Smallest subarray with sum greater than a given value

Given an array of integers and a number x, find the smallest subarray with sum greater than the given value.

```
Examples:
arr[] = {1, 4, 45, 6, 0, 19}
   x  =  51
Output: 3
Minimum length subarray is {4, 45, 6}

arr[] = {1, 10, 5, 2, 7}
   x  = 9
Output: 1
Minimum length subarray is {10}

arr[] = {1, 11, 100, 1, 0, 200, 3, 2, 1, 250}
    x = 280
Output: 4
Minimum length subarray is {100, 1, 0, 200}
```

A **simple solution** is to use two nested loops. The outer loop picks a starting element, the inner loop considers all elements (on right side of current start) as ending element. Whenever sum of elements between current start and end becomes more than the given number, update the result if current length is smaller than the smallest length so far.

Following is C++ implementation of simple approach.

```
# include <iostream>
using namespace std;

// Returns length of smallest subarray with sum greater than x.
// If there is no subarray with given sum, then returns n+1
int smallestSubWithSum(int arr[], int n, int x)
{
    //  Initilize length of smallest subarray as n+1
     int min_len = n + 1;

     // Pick every element as starting point
```

```
     for (int start=0; start<n; start++)
     {
          // Initialize sum starting with current start
          int curr_sum = arr[start];

          // If first element itself is greater
          if (curr_sum > x) return 1;

          // Try different ending points for curremt start
          for (int end=start+1; end<n; end++)
          {
              // add last element to current sum
              curr_sum += arr[end];

              // If sum becomes more than x and length of
              // this subarray is smaller than current smallest
              // length, update the smallest length (or result)
              if (curr_sum > x && (end - start + 1) < min_len)
                 min_len = (end - start + 1);
          }
     }
     return min_len;
}

/* Driver program to test above function */
int main()
{
    int arr1[] = {1, 4, 45, 6, 10, 19};
    int x = 51;
    int n1 = sizeof(arr1)/sizeof(arr1[0]);
    cout << smallestSubWithSum(arr1, n1, x) << endl;

    int arr2[] = {1, 10, 5, 2, 7};
    int n2 = sizeof(arr2)/sizeof(arr2[0]);
    x  = 9;
    cout << smallestSubWithSum(arr2, n2, x) << endl;

    int arr3[] = {1, 11, 100, 1, 0, 200, 3, 2, 1, 250};
    int n3 = sizeof(arr3)/sizeof(arr3[0]);
    x  = 280;
    cout << smallestSubWithSum(arr3, n3, x) << endl;

    return 0;
}
```

Output:

```
3
1
4
```

Time Complexity: Time complexity of the above approach is clearly $O(n^2)$.

Efficient Solution: This problem can be solved in **O(n) time** using the idea used in this post. Thanks to Ankit and Nitin for suggesting this optimized solution.

```
// O(n) solution for finding smallest subarray with sum
// greater than x
#include <iostream>
using namespace std;

// Returns length of smallest subarray with sum greater than x.
// If there is no subarray with given sum, then returns n+1
int smallestSubWithSum(int arr[], int n, int x)
{
    // Initialize current sum and minimum length
    int curr_sum = 0, min_len = n+1;

    // Initialize starting and ending indexes
    int start = 0, end = 0;
    while (end < n)
    {
        // Keep adding array elements while current sum
        // is smaller than x
        while (curr_sum <= x && end < n)
            curr_sum += arr[end++];

        // If current sum becomes greater than x.
        while (curr_sum > x && start < n)
        {
            // Update minimum length if needed
            if (end - start < min_len)
                min_len = end - start;

            // remove starting elements
            curr_sum -= arr[start++];
        }
    }
    return min_len;
}

/* Driver program to test above function */
int main()
{
    int arr1[] = {1, 4, 45, 6, 10, 19};
    int x = 51;
    int n1 = sizeof(arr1)/sizeof(arr1[0]);
    cout << smallestSubWithSum(arr1, n1, x) << endl;

    int arr2[] = {1, 10, 5, 2, 7};
```

```
    int n2 = sizeof(arr2)/sizeof(arr2[0]);
    x  = 9;
    cout << smallestSubWithSum(arr2, n2, x) << endl;

    int arr3[] = {1, 11, 100, 1, 0, 200, 3, 2, 1, 250};
    int n3 = sizeof(arr3)/sizeof(arr3[0]);
    x  = 280;
    cout << smallestSubWithSum(arr3, n3, x);

    return 0;
}
```

Output:

```
3
1
4
```

This article is contributed by **Rahul Jain**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

27 Comments Category: Arrays

Related Posts:

- Longest Span with same Sum in two Binary arrays
- Count Inversions of size three in a give array
- Find the subarray with least average
- Count triplets with sum smaller than a given value
- Find zeroes to be flipped so that number of consecutive 1's is maximized
- Reorder an array according to given indexes

- Find maximum value of Sum(i*arr[i]) with only rotations on given array allowed
- Find maximum average subarray of k length

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.