**Problem Name:** Three sum

**Topics:**

**Companies:**

**Level:** Easy

**Language:** C++

**Problem Statement**:

**Input Format:**

**Output Format:**

**Constraints:**

**Examples:**

**Brute force Solution:**

**Explanation:** The basic thinking logic for this is: Fix any one number in sorted array and find the other two numbers after it. The other two numbers can be easily found using two pointers (as array is sorted) and two numbers should have sum = -1\*(fixed number).

* Traverse the array and fix a number at every iteration.
* If number fixed is +ve, break there because we can't make it zero by searching after it.
* If number is getting repeated, ignore the lower loop and continue. This is for unique triplets. We want the last instance of the fixed number, if it is repeated.
* Make two pointers high and low, and initialize sum as 0.
* Search between two pointers, just similiar to binary search. Sum = num[i] + num[low] + num[high].
* If sum is -ve, means, we need more +ve numbers to make it 0, increament low (low++).
* If sum is +ve, means, we need more -ve numbers to make it 0, decreament high (high--).
* If sum is 0, that means we have found the required triplet, push it in answer vector.
* Now again, to avoid duplicate triplets, we have to navigate to last occurences of num[low] and num[high] respectively. Update the low and high with last occurences of low and high.

**Code:**

**Time Complexity**: O(N2) The total no. of comparison in worst case = Total no. of possible pairs = nC2 = n(n-1)/2 = O(n²)

**Space Complexity:** O(1)

**Optimized Solution:**

### Explanation: HashMap Approach:

In this approach, firstly, we will hash the indices of all elements in a hashMap. In case of repeated elements, the last occurence index would be stored in hashMap.

* Here also we fix a number (num[i]), by traversing the loop. But the loop traversal here for fixing numbers would leave last two indices. These last two indices would be covered by the nested loop.
* If number fixed is +ve, break there because we can't make it zero by searching after it.
* Make a nested loop to fix a number after the first fixed number. (num[j])
* To make sum 0, we would require the -ve sum of both fixed numbers. Let us say this required.
* Now, we will find the this required number in hashMap. If it exists in hashmap and its last occurrence index > 2nd fixed index, we found our triplet. Push it in answer vector.
* Update j to last occurence of 2nd fixed number to avoid duplicate triplets.
* Update i to last occurence of 1st fixed number to avoid duplicate triplets.
* Return answer vector.

**Code:**

**Time Complexity**:

**Space Complexity:**