**Title:** **Find subarray with given sum in an array**

**Abstract:** A simple solution is to consider all subarrays one by one and check the sum of every subarray. Following program implements the simple solution. We run two loops: the outer loop picks a starting point i and the inner loop tries all subarrays starting from. Given an unsorted array of integers, find a subarray which adds to a given number. If there are more than one subarrays with the sum as the given number, print any of them.

**Methodology:**

arr[] = [1, 2, 3], n = 3

All subarrays : [1], [1, 2], [1, 2, 3],

[2], [2, 3], [3]

here first element 'arr[0]' appears 3 times

second element 'arr[1]' appears 4 times

third element 'arr[2]' appears 3 times

Every element arr[i] appears in two types of subsets:

i) In subarrays beginning with arr[i]. There are

(n-i) such subsets. For example [2] appears

in [2] and [2, 3].

ii) In (n-i)\*i subarrays where this element is not

first element. For example [2] appears in

[1, 2] and [1, 2, 3].

Total of above (i) and (ii) = (n-i) + (n-i)\*i

= **(n-i)(i+1)**

For arr[] = {1, 2, 3}, sum of subarrays is:

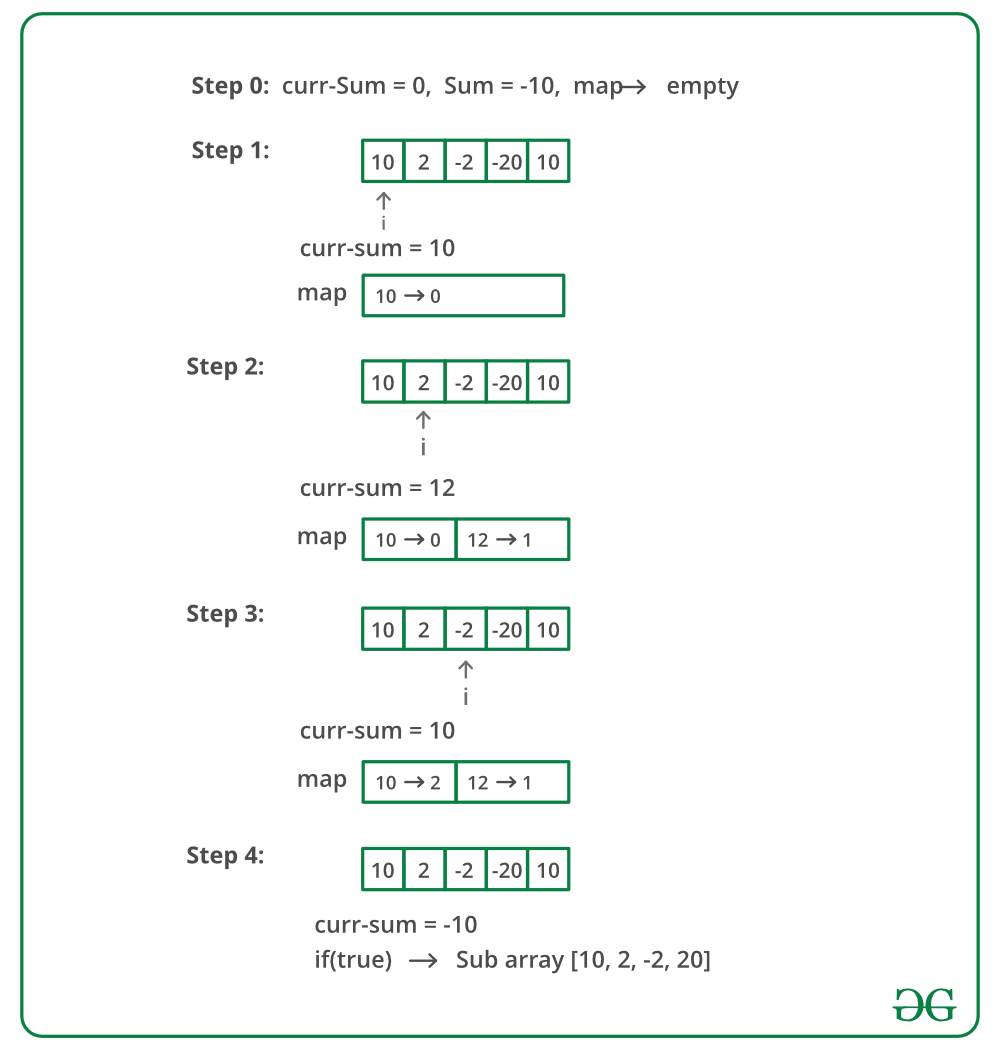
arr[0] \* ( 0 + 1 ) \* ( 3 - 0 ) +

arr[1] \* ( 1 + 1 ) \* ( 3 - 1 ) +

arr[2] \* ( 2 + 1 ) \* ( 3 - 2 )

= 1\*3 + 2\*4 + 3\*3

= 20



**Discussion:**

* Time complexity of method 2 looks more than O(n).
* If we take a closer look at the program, then we can figure out the time complexity is O(n).
* We can prove it by counting the number of operations performed on every element of arr[] in worst case.
* There are at most 2 operations performed on every element: (a) the element is added to the curr\_sum (b) the element is subtracted from curr\_sum.
* The upper bound on number of operations is 2n which is O(n).
* Initialize a variable curr\_sum as first element.
* curr\_sum indicates the sum of current subarray.
* Start from the second element and add all elements one by one to the curr\_sum.
* If curr\_sum becomes equal to sum, then print the solution.
* If curr\_sum exceeds the sum, then remove trailing elements while curr\_sum is greater than sum.

**Source Code:**

**#include<stdio.h>**

**void print(int a[], int i,int j)**

**{**

**printf("[%d..%d]---{ ",i,j);**

**for(int k = i; k<=j; k++){**

**printf("%d ",a[k]);**

**}**

**printf("}\n");**

**}**

**void subArray(int a[],int n, int sum)**

**{**

**for(int i=0;i<n;i++)**

**{**

**int sum\_of=0;**

**for(int j=i;j<n;j++)**

**{**

**sum\_of +=a[j];**

**if(sum\_of==sum){**

**print(a,i,j);**

**}**

**}**

**}**

**}**

**int main(){**

**int a[]={4,5,6,3,1,-7,8,3,-2};**

**int sum=9;**

**int n=sizeof(a)/sizeof(a[0]);**

**subArray(a,n,sum);**

**return 0;**

**}**

**Output:**

**[0..1]---{ 4 5 }**

**[2..3]---{ 6 3 }**

**[6..8]---{ 8 3 -2 }**

**Process returned 0 (0x0) execution time : 0.047 s**

**Press any key to continue.**