Grokking the Coding Interview: Patterns for **Coding Questions**



Pattern: Fast & Slow pointers

LinkedList Cycle (easy) Start of LinkedList Cycle (medium) Happy Number (medium) Middle of the LinkedList (easy) Problem Challenge 1 Challenge 1 Problem Challenge 2 Solution Review: Problem Challenge 2 Problem Challenge 3 Solution Review: Problem Challenge 3

Pattern: Merge Intervals

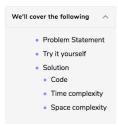


Pattern: Cyclic Sort Introduction

Cyclic Sort (easy)



Triplet Sum to Zero (medium)



Problem Statement

Given an array of unsorted numbers, find all unique triplets in it that add up to zero.

Example 1:

```
Input: [-3, 0, 1, 2, -1, 1, -2]
Output: [-3, 1, 2], [-2, 0, 2], [-2, 1, 1], [-1, 0, 1]
Explanation: There are four unique triplets whose sum is equal to zero.
```

```
Input: [-5, 2, -1, -2, 3]
Output: [[-5, 2, 3], [-2, -1, 3]]
Explanation: There are two unique triplets whose sum is equal to zero.
```

Try solving this question here:



This problem follows the Two Pointers pattern and shares similarities with Pair with Target Sum. A couple of differences are that the input array is not sorted and instead of a pair we need to find triplets with a target

To follow a similar approach, first, we will sort the array and then iterate through it taking one number at a time. Let's say during our iteration we are at number 'X', so we need to find 'Y' and 'Z' such that X+Y+Z==0 . At this stage, our problem translates into finding a pair whose sum is equal to "-X" (as from the above equation Y + Z == -X).

Another difference from Pair with Target Sum is that we need to find all the unique triplets. To handle this, we have to skip any duplicate number. Since we will be sorting the array, so all the duplicate numbers will be next to each other and are easier to skip.

Code

Here is what our algorithm will look like:

```
Python3 G C++
      function search_pair(arr, target_sum, left, triplets) {
  let right = arr.length - 1;
  while (left < right) {</pre>
            const current_sum = arr[left] + arr[right];
if (current_sum === target_sum) { // found the triplet
  triplets.push([-target_sum, arr[left], arr[right]]);
             while (left < right && arr[right] === arr[right + 1]) {</pre>
               else if (target_sum > current_sum) {
      console.log(search_triplets([-3, 0, 1, 2, -1, 1, -2]));
console.log(search_triplets([-5, 2, -1, -2, 3]));
                                                                                                                  SAVE RESET
                                                                                                                                       Close
Output
                                                                                                                                       3.019s
```



```
[[-3, 1, 2], [-2, 0, 2], [-2, 1, 1], [-1, 0, 1]]
[[-5, 2, 3], [-2, -1, 3]]
```

Time complexity

Sorting the array will take O(N*logN). The <code>searchPair()</code> function will take O(N). As we are calling <code>searchPair()</code> for every number in the input array, this means that overall <code>searchTriplets()</code> will take $O(N*logN+N^2)$, which is asymptotically equivalent to $O(N^2)$.

Space complexity

Ignoring the space required for the output array, the space complexity of the above algorithm will be O(N) which is required for sorting.

