

## Largest Subarray with 0sum in C++

```
#include<bits/stdc++.h>
```

```
using namespace std;
```

```
int largest2(vector<int> arr, int n) {
    int max_len = 0;
    for (int i = 0; i < n; i++) {
        int sum = 0;
        for (int j = i; j < n; j++) {
            sum += arr[j];
            if (sum == 0) {
                max_len = max(max_len, j - i + 1);
            }
        }
    }

    return max_len;
}
```

```
int largest3(vector<int> arr, int n) {
    map<int, int> mapp;
    mapp[0]=-1;
    int sum=0;
    int ans=0;
    for (int i = 0; i < n; i++)
    {
        sum+=arr[i];
        if(mapp.find(sum)!=mapp.end()){
            auto it=mapp[sum];
            ans=max(ans,i- it);
        }
        else{
            mapp[sum]=i;
        }
    }
    return ans;
}
```

```
int
largestSubarrayWithZeroSum(vector<int>
& arr) {
    unordered_map<int, int> hm; // Maps
sum to index
    int sum = 0;
    int max_len = 0;

    hm[0] = -1; // Initialize to handle the case
where sum becomes 0 at the start

    for (int i = 0; i < arr.size(); i++) {
        sum += arr[i];

        if (hm.find(sum) != hm.end()) {
            int len = i - hm[sum];
            if (len > max_len) {
                max_len = len;
            }
        } else {
            hm[sum] = i;
        }
    }
}
```

### Step 1: Understanding the Problem

- We need to find the **largest subarray with sum = 0**.
- The input array is:  
  
{2, 8, -3, -5, 2, -4, 6, 1, 2, 1, -3, 4}
- The program runs **three different implementations** for this:
  - largestSubarrayWithZeroSum()** → Optimized using unordered\_map.
  - largest2()** → Brute-force approach.
  - largest3()** → Using map.

### Step 2: Dry Run for largestSubarrayWithZeroSum() (Optimized Hashing Approach)

Index (i)	arr[i]	Sum	hm (Sum → Index)	Max Length (max_len)
0	2	2	{0:-1, 2:0}	0
1	8	10	{0:-1, 2:0, 10:1}	0
2	-3	7	{0:-1, 2:0, 10:1, 7:2}	0
3	-5	2	<b>Found 2 at index 0 → 3 - 0 = 3</b>	3
4	2	4	{0:-1, 2:0, 10:1, 7:2, 4:4}	3
5	-4	0	<b>Found 0 at index -1 → 5 - (-1) = 6</b>	6
6	6	6	{0:-1, 2:0, 10:1, 7:2, 4:4, 6:6}	6
7	1	7	<b>Found 7 at index 2 → 7 - 2 = 5</b>	6
8	2	9	{0:-1, 2:0, 10:1, 7:2, 4:4, 6:6, 9:8}	6
9	1	10	<b>Found 10 at index 1 → 9 - 1 = 8</b>	8
10	-3	7	<b>Found 7 at index 2 → 10 - 2 = 8</b>	8

<

	<b>Index (i)</b>	<b>arr[i]</b>	<b>Sum</b>	<b>mapp (Sum → Index)</b>	<b>Max Length (ans)</b>
				4:4}	
	5	-4	0	<b>Found 0 at index -1 → 5 - (-1) = 6</b>	<b>6</b>
	6	6	6	{0:-1, 2:0, 10:1, 7:2, 4:4, 6:6}	6
	7	1	7	<b>Found 7 at index 2 → 7 - 2 = 5</b>	6
	8	2	9	{0:-1, 2:0, 10:1, 7:2, 4:4, 6:6, 9:8}	6
	9	1	10	<b>Found 10 at index 1 → 9 - 1 = 8</b>	<b>8</b>
	10	-3	7	<b>Found 7 at index 2 → 10 - 2 = 8</b>	8
	11	4	11	{0:-1, 2:0, 10:1, 7:2, 4:4, 6:6, 9:8, 11:11}	8
<b>Final Output of largest3() → 8</b>					
Final Outputs					
	<b>Function</b>		<b>Approach</b>	<b>Output</b>	
	largestSubarrayWithZeroSum()		Hashing (unordered_map)	8	
	largest2()		Brute-force (O(N²))	8	
	largest3()		Hashing (map)	8	
Output: 8 8 8					