, 2, -8, 1, 7

Input: arr[] = {1, 2, 3}
Output: 0
There is no subarray with 0 sum

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

We strongly recommend you to minimize your browser and try this yourself first.

A **simple solution** is to consider all subarrays one by one and check the sum of every subarray. We can run two loops: the outer loop picks a starting point i and the inner loop tries all subarrays starting from i. Time complexity of this method is $O(n^2)$.

Below are implementations of this solution.

C/C++

```
/* A simple C++ program to find largest subarray with 0 sum */
#include<bits/stdc++.h>
using namespace std;

// Returns length of the largest subarray with 0 sum
int maxLen(int arr[], int n)
{
    int max_len = 0; // Initialize result

    // Pick a starting point
    for (int i = 0; i < n; i++)
    {
        // Initialize currr_sum for every starting point
        int curr_sum = 0;

        // try all subarrays starting with 'i'
        for (int j = i; j < n; j++)
        {
            curr_sum += arr[j];

            // If curr_sum becomes 0, then update max_len
            // if required
            if (curr_sum == 0)
               max_len = max(max_len, j-i+1);
        }
    }
    return max_len;
}

// Driver program to test above function
int main()
{
    int arr[] = {15, -2, 2, -8, 1, 7, 10, 23};
    int n = sizeof(arr)/sizeof(arr[0]);
    cout << "Length of the longest 0 sum subarray is "
         << maxLen(arr, n);
    return 0;
}
```

Python

```
# Python program to find the length of largest subarray with 0 sum

# returns the length
def maxLen(arr):

    # initialize result
    max_len = 0

    # pick a starting point
    for i in range(len(arr)):

        # initialize sum for every starting point
        curr_sum = 0

        # try all subarrays starting with 'i'
        for j in range(i, len(arr)):

            curr_sum += arr[j]

            # if curr_sum becomes 0, then update max_len
            if curr_sum == 0:
                max_len = max(max_len, j-i+1)

    return max_len


# test array
arr = [15, -2, 2, -8, 1, 7, 10, 13]

print "Length of the longest 0 sum subarray is %d" %  maxLen(arr)
```

Output:

```
Length of the longest 0 sum subarray is 5
```

We can **Use Hashing** to solve this problem in O(n) time. The idea is to iterate through the array and for every element arr[i], calculate sum of elements form 0 to i (this can simply be done as sum += arr[i]). If the current sum has been seen before, then there is a zero sum array. Hashing is used to store the sum values, so that we can quickly store sum and find out whether the current sum is seen before or not.

Following are implementations of the above approach.

Java

```
// A Java program to find maximum length subarray with 0 sum
import java.util.HashMap;

class MaxLenZeroSumSub {

    // Returns length of the maximum length subarray with 0 sum
    static int maxLen(int arr[])
    {
        // 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

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

            if (arr[i] == 0 && max_len == 0)
                max_len = 1;

            if (sum == 0)
                max_len = i+1;

            // Look this sum in hash table
            Integer prev_i = hM.get(sum);

            // If this sum is seen before, then update max_len
            // if required
            if (prev_i != null)
               max_len = Math.max(max_len, i-prev_i);
            else  // Else put this sum in hash table
               hM.put(sum, i);
        }

        return max_len;
    }

    // Drive method
    public static void main(String arg[])
    {
        int arr[] = {15, -2, 2, -8, 1, 7, 10, 23};
        System.out.println("Length of the longest 0 sum subarray is "
                           + maxLen(arr));
    }
}
```

C++

```
// C++ program to find the length of largest subarray with 0 sum
#include <bits/stdc++.h>
using namespace std;

// Returns Length of the required subarray
int maxLen(int arr[], int n)
{
    // Map to store the previous sums
    map<int, int> presum;

    int sum = 0;  // Initialise the sum of elements
    int max_len = 0; // Initialise result

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

        if (arr[i]==0 && max_len==0)
            max_len = 1;
        if (sum == 0)
            max_len = i+1;

        // Look for this sum in Hash table
        if(presum.find(sum) != presum.end())
        {
            // If this sum is seen before, then update max_len
            max_len = max(max_len, i-presum[sum]);
        }
        else
        {
            // Else insert this sum with index in hash table
            presum[sum] = i;
        }
    }

    return max_len;
}

// Driver Program to test above function
int main()
{
    int arr[] = {15, -2, 2, -8, 1, 7, 10, 23};
    int n = sizeof(arr)/sizeof(arr[0]);

    cout<<"Length of the longest 0 sum subarray is "<< maxLen(arr, n);

    return 0;
}
```

Python

```
# A python program to find maximum length subarray with 0 sum in o(n) time

# Returns the maximum length
def maxLen(arr):

    # NOTE: Dictonary in python in implemented as Hash Maps
    # Create an empty hash map (dictonary)
    hash_map = {}

    # Initialize result
    max_len = 0

    # Initialize sum of elements
    curr_sum = 0

    # Traverse through the given array
    for i in range(len(arr)):

        # Add the current element to the sum
        curr_sum += arr[i]

        if arr[i] is 0 and max_len is 0:
            max_len = 1

        if curr_sum is 0:
            max_len = i+1

        # NOTE: 'in' operation in dictonary to search key takes O(1)
        # Look if current sum is seen before
        if curr_sum in hash_map:
            max_len = max(max_len, i - hash_map[curr_sum] )
        else:
            # else put this sum in dictonary
            hash_map[curr_sum] = i

    return max_len


# test array
arr = [15, -2, 2, -8, 1, 7, 10, 13]

print "Length of the longest 0 sum subarray is %d" %  maxLen(arr)
```

Output:

```
Length of the longest 0 sum subarray is 5
```

Time Complexity of this solution can be considered as O(n) under the assumption that we have good hashing function that allows insertion and retrieval operations in O(1) time.