3.Hard\04.4_sum.cpp

```cpp
/*
QUESTION:
Given an array nums of n integers, return an array of all the unique quadruplets [nums[a], nums[b], nums[c], nums[d]] such that:
- 0 <= a, b, c, d < n
- a, b, c, and d are distinct.
- nums[a] + nums[b] + nums[c] + nums[d] == target

Example:
Input: nums = [1,0,-1,0,-2,2], target = 0
Output: [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]

APPROACH:
To find the unique quadruplets that sum up to the target, we can use a similar approach as the threeSum problem. We will fix two elements (nums[a] and nums[b]) and use two pointers to find the remaining two elements (nums[c] and nums[d]) that sum up to the target.

1. Sort the input array nums in ascending order.
2. Iterate through the array with two pointers: a and b.
3. For each pair of elements nums[a] and nums[b], use two pointers c and d to find the remaining two elements that sum up to the target.
   - Initialize c as b + 1 and d as the last index of the array.
   - Calculate the target sum as trgt = target - (nums[a] + nums[b]).
   - While c < d, compare the sum of nums[c] and nums[d] with the target sum.
     - If the sum is equal to the target sum, we found a quadruplet. Add it to the answer and move the pointers c and d.
       - Important: Skip any duplicate elements while moving c and d.
     - If the sum is greater than the target sum, decrement d.
     - If the sum is less than the target sum, increment c.
4. Skip any duplicate elements for pointers a and b to avoid duplicate quadruplets.
5. Return the answer array containing unique quadruplets.


CODE:
*/

vector<vector<int>> fourSum(vector<int> &nums, int target)
{
    vector<vector<int>> ans;
    long long trgt = (long long)(target); // to handle overflow
    sort(nums.begin(), nums.end());

    for (int a = 0; a < nums.size(); a++)
    {
        for (int b = a + 1; b < nums.size(); b++)
        {
            if (a == b)
                continue;

            int c = b + 1;
            int d = nums.size() - 1;
            long long tar = trgt - (nums[a] + nums[b]);
```

```cpp

            while (c < d)
            {
                long long sum = nums[c] + nums[d];

                if (sum == tar)
                {
                    ans.push_back({nums[a], nums[b], nums[c], nums[d]});
                    c++;
                    d--;

                    // Skip duplicate elements
                    while (c < d && nums[c] == nums[c - 1])
                        c++;
                    while (c < d && nums[d] == nums[d + 1])
                        d--;
                }
                else if (sum > tar)
                {
                    d--;
                }
                else
                {
                    c++;
                }
            }

            // Skip duplicate elements
            while (b + 1 < nums.size() && nums[b + 1] == nums[b])
                b++;
        }

        // Skip duplicate elements
        while (a + 1 < nums.size() && nums[a + 1] == nums[a])
            a++;
    }

    return ans;
}

/*
TIME COMPLEXITY: O(n^3), where n is the size of the input array nums.
SPACE COMPLEXITY: O(1), as we are using a constant amount of extra space.
*/

```