Subarray with zero sum

Given an array of positive and negative numbers, find if there is a subarray (of size at-least one) with 0 sum.

Examples :

> **Input:** {4, 2, -3, 1, 6}
>
> **Output:** true
>
> **Explanation:**
>
> There is a subarray with zero sum from index 1 to 3.
>
> **Input:** {4, 2, 0, 1, 6}
>
> **Output**: true
>
> **Explanation** :
>
> The third element is zero. A single element is also a sub-array.
>
> **Input:** {-3, 2, 3, 1, 6}
>
> **Output:** false

A **simple solution** is to consider all subarrays one by one and check the sum of every subarray. We can run two loops: the outer loop picks a starting point i and the inner loop tries all subarrays starting from i (See this for implementation). The time complexity of this method is O(n2).

We can also **use hashing**. The idea is to iterate through the array and for every element arr[i], calculate the sum of elements from 0 to i (this can simply be done as sum += arr[i]). If the current sum has been seen before, then there is a zero-sum array. Hashing is used to store the sum values so that we can quickly store sum and find out whether the current sum is seen before or not.

Example :

```
arr[] = {1, 4, -2, -2, 5, -4, 3}

If we consider all prefix sums, we can
notice that there is a subarray with 0
sum when :
1) Either a prefix sum repeats or
2) Or prefix sum becomes 0.

Prefix sums for above array are:
1, 5, 3,1, 6, 2, 5

Since prefix sum 1 repeats, we have a subarray
with 0 sum.
```

Time Complexity of this can be considered as O(n) under the assumption that we have good hashing function that allows insertion and retrieval operations in O(1) time.

Space Complexity: O(n) .Here we required extra space for unordered_set to insert array elements.