3Sum Closest

Given an array *S* of *n* integers, find three integers in *S* such that the sum is closest to a given number, target. Return the sum of the three integers. You may assume that each input would have exactly one solution.

```
For example, given array S = {-1 2 1 -4}, and target = 1.

The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).
```

Solution 1

Here is a solution in Order(N^2). I got help from this post on stackoverflow
Can we improve this time complexity ?

```
int threeSumClosest(vector<int> &num, int target) {
    vector<int> v(num.begin(), num.end()); // I didn't wanted to disturb original
array.
    int n = 0;
    int ans = 0;
    int sum;

    sort(v.begin(), v.end());

    // If less then 3 elements then return their sum
    while (v.size() <= 3) {
        return accumulate(v.begin(), v.end(), 0);
    }

    n = v.size();

    /* v[0] v[1] v[2] ... v[i] .... v[j] ... v[k] ... v[n-2] v[n-1]
     *                    v[i]  <=  v[j]  <= v[k] always, because we sorted our a
rray.
     * Now, for each number, v[i] : we look for pairs v[j] & v[k] such that
     * absolute value of (target - (v[i] + v[j] + v[k]) is minimised.
     * if the sum of the triplet is greater then the target it implies
     * we need to reduce our sum, so we do K = K - 1, that is we reduce
     * our sum by taking a smaller number.
     * Simillarly if sum of the triplet is less then the target then we
     * increase out sum by taking a larger number, i.e. J = J + 1.
     */
    ans = v[0] + v[1] + v[2];
    for (int i = 0; i < n-2; i++) {
        int j = i + 1;
        int k = n - 1;
        while (j < k) {
            sum = v[i] + v[j] + v[k];
            if (abs(target - ans) > abs(target - sum)) {
                ans = sum;
                if (ans == target) return ans;
            }
            (sum > target) ? k-- : j++;
        }
    }
    return ans;
}
```

Edit: Thanks @thr for pointing out that. I have corrected it and also renamed 'mx' by 'ans'.

written by vaibhavatul47 original link here

Solution 2

Similar to 3 Sum problem, use 3 pointers to point current element, next element and the last element. If the sum is less than target, it means we have to add a larger element so next element move to the next. If the sum is greater, it means we have to add a smaller element so last element move to the second last element. Keep doing this until the end. Each time compare the difference between sum and target, if it is less than minimum difference so far, then replace result with it, otherwise keep iterating.

```
public class Solution {
    public int threeSumClosest(int[] num, int target) {
        int result = num[0] + num[1] + num[num.length - 1];
        Arrays.sort(num);
        for (int i = 0; i < num.length - 2; i++) {
            int start = i + 1, end = num.length - 1;
            while (start < end) {
                int sum = num[i] + num[start] + num[end];
                if (sum > target) {
                    end--;
                } else {
                    start++;
                }
                if (Math.abs(sum - target) < Math.abs(result - target)) {
                    result = sum;
                }
            }
        }
        return result;
    }
}
```

written by chase1991 original link here

Solution 3

Sort the vector and then no need to run*O(N^3)* algorithm as each index has a direction to move.

The code starts from this formation.

```
-------------------------------------------------
^  ^                                              ^
|  |                                              |
|  +- second                                    third
+-first
```

if *nums[first] + nums[second] + nums[third]* is smaller than the*target*, we know we have to increase the sum. so only choice is moving the second index forward.

```
-------------------------------------------------
^    ^                                            ^
|    |                                            |
|    +- second                                  third
+-first
```

if the*sum* is bigger than the*target*, we know that we need to reduce the*sum*. so only choice is moving '*third*' to backward. of course if the*sum* equals to *target*, we can immediately return the *sum*.

```
-------------------------------------------------
^    ^                                          ^
|    |                                          |
|    +- second                                 third
+-first
```

when *second* and *third* cross, the round is done so start next round by moving *'first*' and resetting *second* and *third*.

```
-------------------------------------------------
  ^    ^                                         ^
  |    |                                         |
  |    +- second                                 third
  +-first
```

while doing this, collect the*closest sum* of each stage by calculating and comparing delta. Compare *abs(target-newSum)* and *abs(target-closest)*. At the end of the process the three indexes will eventually be gathered at the end of the array.

```
-------------------------------------------------
                                        ^    ^    ^
                                        |    |    `- third
                                        |    +- second
                                        +-first
```

if no exactly matching *sum* has been found so far, the value in*closest* will be the answer.

```
int threeSumClosest(vector<int>& nums, int target) {
    if(nums.size() < 3) return 0;
    int closest = nums[0]+nums[1]+nums[2];
    sort(nums.begin(), nums.end());
    for(int first = 0 ; first < nums.size()-2 ; ++first) {
        if(first > 0 && nums[first] == nums[first-1]) continue;
        int second = first+1;
        int third = nums.size()-1;
        while(second < third) {
            int curSum = nums[first]+nums[second]+nums[third];
            if(curSum == target) return curSum;
            if(abs(target-curSum)<abs(target-closest)) {
                closest = curSum;
            }
            if(curSum > target) {
                --third;
            } else {
                ++second;
            }
        }
    }
    return closest;
}
```

written by asbear original link here