

# 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 outer 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  $O(n \log n)$  to sort,  $O(n)$  time to enumerate outer 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 absolute difference 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  $O(n \log n)$  sorting then achieve  $O(n^2)$  time complexity.
- Fix outer loop and optimize inner loop.