

# Grokking the Coding Interview: Patterns for Coding Questions



# Pattern: Fast & Slow pointers



## Pattern: Merge Intervals

MW

**(** 

Explore

寥

Tracks

My Courses

⇔

٥+

Challenge 3



#### Pattern: Cyclic Sort

Challenge 3

Problem Challenge 3

Solution Review: Problem



# Solution Review: Problem Challenge 1



## Quadruple Sum to Target (medium)

Given an array of unsorted numbers and a target number, find all **unique quadruplets** in it, whose **sum is equal to the target number**.

#### Example 1:

```
Input: [4, 1, 2, -1, 1, -3], target=1
Output: [-3, -1, 1, 4], [-3, 1, 1, 2]
Explanation: Both the quadruplets add up to the target.
```

#### Example 2:

```
Input: [2, 0, -1, 1, -2, 2], target=2
Output: [-2, 0, 2, 2], [-1, 0, 1, 2]
Explanation: Both the quadruplets add up to the target.
```

#### Solution

This problem follows the Two Pointers pattern and shares similarities with Triplet Sum to Zero.

We can follow a similar approach to iterate through the array, taking one number at a time. At every step during the iteration, we will search for the quadruplets similar to Triplet Sum to Zero whose sum is equal to the given target.

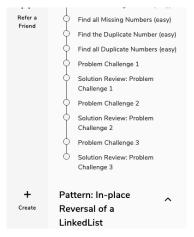
#### Code

Here is what our algorithm will look like:

```
Python3 G C++
       nction search_quadruplets(arr, target) {
       arr.sort((a, b) => a - b)
       const quadruplets = [];
       for (let i = 0; i < arr.length - 3; i++) {</pre>
         if (i > 0 && arr[i] === arr[i - 1]) {
         for (let j = i + 1; j < arr.length - 2; j++) {
           if (j > i + 1 && arr[j] === arr[j - 1]) {
           search_pairs(arr, target, i, j, quadruplets);
       return quadruplets;
     function search_pairs(arr, targetSum, first, second, quadruplets) {
         right = arr.length - 1;
       while ((left < right)) {
        sum = arr[first] + arr[second] + arr[left] + arr[right];
         if (sum === targetSum) -
           quadruplets.push([arr[first], arr[second], arr[left], arr[right]]);
                                                                                                   RESET
                                                                                                             03
                                                                                                        Close
                                                                                                       3.319s
Output
 [[-3, -1, 1, 4], [-3, 1, 1, 2]]
 [[-2, 0, 2, 2], [-1, 0, 1, 2]]
```

## Time complexity

Sorting the array will take O(N\*logN). Overall searchQuadruplets() will take  $O(N*logN+N^3)$ , which is asymptotically equivalent to  $O(N^3)$ .



# Space complexity

The space complexity of the above algorithm will be  $\mathcal{O}(N)$  which is required for sorting.

