34. 3 Sum | Brute - Better - Optimal with Codes

15. 3Sum

Medium 25136 2267 Add to List Share

Given an integer array nums, return all the triplets `[nums[i], nums[j], nums[k]]` such that `i != j, i != k,` and `j != k,` and `nums[i] + nums[j] + nums[k] == 0.`

Notice that the solution set must not contain duplicate triplets.

Example 1:

```
Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
Explanation:
nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.
nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.
nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.
The distinct triplets are [-1,0,1] and [-1,-1,2].
Notice that the order of the output and the order of the triplets does not
matter.
```

arr[i] + arr[j] + arr[k] = 0

(i != j != k)

You cannot take same element at once.

arr[] = {-1, 0, 1, 2, -1, -4}

{-1, 2, -1}

{0, 1, -1}

} You have to return unique triplet

1) Brute Force :

— Try out all triplet

arr[] = {-1, 0, 1, 2, -1, -4}

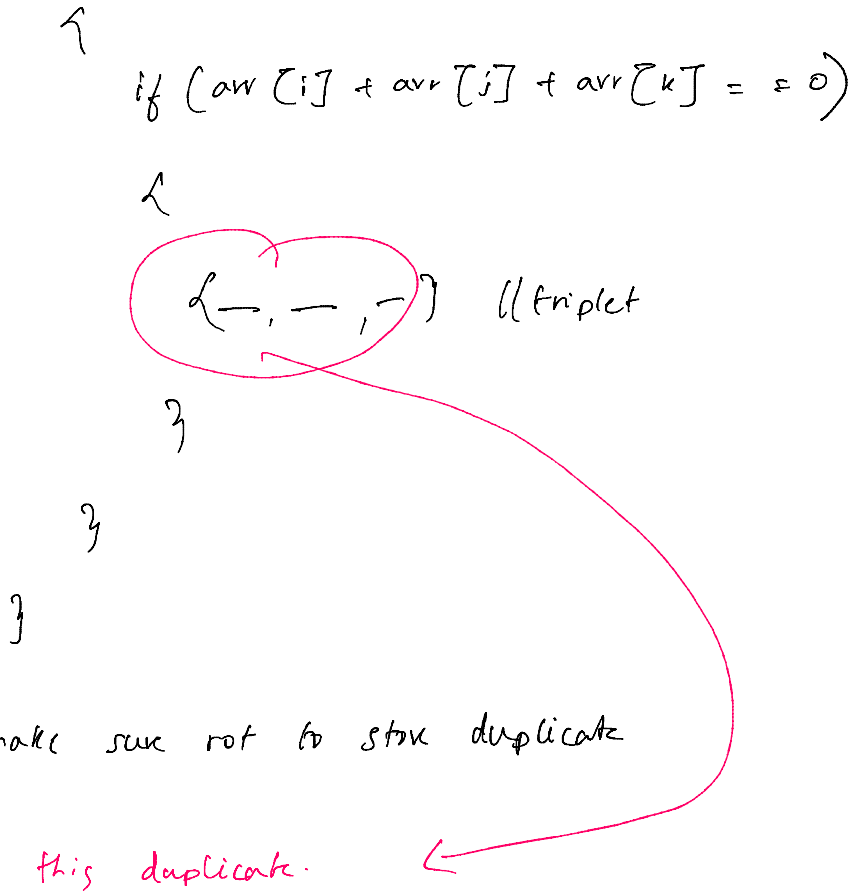
1st ele

so on...

[_ , _ , _]

1 2 3

```
for(i = 0; i<n; i++)
{
   for(j = i+1 ⟶ n)
   {
      for(k = j+1 ⟶ n)
```




```cpp
#include<bits/stdc++.h>
vector<vector<int>> triplet(int n, vector<int> &num)
{
    set<vector<int>> st; // store all the triple in form of set ds in vector
    for(int i = 0;i<n;i++){
        for(int j = i+1;j<n;j++){
            for(int k = j+1;k<n;k++){
                if(num[i] + num[j] + num[k] == 0){
                    vector<int> temp = {num[i], num[j], num[k]};
                    sort(temp.begin(), temp.end());
                    st.insert(temp);
                }
            }
        }
    }
    vector<vector<int>> ans(st.begin(), st.end());
    return ans;

}

```

T.C => $O(n^3 \times \log(\text{no of unique}))$

S.C => $2 \times O(\text{no of triplets})$

2) Better Solution :

arr[] = {-1, 0, 1, 2, -1, -4}

To do in $O(n^2)$, you need to get rid of 3rd loop.

$$arr[i] + arr[j] + arr[k] = 0$$

$$arr[k] = -(arr[i] + arr[j])$$

arr[] = {-1, 0, 1, 2, -1, -4}
(i = -1, j = -1)

$$arr[k] = -(-1 + -1)$$

$$= -(-2)$$

$$arr[k] = 2$$ (is there +2 in the array)

In order to look for arr[k] without iterating is hashing.

