Triplet Sum Close to Target (medium)

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 target.
```

Example 2:

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

Example 3:

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

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
const triplet_sum_close_to_target = function(arr, target_sum) {
  // TODO: Write your code here
  return -1;
};
```

TEST SAVE RESET

Solution

This problem follows the **Two Pointers** pattern and is quite similar to Triplet Sum to Zero.

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:

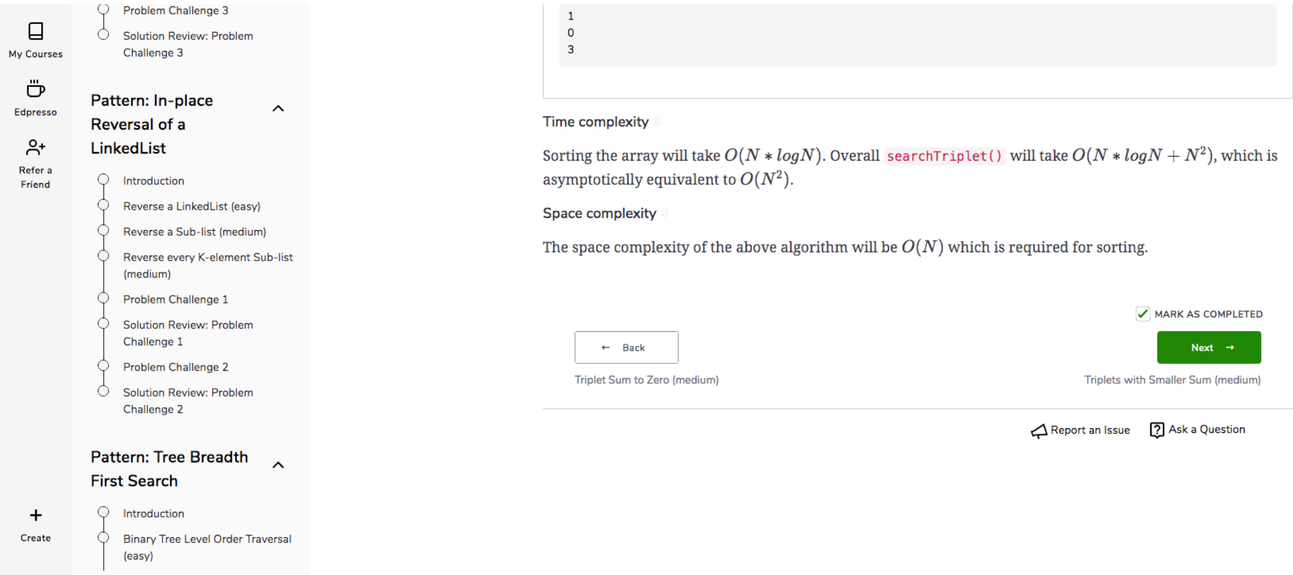
Java | Python3 | C++ | JS

```
       return targetSum - target_diff; // return sum of all the numbers
     }

     if (Math.abs(target_diff) < Math.abs(smallest_difference)) {
       smallest_difference = target_diff; // save the closest difference
     }
     // the second part of the following 'if' is to handle the smallest sum when we have more than one so
     if (Math.abs(target_diff) < Math.abs(smallest_difference) ||
       (Math.abs(target_diff) === Math.abs(smallest_difference) && target_diff > smallest_difference)) {
       smallest_difference = target_diff; // save the closest and the biggest difference
     }


     if (target_diff > 0) {
       left += 1; // we need a triplet with a bigger sum
     } else {
       right -= 1; // we need a triplet with a smaller sum
     }
   }
 }
 return targetSum - smallest_difference;



onsole.log(triplet_sum_close_to_target([-2, 0, 1, 2], 2));
onsole.log(triplet_sum_close_to_target([-3, -1, 1, 2], 1));
onsole.log(triplet_sum_close_to_target([1, 0, 1, 1], 100));
```

RUN SAVE RESET

Close

Output 2.575s



My Courses

Edpresso

Refer a Friend

Create

Problem Challenge 3

Solution Review: Problem Challenge 3

Pattern: In-place Reversal of a LinkedList

Introduction

Reverse a LinkedList (easy)

Reverse a Sub-list (medium)

Reverse every K-element Sub-list (medium)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

Pattern: Tree Breadth First Search

Introduction

Binary Tree Level Order Traversal (easy)

```
1
0
3
```

Time complexity

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

Space complexity

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

☑ MARK AS COMPLETED

← Back

Triplet Sum to Zero (medium)

Next →

Triplets with Smaller Sum (medium)

Report an Issue

Ask a Question