3 $\boxed{\text{3 Sum}}$

return all the unique triplets such that

$a_i + a_j + a_k = 0$

~~& $a_i \neq a_j$~~ $i \neq j \neq k$.

$\boxed{\text{Brute}}$:-

arr[ ] = [-1, 0, 1, 2, -1, -4].

Triplets : [_, _, _].

explore all the different ways to select 3 different index elem.

or arr[ ] = [ ]

or [ ]

or [ ]

$\boxed{\text{pseudocode}}$:-

for unique (store in set)

```
for (i=0; i<n; i++){
    for (j=i+1; j<n; j++)
        for (k=j+1; k<n; k++)
            → sum = 0.
    ← sort triplet
```

T.C of Brute = $O(n^3)$ → So optimize it

Better | ∴ arr[i] + arr[j] + arr[k] = 0.

$O(n^2)$ for loop — we're iterating among diff. values of k.

i = 0 → n-1

j = i+1 → n-1

which is basically −(arr[i] + arr[j])

So, without traversing can we search if −(arr[i] + arr[j]) is present in array or not?

→ Hashing → (but i ≠ j ≠ k)

~~(& unless both arr[i] & arr[j] are 0) arr[k] will~~

Sol^n :- (we'll only put those elem in map such that we never choose same index)

Dry Run:

$$\begin{array}{cccccc} i & j & j & & & \\ \downarrow & \downarrow & \downarrow & & & \\ -1 & 0 & 1 & 2 & -1 & 4 \end{array}$$

−(−1+0) is not present so add arr[j] in map & j++

Approach: fixing i, do j++ all elem betn i & j should be present in mp.

| 0. |

hashing approach :

S1) $i = 0 \rightarrow n-1$

$\hookrightarrow j = i+1 \rightarrow n-1$

$\Downarrow$

& look for third item $= -(arr_i + arr_j)$ in the hashset $\Leftarrow$ { push every value in betn i & j in a hashset.

$\Downarrow$

if present → then.

triplet = [arr_i, arr_j, third item]

↓ sort it

~~Sorted~~

(∴ sorted triplet is stored in a set to store all unique triplets).

↓

then push all values of the set in vector.

syntax :

```
vector <vector <int> >
    ans (st.begin (), st.end ()).
```

T.C ⇒ $O(n^2) \times \log(m)$ ↳ (size of st for third item search)

S.C ⇒ $O($~~size of~~ $n) + O($no of triplets$)$

M3) optimal Approach :-

Aim → To avoid the use of a set :- To store all the triplets

Approach :- {2 pointer + sorting}

approach :-

→ fixing i :-
generate only unique(a_j, a_k).
(i.e skip duplicates of j, k for same i)

→ if same a_i
→ skip the iteration

∵ sorted array :
So whatever triplet we're generating is sorted
+
we can skip duplicate elem (∵ consecutive)

i (const)
↓

DryRun :-

arr[] = [-2, -2, -2, -1, -1, -1, 0, 0, 0, 2, 2, 2, 2]
(sorted)

j j j j j j j ... k k

S3 move till diff

same   same ⇒ So they cross each others

Sum : $-2 + -2 + 2 = -2 < 0$ } inc sum
S1 j++

$-2 - 1 + 2 = -1 < 0$ → j++

S2
$-2 + 0 + 2 = 0$ → generate triplet → Now j++ & k--

for a new triplet ← both've to be differ

(S4) → Now choose next (i)

But make sure its a new a_i

(∵ ~~else~~ we want to avoid checking for triplet ⇒ so new a_i ensures possibility of a new triplet)

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

i i i j j j j k ~~k~~ ~~k~~ ~~k~~ k

j

same as prev a_i (i++ till new a_i is not encountered)

const

sum = -1 -1 +2 = 0

new triplet generated

(j++, k--)

till both are different

why not till one is different

bcz of: $a_k = -(a_i + a_j)$ & not anything else

will not give 0

∵ if $a_i + a_j + a_k = 0$

fixed

if one changes & the other doesn't i.e (2 const in eqn so 3rd var const)

$-1 + 0 + 0 = -1 < 0 \rightsquigarrow$ inc sum

$j++$

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

$$\begin{array}{ccccccc} \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ i & \cancel{j} & \cancel{k} & \cancel{k} & \cancel{k} & \cancel{k} & \cancel{k} \\ & k & j & & & & \end{array}$$

$0 + 0 + 2 > 0 \rightarrow$ dec sum $\rightarrow k--$

$0 + 0 + 0 = 0 \rightarrow$ Triplet generated

$\downarrow$

$j++, k--$