Solution Review: Problem Challenge 1

We'll cover the following

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 (https://www.educative.io/collection/page/5668639101419520/5671464854355968/5679549973004288/).

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 (https://www.educative.io/collection/page/5668639101419520/5671464854355968/5679549973004288/) whose sum is equal to the given target.

Code#

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
import java.util.*;

class QuadrupleSumToTarget {

  public static List<List<Integer>> searchQuadruplets(int[] arr, int targe
    Arrays.sort(arr);
    List<List<Integer>> quadruplets = new ArrayList<>();
```

```
    for (int i = 0; i < arr.length - 3; i++) {
      if (i > 0 && arr[i] == arr[i - 1]) // skip same element to avoid dup
        continue;
      for (int j = i + 1; j < arr.length - 2; j++) {
        if (j > i + 1 && arr[j] == arr[j - 1]) // skip same element to avo
          continue;
        searchPairs(arr, target, i, j, quadruplets);
      }
    }
    return quadruplets;
  }

  private static void searchPairs(int[] arr, int targetSum, int first, int
    int left = second + 1;
    int right = arr.length - 1;
    while (left < right) {
      int sum = arr[first] + arr[second] + arr[left] + arr[right];
      if (sum == targetSum) { // found the quadruplet
        quadruplets.add(Arrays.asList(arr[first], arr[second], arr[left],
        left++;
        right--;
```

Run Save Reset

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 $O(N)$ which is required for sorting.

Back

Problem Challenge 1

Next

Problem Challenge 2

Mark as Completed

Report an Issue

Ask a Question
(https://discuss.educative.io/tag/solution-review-problem-challenge-1__pattern-two-pointers__grokking-the-coding-interview-patterns-for-coding-questions?open=true&ctag=grokking-the-coding-interview-patterns-for-coding-questions__design-gurus&aid=5668639101419520&cid=5671464854355968&pid=4556001657225216)