4Sum

Given an array S of n integers, are there elements a, b, c, and d in S such that $a + b + c + d =$ target? Find all unique quadruplets in the array which gives the sum of target.

Note:

- Elements in a quadruplet (a,b,c,d) must be in non-descending order. (ie, $a \le b \le c \le d$)
- The solution set must not contain duplicate quadruplets.

```
For example, given array S = {1 0 -1 0 -2 2}, and target = 0.

A solution set is:
(-1,  0, 0, 1)
(-2, -1, 1, 2)
(-2,  0, 0, 2)
```

Solution 1

For the reference, please have a look at my explanation of `3Sum` problem because the algorithm are exactly the same. The link is as blow.

My 3Sum problem answer

The key idea is to downgrade the problem to a `2Sum` problem eventually. And the same algorithm can be expand to `NSum` problem.

After you had a look at my explanation of `3Sum`, the code below will be extremely easy to understand.

```
class Solution {
public:
    vector<vector<int> > fourSum(vector<int> &num, int target) {

        vector<vector<int> > res;

        if (num.empty())
            return res;

        std::sort(num.begin(),num.end());

        for (int i = 0; i < num.size(); i++) {

            int target_3 = target - num[i];

            for (int j = i + 1; j < num.size(); j++) {

                int target_2 = target_3 - num[j];

                int front = j + 1;
                int back = num.size() - 1;

                while(front < back) {

                    int two_sum = num[front] + num[back];

                    if (two_sum < target_2) front++;

                    else if (two_sum > target_2) back--;

                    else {

                        vector<int> quadruplet(4, 0);
                        quadruplet[0] = num[i];
                        quadruplet[1] = num[j];
                        quadruplet[2] = num[front];
                        quadruplet[3] = num[back];
                        res.push_back(quadruplet);

                        // Processing the duplicates of number 3
                        while (front < back && num[front] == quadruplet[2]) ++fro
nt;
```

```
                        // Processing the duplicates of number 4
                        while (front < back && num[back] == quadruplet[3]) --back
;

                    }
                }

                // Processing the duplicates of number 2
                while(j + 1 < num.size() && num[j + 1] == num[j]) ++j;
            }

            // Processing the duplicates of number 1
            while (i + 1 < num.size() && num[i + 1] == num[i]) ++i;

        }

        return res;

    }
};
```

written by kun3 original link here

Solution 2

```
class Solution {
public:
    vector<vector<int>> fourSum(vector<int>& nums, int target) {
        vector<vector<int>> total;
        int n = nums.size();
        if(n<4)  return total;
        sort(nums.begin(),nums.end());
        for(int i=0;i<n-3;i++)
        {
            if(i>0&&nums[i]==nums[i-1]) continue;
            if(nums[i]+nums[i+1]+nums[i+2]+nums[i+3]>target) break;
            if(nums[i]+nums[n-3]+nums[n-2]+nums[n-1]<target) continue;
            for(int j=i+1;j<n-2;j++)
            {
                if(j>i+1&&nums[j]==nums[j-1]) continue;
                if(nums[i]+nums[j]+nums[j+1]+nums[j+2]>target) break;
                if(nums[i]+nums[j]+nums[n-2]+nums[n-1]<target) continue;
                int left=j+1,right=n-1;
                while(left<right){
                    int sum=nums[left]+nums[right]+nums[i]+nums[j];
                    if(sum<target) left++;
                    else if(sum>target) right--;
                    else{
                        total.push_back(vector<int>{nums[i],nums[j],nums[left],nu
ms[right]});
                        do{left++;}while(nums[left]==nums[left-1]&&left<right);
                        do{right--;}while(nums[right]==nums[right+1]&&left<right)
;
                    }
                }
            }
        }
        return total;
    }
};
```

written by cx1992 original link here

Solution 3

```
public class Solution {
    public List<List<Integer>> fourSum(int[] num, int target) {
        ArrayList<List<Integer>> ans = new ArrayList<>();
        if(num.length<4)return ans;
        Arrays.sort(num);
        for(int i=0; i<num.length-3; i++){
            if(i>0&&num[i]==num[i-1])continue;
            for(int j=i+1; j<num.length-2; j++){
                if(j>i+1&&num[j]==num[j-1])continue;
                int low=j+1, high=num.length-1;
                while(low<high){
                    int sum=num[i]+num[j]+num[low]+num[high];
                    if(sum==target){
                        ans.add(Arrays.asList(num[i], num[j], num[low], num[high]
));
                        while(low<high&&num[low]==num[low+1])low++;
                        while(low<high&&num[high]==num[high-1])high--;
                        low++;
                        high--;
                    }
                    else if(sum<target)low++;
                    else high--;
                }
            }
        }
        return ans;
    }
}
```

written by casualhero original link here