Four Sum

Problem Statement: Given an array of N integers, your task is to find unique quads that add up to give a target value. In short, you need to return *an array of all the unique quadruplets* [arr[a], arr[b], arr[c], arr[d]] such that their sum is equal to a given *target*.

Pre-req: Binary Search and 2-sum problem

Note:

- 0 <= a, b, c, d < n
- a, b, c, and d are distinct.
- arr[a] + arr[b] + arr[c] + arr[d] == target

Example 1:

```
Input Format: arr[] = [1,0,-1,0,-2,2], target = 0



Result: [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]



Explanation: We have to find unique quadruplets from

the array such that the sum of those elements is

equal to the target sum given that is 0.

The result obtained is such that the sum of the

quadruplets yields 0.
```

Example 2:

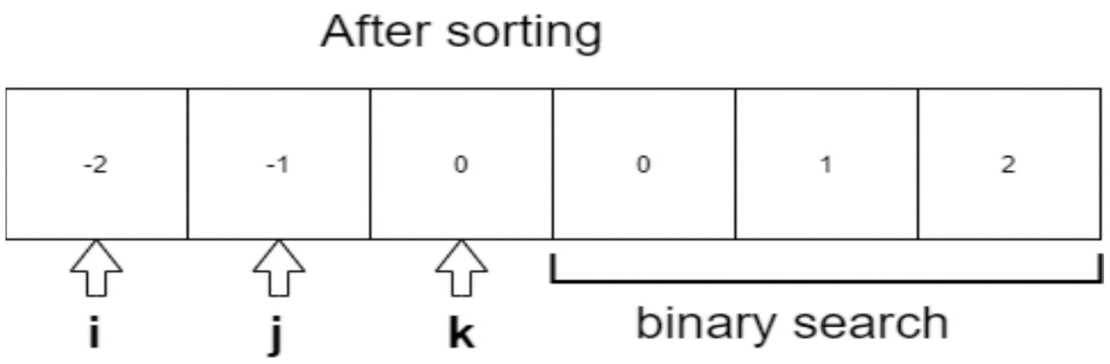
```
Input Format: arr[] = [4,3,3,4,4,2,1,2,1,1], target = 9



Result: [[1,1,3,4],[1,2,2,4],[1,2,3,3]]
```

Solution:

***Disclaimer**: Don't jump directly to the solution, try it out yourself first.*

Solution 1: Using 3 pointers and Binary Search

Intuition:



Approach:

The main idea is to sort the array, and then we can think of searching in the array using the binary search technique. Since we need the 4 numbers which sum up to target. So we will fix three numbers as **nums[i], nums[j] and nums[k]**, and search for the remaining **(target – (nums[i] + nums[j] + nums[k]))** in the array. Since we sorted the array during the start, we can apply **binary search** to search for this value, and if it occurs we can store them. In order to get the unique quads, we use a set data structure to store them.

Time Complexity: O(**N log N + N^3 logN**)

Reason: Sorting the array takes **N log N** and three nested loops will be taking **N^3** and for binary search, it takes another log N.

Space Complexity: O(M * 4), where M is the number of quads

```
public List<List<Integer>> fourSum(int[] nums, int target) {
            HashSet<List<Integer>> contain=new HashSet();
        Arrays.sort(nums);
        for(int i=0;i<nums.length;i++)
```

```
{
    for(int j=i+1;j<nums.length;j++)
    {
        for(int k=j+1;k<nums.length;k++)
        {
            long sum=target-((long)nums[i]+(long)nums[j]+(long)nums[k]);
            int l=k+1;
            int r=nums.length-1;
            while(l<=r)
            {
                int mid=(l+r)/2;
                if((long)nums[mid]==sum)
                {
                    List<Integer> temp=new ArrayList();
                    temp.add(nums[i]);
                     temp.add(nums[j]);
                     temp.add(nums[k]);
                     temp.add(nums[mid]);
                    contain.add(temp);
                    break;
                }
                else if((long)nums[mid]<sum)
                {
                    l=mid+1;
                }
                else
                {
                    r=mid-1;
                }
            }
        }
    }
}
return new ArrayList(contain);
}
```

Solution 2: Optimized Approach

Intuition: In the previous approach we fixed three-pointers and did a binary search to find the fourth. Over here we will fix two-pointers and then find the remaining two elements using two pointer technique as the array will be sorted at first.

Approach: Sort the array, and then fix two pointers, so the remaining sum will be **(target – (nums[i] + nums[j]))**. Now we can fix two pointers, one front, and another back, and easily use a two-pointer to find the remaining two numbers of the quad. In order to avoid duplications, we jump the duplicates, because taking duplicates will result in repeating quads. We can easily jump duplicates, by skipping the same elements by running a loop.
Dry-run: In case you want to see the dry run of this approach, please watch the video at the bottom.

Time Complexity: O(n^3)
Reason: There are 2 nested loops and the front pointer as well as the right pointer (Third nested loop)

Space Complexity: O(1), (Generally the space complexity that is used to return the answer is ignored)

```
public List<List<Integer>> fourSum(int[] nums, int target) {

       List<List<Integer>> res =new ArrayList();
        if(nums==null||nums.length==0)
            return res;
        Arrays.sort(nums);
        for(int i=0;i<nums.length;i++)
        {
            if(i>0&&nums[i-1]==nums[i])
                continue;
            for(int j=i+1;j<nums.length;j++)
            {
                if(j>i+1&&nums[j-1]==nums[j])
                    continue;
                int l=j+1;
                int r=nums.length-1;
                long sum=target-((long)nums[i]+(long)nums[j]);
                while(l<r)
                {
                    long curr= nums[l]+nums[r];
                    if(curr==sum)
                    {
                        List<Integer> list=new ArrayList();
                        list.add(nums[i]);
                        list.add(nums[j]);
                list.add(nums[l]);
                list.add(nums[r]);
                res.add(list);
               while(l<r&&nums[l]==list.get(2))l++;
                while(l<r&&nums[r]==list.get(3))r--;
            }
            else if(curr>sum)
            {
                r--;
            }
            else
            {
                l++;
            }
        }
    }
}
return res;
}
```