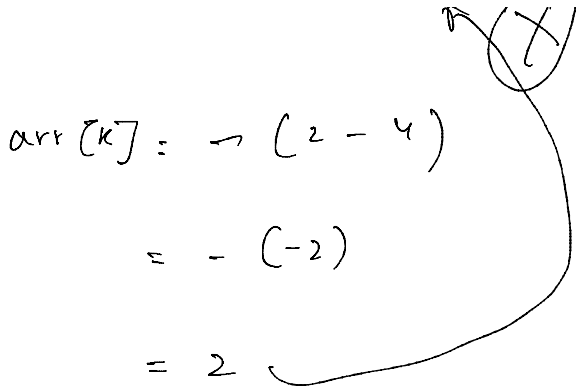
Dry Run for Hashing,

- You have to keep in mind you cannot pick one element twice.

E.g.

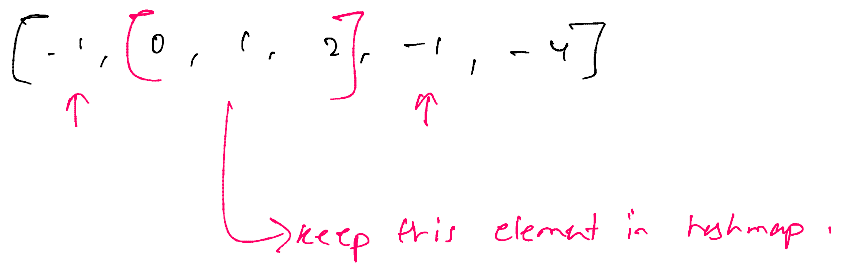
[-1, 0, 1, 2, -1, -4]

arr[k] = -(2 - 4)



So be mindful that if you are looking for 2 in the array then we have to be sure that it's not the current '2', '-4' that you have taken for arr[i] and arr[j] something different from those two.

You can do it by



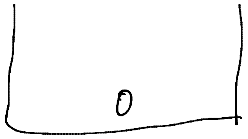
arr[] : [-1, 0, 1, 2, -1, -4]

i j j

arr[k] = -(arr[i] + arr[j])

= -(-1 + 0)

= 1 (is it in set NO)



set

then move j and keep the prev j in the hash map.

= - (-1 + 1)

= - (0)

= 0 (it's then in hashset)

So one of the ans after sorting (-1, 0, 1)

Keep doing that for every element.

T.C ⇒ $O(n^2 \log n)$ S.C ⇒ $O(N) + O(\text{no of unique triplets}) \times 2$

C++ • Autocomplete

```cpp
class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& num) {
        int n = num.size();
    set<vector<int>> st;
    for(int i = 0;i<n;i++){
        set<int> hashset;
        for(int j = i+1;j<n;j++){
            int third = -(num[i] + num[j]);
            if(hashset.find(third) != hashset.end()){
                vector<int> temp = {num[i], num[j], third};
                sort(temp.begin(), temp.end());
                st.insert(temp);
            }
            hashset.insert(num[j]);
        }
    }
    vector<vector<int>> ans(st.begin(), st.end());
    return ans;
    }
};
```

```cpp
#include<bits/stdc++.h>
vector<vector<int>> triplet(int n, vector<int> &num)
{
    set<vector<int>> st;
    for(int i = 0;i<n;i++){
        set<int> hashset;
        for(int j = i+1;j<n;j++){
            int third = -(num[i] + num[j]);
            if(hashset.find(third) != hashset.end()){
                vector<int> temp = {num[i], num[j], third};
                sort(temp.begin(), temp.end());
                st.insert(temp);
            }
            hashset.insert(num[j]);
        }
    }
    vector<vector<int>> ans(st.begin(), st.end());
    return ans;

}
```

Optimal Approach:

arr[] = [-1, 2, 2, -2, -1, 0, -1, 0, 0, -2, 2, -2, 2]

Now you need to get rid of set d.s.

