Find subarray with given sum | Set 2 (Handles Negative Numbers)

Given an unsorted array of integers, find a subarray which adds to a given number. If there are more than one subarrays with sum as the given number, print any of them.

Examples:

```
Input: arr[] = {1, 4, 20, 3, 10, 5}, sum = 33
Ouptut: Sum found between indexes 2 and 4

Input: arr[] = {10, 2, -2, -20, 10}, sum = -10
Ouptut: Sum found between indexes 0 to 3

Input: arr[] = {-10, 0, 2, -2, -20, 10}, sum = 20
Ouptut: No subarray with given sum exists
```

We have discussed a solution that do not handles negative integers here. In this post, negative integers are also handled.

A simple solution is to consider all subarrays one by one and check if sum of every subarray is equal to given sum or not. The complexity of this solution would be O(n^2).

An efficient way is to use a map. The idea is to maintain sum of elements encountered so far in a variable (say curr_sum). Let the given number is sum. Now for each element, we check if curr_sum – sum exists in the map or not. If we found it in the map that means, we have a subarray present with given sum, else we insert curr_sum into the map and proceed to next element. If all elements of the array are processed and we didn't find any subarray with given sum, then subarray doesn't exists.

Below is C++ implementation of above idea –

```
// C++ program to print subarray with sum as given sum
#include<bits/stdc++.h>
using namespace std;

// Function to print subarray with sum as given sum
void subArraySum(int arr[], int n, int sum)
{
    // create an empty map
    unordered_map<int, int> map;

    // Maintains sum of elements so far
    int curr_sum = 0;

    for (int i = 0; i < n; i++)
    {
        // add current element to curr_sum
        curr_sum = curr_sum + arr[i];

        // if curr_sum is equal to target sum
        // we found a subarray starting from index 0
        // and ending at index i
        if (curr_sum == sum)
        {
            cout << "Sum found between indexes "
                 << 0 << " to " << i << endl;
            return;
        }

        // If curr_sum - sum already exists in map
        // we have found a subarray with target sum
        if (map.find(curr_sum - sum) != map.end())
        {
            cout << "Sum found between indexes "
                 << map[curr_sum - sum] + 1
                 << " to " << i << endl;
            return;
        }

        map[curr_sum] = i;
    }

    // If we reach here, then no subarray exists
    cout << "No subarray with given sum exists";
}

// Driver program to test above function
int main()
{
    int arr[] = {10, 2, -2, -20, 10};
    int n = sizeof(arr)/sizeof(arr[0]);
    int sum = -10;

    subArraySum(arr, n, sum);

    return 0;
}
```

Output:

```
Sum found between indexes 0 to 3
```

Time complexity of above solution is O(n) as we are doing only one traversal of the array.

Auxiliary space used by the program is O(n).