Triplet Sum Close to Target (medium)

We'll cover the following

- Problem Statement
- Try it yourself
- Solution
  - Code
  - Time complexity
  - Space complexity

Problem Statement

Given an array of unsorted numbers and a target number, find a **triplet in the array whose sum is as close to the target number as possible**, return the sum of the triplet. If there are more than one such triplet, return the sum of the triplet with the smallest sum.

Example 1:

```
Input: [-2, 0, 1, 2], target=2
Output: 1
Explanation: The triplet [-2, 1, 2] has the closest sum to the tar
get.
```

Example 2:

```
Input: [-3, -1, 1, 2], target=1
Output: 0
Explanation: The triplet [-3, 1, 2] has the closest sum to the tar
get.
```

Example 3:

```
Input: [1, 0, 1, 1], target=100
Output: 3
Explanation: The triplet [1, 1, 1] has the closest sum to the targ
et.
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
6       if (min==0){
7         return ans.reduce(reducer);
8       }
9     }
10    return ans.reduce(reducer);
11  };
12  //
13  const helper = (arr,left_value,mid,target_sum,mi
14    let right = arr.length-1;
15    while (mid<right){
16      const curr_value = left_value+arr[mid]+arr[r
17      curr_min = curr_value-target_sum;
18      if (min>Math.abs(curr_min)){
19        min = Math.abs(curr_min);
20        ans = [left_value,arr[mid],arr[right]]
21      }
22      if (curr_min==0){
23        return [min,ans];
24      } else if (curr_min>0){
25        right--;
26      } else{
27        mid++;
28      }
```

```
29    }
30    return [min,ans];
31  }
32
33  const reducer = (accumulator, currentValue) => a
```

Show Results Show Console

2.53s

3 of 3 Tests Passed

Result	Input	Expected Output	Actual Output	Rea
✓	triplet_sum_close_to_target([-2,0,1,2], …	1	1	Succ
✓	triplet_sum_close_to_target([-3,-1,1,2], …	0	0	Succ
✓	triplet_sum_close_to_target([1,0,1,1], 1 …	3	3	Succ

Solution

This problem follows the **Two Pointers** pattern and is quite similar to 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, we will save the difference between the triplet and the target number, so that in the end, we can return the triplet with the closest sum.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
function triplet_sum_close_to_target(arr, target
  arr.sort((a, b) => a - b);
  let smallest_difference = Infinity;
  for (let i = 0; i < arr.length - 2; i++) {
    let left = i + 1,
      right = arr.length - 1;
    while (left < right) {
      const target_diff = targetSum - arr[i] - a
      if (target_diff === 0) { // we've found a
        return targetSum - target_diff; // retur
      }

      if (Math.abs(target_diff) < Math.abs(small
        smallest_difference = target_diff; // sa
      }
      // the second part of the following 'if' i
      if (Math.abs(target_diff) < Math.abs(small
        (Math.abs(target_diff) === Math.abs(smal
        smallest_difference = target_diff; // sa
      }


      if (target_diff > 0) {
        left += 1; // we need a triplet with a b
      } else {
        right -= 1; // we need a triplet with a
      }
    }
```

Time complexity

Sorting the array will take $O(N * logN)$. Overall, the function will take $O(N * logN + N^2)$, which is asymptotically equivalent to $O(N^2)$.

Space complexity

The above algorithm's space complexity will be $O(N)$, which is required for sorting.

Back

Triplet Sum to Zero (medium)

Next

Triplets with Smaller Sum (medium)

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/triplet-sum-close-to-target-medium__pattern-two-pointers__grokking-the-coding-interview-patterns-for-coding-questions)