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.

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:

Java | Python3 | C++ | JS

```java
import java.util.*;

class QuadrupleSumToTarget {

  public static List<List<Integer>> searchQuadruplets(int[] arr, int target) {
    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 duplicate quadruplets
        continue;
      for (int j = i + 1; j < arr.length - 2; j++) {
        if (j > i + 1 && arr[j] == arr[j - 1]) // skip same element to avoid duplicate quadruplets
          continue;
        searchPairs(arr, target, i, j, quadruplets);
      }
    }
    return quadruplets;
  }

  private static void searchPairs(int[] arr, int targetSum, int first, int second, List<List<Integer>> qu
    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], arr[right]));
        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

Completed

Report an Issue Ask a Question