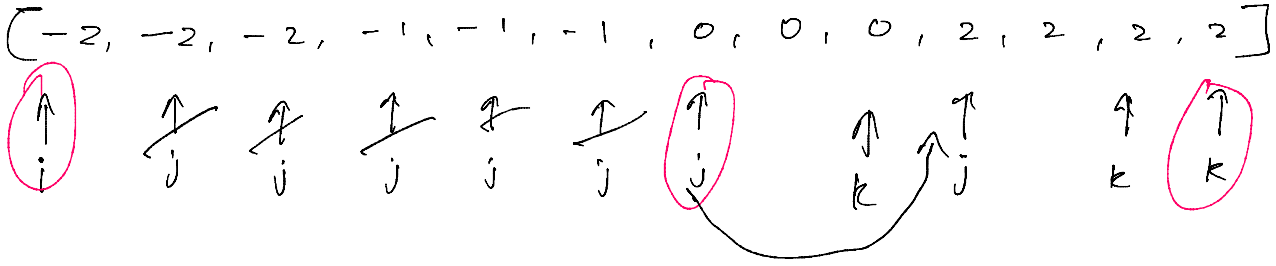
Now you need to get rid of set d.s.

sort it

$[-2, -2, -2, -1, -1, -1, 0, 0, 0, 2, 2, 2, 2]$

Algorithm used here is two pointer approach:

arr[i] + arr[j] + arr[k]



i → will be always constant (at first)

j → (i+1)

k → at last

add up all three element

$-2 - 2 + 2 = -2 < 0$

We need = 0, but here < 0, so you need to increase

so we will do j++,

$-2 - 1 + 2 = -1 < 0$

again same

$-2 + 0 + 2 = 0 == 0$

so when it's $==0$ that one of the triplet.

so store it $\{-2, 0, 2\}$

once it's $==0$ move j++ and k--

↳ until the value is not '0', because the triplet will be same.

once k cross j, you stop as it's no more stoped.

Now again you will move 'i', but the element which is not '-2'.

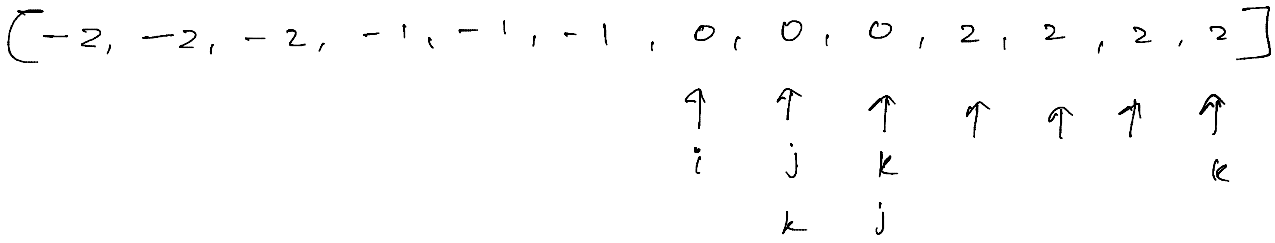
$[-2, -2, -2, -1, -1, -1, 0, 0, 0, 2, 2, 2, 2]$

i j j k j k

$-1 + (-1) + 2 == 0$

So store it $\{-1, -1, 2\}$

$-1 + 0 + 0 = -1 < 0$



$0 + 0 + 2 = 2 > 0$

So k--

You' get another triplet {0,0,0}

```cpp
#include <bits/stdc++.h>
vector<vector<int>> triplet(int n, vector<int> &num) {
  vector<vector<int>> ans;
  sort(num.begin(), num.end());
  for (int i = 0; i < n; i++) {
    if (i > 0 && num[i] == num[i - 1])
      continue;
    int j = i + 1;
    int k = n - 1;
    while (j < k) {
      int sum = num[i] + num[j] + num[k];
      if (sum < 0) {
        j++;
      } else if (sum > 0) {
        k--;
      } else {
        // one of the triplet
        vector<int> temp = {num[i], num[j], num[k]};
        ans.push_back(temp);
        j++;
        k--;
        while (j < k && num[j] == num[j - 1])
          j++;
        while (j < k && num[k] == num[k + 1])
          k--;
      }
    }
  }
  return ans;
}

```

T.C => $(n \log n) + O(N \times N)$

S.C => O (no of unique)