3 Sum

BRUTE FORCE :

Use 3 loops to find all possible combinations and wherever sum is equal to target push that into ans. For removing duplicates make use of set.

- Time Complexity : O(N^3)
- Space Complexity : O(2*unique triplets).

BETTER APPROACH :

Using two loops(i and j iterating variable) for getting two elements. Calculating sum using sum=arr[i]+arr[j] so now the element that is needed to complete the sum is 0-sum. Now as we keep traversing in the inner loop we maintain a set where we search whether we have seen the element before between i and jth position if yes then it would be stored in the set. The basic idea is to fix the ith position and iterate j from i+1 to n we keep storing arr[j] in set and keep searching for target-sum in the set. If found we push the triplet i.e {arr[i],arr[j],target-sum} into a new set which maintains unique triplets.

Finally pushing the set entries into answer vector.

```
class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& arr) {
    set<vector<int>> s;
    int n=arr.size();
    for(int i=0;i<n;i++)
    {
      set<int> hash;
      for(int j=i+1;j<n;j++)
      {
        int sum=arr[i]+arr[j];
        int needed=0-sum;
        if(hash.find(needed)!=hash.end())
        {
          vector<int> temp={arr[i],arr[j],needed};
          sort(temp.begin(),temp.end());
          s.insert(temp);
        }
        hash.insert(arr[j]);
      }
    }
```

```
    vector<vector<int>> ans(s.begin(),s.end());
    return ans;
    }

};
```

- Time Complexity : O(N^2 logN) if using unordered set then O(N^2)
- Space Complexity : O(2* unique triplets)

Optimal Approach :

Sort the array. We use enhanced 2 pointer approach where we fix ith position and then run a two pointer approach on the rest part of the array. For duplicate elements as the array is sorted duplicate elements will occur side by side. so if duplicate elements appear we will skip these elements.

```
class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& arr) {
    vector<vector<int>> ans;
    int n=arr.size();
    sort(arr.begin(),arr.end());
    for(int i=0;i<n-2;i++)
    {
      if(i>0 && arr[i]==arr[i-1]) continue;
      int j=i+1,k=n-1,sum=0-arr[i];
      while(j<k)
      {
        if(arr[j]+arr[k]==sum)
        {
          vector<int> temp={arr[i],arr[j],arr[k]};
          ans.push_back(temp);
          j++;k--;
          while(j<k && arr[j]==arr[j-1]) j++;
          while(j<k && arr[k]==arr[k+1]) k--;
        }
        else if(arr[j]+arr[k]<sum)
          j++;
        else
          k--;
      }

    }
    return ans;
    }

};
```

- Time Complexity : O(N^2)
- Space Complexity : O(unique triplets) used for output only otherwise O(1).