4 Sum

BRUTE FORCE :

Run 4 loops to traverse each element and if it equals sum store into ans. One thing to take care of is that it should not contain duplicates so for this purpose we make use of set but in a set we also need to keep the elements in a specific order to check for duplicates for ex 1 2 3 4 and 3 1 2 4 are both same quads but will be treated distinct by set thus we sort these quads and then store in a set to remove duplicates.

At last we create our ans vector and copy all the elements of the set.

```
class Solution {
public:
    vector<vector<int>> fourSum(vector<int>& nums, int target) {
       set<vector<int>> s;
       int n=nums.size();
       for(int i=0;i<n;i++)
       {
           for(int j=i+1;j<n;j++)
           {
               for(int k=j+1;k<n;k++)
               {
                   for(int l=k+1;l<n;l++)
                   {
                       long long sum=nums[i]+nums[j];
                       sum+=nums[k];
                       sum+=nums[l];
                       if(sum==target)
                       {
                           vector<int> temp={nums[i],nums[j],nums[k],nums[l]};
                           sort(temp.begin(),temp.end());
                           s.insert(temp);
                       }
                   }
               }
           }
       }
       vector<vector<int>> ans(s.begin(),s.end());
       return ans;
    }
};
```

- Time Complexity : O(N^4)

- Space Complexity : O(2 * no. of the quadruplets) as we are using a set data structure and a list to store the quads.

Better Approach :

We will use 3loops now instead of 4 and use a hashset to find the fourth element. We need to find all unique combination so we need the elem at i,j,k but to find 4th element we check in the hashset whether the elem is present or not if it is not present then we add nums[k] into the ahshset and then move k in this fashion numbers between j and k will be stored in hashset uniquely and we can find a combination that gives the target without requiring 4th loop here j and k works as window start and end to find a variable.

before k loop hashset must be defined because as j moves a new window will be formed.

```
class Solution {
public:
    vector<vector<int>> fourSum(vector<int>& nums, int target) {
       set<vector<int>> s;

       int n=nums.size();
       for(int i=0;i<n;i++)
       {
           for(int j=i+1;j<n;j++)
           {
               set<long long> hash;
               for(int k=j+1;k<n;k++)
               {

                       long long sum=nums[i]+nums[j];
                       sum+=nums[k];
                       long long remains=target-sum;
                       if(hash.find(remains)!=hash.end())
                       {
                           vector<int> temp={nums[i],nums[j],nums[k],(int)remains};
                           sort(temp.begin(),temp.end());
                           s.insert(temp);
                       }
                       hash.insert(nums[k]);

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

- Time Complexity : O(N^3 log (M)) M is unique elements stored in hashset.
- Space Complexity : O(2 * no. of the quadruplets)+O(N)

Reason:

we are using a set data structure and a list to store the quads. This results in the first term. And the second space is taken by the set data structure we are using to store the array elements. At most, the set can contain approximately all the array elements and so the space complexity is O(N).

Optimal Approach :

We now use two pointer approach initially we sort the array and keep i and j positions as fixed then we use two pointer to keep track of sum if sum is equal then we include it in ans and if it is less we move k pointer and if it is greater we shift l pointer to left. To deal with duplicate values we will make a comparison inside each loop whenever i>0 this means i is not at 0h position we need to check whether the previous elem that i was pointing to is the same element value that i is currently pointing to if yes then we continue the loop until i find a new value same goes for j but j>i+1 now for k and l when we have found sum==target then we need to shift both k and l k++ and l— but what if they both again point to the same element value as previous for this we need to add a while loop for k and l as well to continue incrementing and decrementing value of k and l respectively until a new value is found also k<l should always be satisfied during these iterations.

```
class Solution {
public:
    vector<vector<int>> fourSum(vector<int>& nums, int target) {
       sort(nums.begin(),nums.end());
       vector<vector<int>> ans;
       int n=nums.size();
       for(int i=0;i<n;i++)
       {
           if(i>0 && nums[i]==nums[i-1]) continue;
           for(int j=i+1;j<n;j++)
           {
               if(j>i+1 && nums[j]==nums[j-1]) continue;
               int k=j+1;
               int l=n-1;
               while(k<l)
               {
                   long long sum=nums[i]+nums[j];
```

```
                    sum+=nums[k];
                    sum+=nums[l];
                    if(sum==target)
                    {
                        vector<int> temp={nums[i],nums[j],nums[k],nums[l]};
                        ans.push_back(temp);
                        k++;l--;
                        while(k<l && nums[k]==nums[k-1]) k++;
                        while(k<l && nums[l]==nums[l+1]) l--;
                    }
                    else if(sum<target)
                         k++;
                    else
                         l--;
                }

            }
        }
        return ans;
    }
};
```

- Time Complexity : O(N^3) + O(NlogN)

Each of the pointers i and j, is running for approximately N times. And both the pointers k and l combined can run for approximately N times including the operation of skipping duplicates. So the total time complexity will be O(N3). NlogN is for sorting the array.

- Space Complexity : O(no. of quadruplets), ***This space is only used to store the answer. We are not using any extra space to solve this problem.*** So, from that perspective, space complexity can be written as O(1).