MW
Explore
Tracks
My Courses
Edpresso

Solution Review: Problem Challenge 1

We'll cover the following

- Quadruple Sum to Target (medium)
- Solution
  - Code
  - Time complexity
  - Space complexity

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

```
function search_quadruplets(arr, target) {
  arr.sort((a, b) => a - b)
  const quadruplets = [];
  for (let i = 0; i < arr.length - 3; i++) {
    // skip same element to avoid duplicate quadruplets
    if (i > 0 && arr[i] === arr[i - 1]) {
      continue;
    }
    for (let j = i + 1; j < arr.length - 2; j++) {
      // skip same element to avoid duplicate quadruplets
      if (j > i + 1 && arr[j] === arr[j - 1]) {
        continue;
      }
      search_pairs(arr, target, i, j, quadruplets);
    }
  }
  return quadruplets;
}


function search_pairs(arr, targetSum, first, second, quadruplets) {
  let left = second + 1,
    right = arr.length - 1;
  while ((left < right)) {
    sum = arr[first] + arr[second] + arr[left] + arr[right];
    if (sum === targetSum) { // found the quadruplet
      quadruplets.push([arr[first], arr[second], arr[left], arr[right]]);
      left += 1;
```

RUN SAVE RESET

Close

Output 3.319s

```
[ [ -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 $O(N^3)$.

Refer a Friend

+ Create

Pattern: In-place Reversal of a LinkedList

Space complexity

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

☑ MARK AS COMPLETED

← Back
Problem Challenge 1

Next →
Problem Challenge 2

Report an Issue