3Sum

Given an array S of n integers, are there elements a, b, c in S such that $a + b + c = 0$? Find all unique triplets in the array which gives the sum of zero.

Note:

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

```
For example, given array S = {-1 0 1 2 -1 -4},

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

Solution 1

Hi guys!

The idea is to sort an input array and then run through all indices of a possible first element of a triplet. For each possible first element we make a standard bi-directional 2Sum sweep of the remaining part of the array. Also we want to skip equal elements to avoid duplicates in the answer without making a set or smth like that.

```
public List<List<Integer>> threeSum(int[] num) {
    Arrays.sort(num);
    List<List<Integer>> res = new LinkedList<>();
    for (int i = 0; i < num.length-2; i++) {
        if (i == 0 || (i > 0 && num[i] != num[i-1])) {
            int lo = i+1, hi = num.length-1, sum = 0 - num[i];
            while (lo < hi) {
                if (num[lo] + num[hi] == sum) {
                    res.add(Arrays.asList(num[i], num[lo], num[hi]));
                    while (lo < hi && num[lo] == num[lo+1]) lo++;
                    while (lo < hi && num[hi] == num[hi-1]) hi--;
                    lo++; hi--;
                } else if (num[lo] + num[hi] < sum) lo++;
                else hi--;
           }
        }
    }
    return res;
}
```

Have a nice coding!

written by shpolsky original link here

Solution 2

the key idea is the same as the `TwoSum` problem. When we fix the `1st` number, the `2nd` and `3rd` number can be found following the same reasoning as `TwoSum`.

The only difference is that, the `TwoSum` problem of LEETCODE has a unique solution. However, in `ThreeSum`, we have multiple duplicate solutions that can be found. Most of the OLE errors happened here because you could've ended up with a solution with so many duplicates.

The naive solution for the duplicates will be using the STL methods like below :

```
std::sort(res.begin(), res.end());
res.erase(unique(res.begin(), res.end()), res.end());
```

But according to my submissions, this way will cause you double your time consuming almostly.

A better approach is that, to jump over the number which has been scanned, no matter it is part of some solution or not.

If the three numbers formed a solution, we can safely ignore all the duplicates of them.

We can do this to all the three numbers such that we can remove the duplicates.

Here's my AC C++ Code:

```
vector<vector<int> > threeSum(vector<int> &num) {

    vector<vector<int> > res;

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

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

        int target = -num[i];
        int front = i + 1;
        int back = num.size() - 1;

        while (front < back) {

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

            // Finding answer which start from number num[i]
            if (sum < target)
                front++;

            else if (sum > target)
                back--;

            else {
                vector<int> triplet(3, 0);
                triplet[0] = num[i];
                triplet[1] = num[front];
                triplet[2] = num[back];
                res.push_back(triplet);

                // Processing duplicates of Number 2
                // Rolling the front pointer to the next different number forward
s
                while (front < back && num[front] == triplet[1]) front++;

                // Processing duplicates of Number 3
                // Rolling the back pointer to the next different number backward
s
                while (front < back && num[back] == triplet[2]) rear--;
            }

        }

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

    }

    return res;

}
```

written by kun3 original link here

Solution 3

Sort the array, iterate through the list, and use another two pointers to approach the target. Runtime: `7ms`

```
public List<List<Integer>> threeSum(int[] nums) {
    List<List<Integer>> result = new ArrayList<>();
    if(nums == null || nums.length < 3) return result;
    Arrays.sort(nums);

    int len = nums.length;
    for(int i = 0; i < len; i++) {
        if(i > 0 && nums[i] == nums[i - 1]) continue;        // Skip same results
        int target = 0 - nums[i];
        int j = i + 1, k = len - 1;
        while(j < k) {
            if(nums[j] + nums[k] == target) {
                result.add(Arrays.asList(nums[i], nums[j], nums[k]));
                while(j < k && nums[j] == nums[j + 1]) j++;  // Skip same results
                while(j < k && nums[k] == nums[k - 1]) k--;  // Skip same results
                j++; k--;
            } else if(nums[j] + nums[k] < target) {
                j++;
            } else {
                k--;
            }
        }
    }
    return result;
}
```

written by yavinci original link here