Largest Subarray with 0sum in C++

```cpp
#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;
        }
    }
```

Dry Run:

Input:

arr = {2, 8, -3, -5, 2, -4, 6, 1, 2, 1, -3, 4}

Brute Force Approach (largest2):

- The outer loop starts from i = 0 and the inner loop starts from j = i to calculate the sum of subarrays.
- It checks if the sum becomes zero and keeps track of the maximum length of subarrays where the sum is zero.

For example:

- i = 0 to j = 5, sum = 0, length = 6, so max_len = 6.
- i = 1 to j = 7, sum = 0, length = 7, so max_len = 7.
- Continue the same till the end.

Optimized Approach (largest3):

- The map stores the cumulative sum at each index.
- It checks if the cumulative sum has been encountered before. If yes, then the subarray sum between those two indices is zero.

For example:

- At i = 0, cumulative sum = 2, map stores 2: 0.
- At i = 1, cumulative sum = 10, map stores 10: 1.
- At i = 2, cumulative sum = 7, map stores 7: 2.
- At i = 3, cumulative sum = 2, found 2 at index 0, so subarray length = 3.

Final Approach (largestSubarrayWithZeroSum):

- The logic here is very similar to the optimized approach. It uses the unordered map for efficiency. The result is calculated as the maximum length of subarrays with zero sum.

Output:

- For each method, the result is calculated as follows:
    - **Brute Force (largest2)**: 8
    - **Optimized Approach (largest3)**: 8

```
    return max_len;
}

int main() {
    vector<int> arr = {2, 8, -3, -5, 2, -4, 6, 1, 2, 1, -3, 4};
    int max_length =
largestSubarrayWithZeroSum(arr);
    cout << max_length << endl; // Output: 5

    int n=arr.size();
    int res=largest2(arr,n);
    cout<<res<<endl;

    int res3=largest3(arr,n);
    cout<<res3<<endl;

    return 0;
}
```

- **Final Approach (largestSubarrayWithZeroSum)**: 8

Final Output:

```
8
8
8
```

Output:

```
8
8
8
```