Description

16. 3Sum Closest

Given an array nums of n integers and an integer target, find three integers in nums such that the sum is closest to target. Return the sum of the three integers. You may assume that each input would have exactly one solution.

```
Example:
Given array nums = [-1, 2, 1, -4], and target = 1.
The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).
```

Idea

My primitive idea is to use a 3 layer (i,j,k) nested loop to enumerate all possible combination triplets to find the sum that is closest to the target, which will take O(n^3) time. How do we speed up? We can still enumerate the outter loop i, so this problem is reduced to a two sum closest problem where we need to find nums[j] + nums[k] closest to target - nums[i]. If it is an two sum equal problem, we can use a map to find nums[j] + nums[k] == target - nums[i]. But it is a closest problem so we can sort first then use two approaching pointers with one scan. This way we use O(nlogn) to sort, O(n) time to enumeate outter loop i, and O(n) time for the two approaching pointers in the inner loop, which makes the algorithm O(n^2). For the inner loop. We are looking for num[j] + nums[k] closest to target - nums[i]. Since it is sorted, j starts from left most side, and k starts from right most side. Then if nums[j] + nums[k] > target - nums[i], we need to decrease the sum so k move to left, on the contrary, if num[j] + nums[k] < target - nums[i], we need to increase the sum so j move to right, otherwise if it equal, we can just return directly because the absolut differece is zero.

Java

```
class Solution {
    public int threeSumClosest(int[] nums, int target) {
```

```
        Arrays.sort(nums);
        int res = nums[0] + nums[1] + nums[2];
        for (int i = 0; i < nums.length; i++) {
            int j = i + 1;
            int k = nums.length - 1;
            while (j < k) {
                int sum = nums[i] + nums[j] + nums[k];
                if (Math.abs(res - target) > Math.abs(sum - target)) {
                    res = sum;
                }
                if (sum == target) {
                    return sum;
                } else if (sum > target) {
                    k--;
                } else {
                    j++;
                }
            }
        }
        return res;
    }
}
```

C++

```
class Solution {
public:
    int threeSumClosest(vector<int>& nums, int target) {
        sort(nums.begin(), nums.end());
        int res = nums[0] + nums[1] + nums[2];
        for (int i = 0; i < nums.size(); i++) {
            int j = i + 1;
            int k = nums.size() - 1;
            while (j < k) {
                int sum = nums[i] + nums[j] + nums[k];
                if (abs(res - target) > abs(sum - target)) {
                    res = sum;
                }
                if (sum == target) {
                    return sum;
                } else if (sum > target) {
                    k--;
                } else {
                    j++;
                }
            }
        }
        return res;
```

```
    }
};
```

Summary

- O(n^3) brute force may be speed up by O(nlogn) sorting then achieve O(n^2) time complexity.
- Fix outter loop and optimize inner loop.

3Sum

This is actually the same idea of 3Sum Closest. After sorting, we fix the outter loop and use two approaching pointers in the inner loop, if we find equal, we move both approaching pointers j and k, if sum is less than 0, we increase sum by moving smaller number j to right, if sum is larger than 0, we decrese sum by moving larger number k to left. If one triplet is found, we need to deduplicate by skipping the same numbers.

Java

```java
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> res = new ArrayList<>();
        Arrays.sort(nums);

        int i = 0;
        while (i < nums.length) {
            int j = i + 1;
            int k = nums.length - 1;
            int target = -nums[i];
            while (j < k) {
                int sum = nums[j] + nums[k];
                if (sum == target) {
                    res.add(Arrays.asList(nums[i], nums[j], nums[k]));
                    do { // deduplication
                        j++;
                    } while (j < k && nums[j] == nums[j - 1]);
                    do { // deduplication
                        k--;
                    } while (j < k && nums[k] == nums[k + 1]);
                } else if (sum < target) {
                    j++;
```

```
                } else {
                    k--;
                }
            }
            do { // deduplication
                i++;
            } while (i < nums.length && nums[i] == nums[i - 1]);
        }

        return res;
    }
}
```

C++

```
class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& nums) {
        vector<vector<int>> res;
        sort(nums.begin(), nums.end());

        int i = 0;
        while (i < nums.size()) {
            int j = i + 1;
            int k = nums.size() - 1;
            int target = -nums[i];
            while (j < k) {
                int sum = nums[j] + nums[k];
                if (sum == target) {
                    res.push_back({nums[i], nums[j], nums[k]});
                    do {
                        j++;
                    } while (j < k && nums[j] == nums[j - 1]);
                    do {
                        k--;
                    } while (j < k && nums[k] == nums[k + 1]);
                } else if (sum < target) {
                    j++;
                } else {
                    k--;
                }
            }
            do {
                i++;
            } while (i < nums.size() && nums[i] == nums[i - 1]);
        }

        return res;
```

```
    }
};
```