Triplet Sum Close to Target (medium)

We'll cover the following

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:

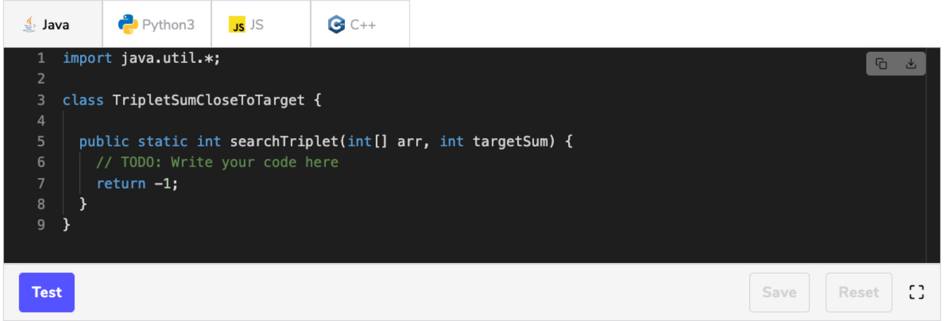
```
Input: [-3, -1, 1, 2], target=1
Output: 0
Explanation: The triplet [-3, 1, 2] has the closest sum to the target.
```

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
import java.util.*;

class TripletSumCloseToTarget {

  public static int searchTriplet(int[] arr, int targetSum) {
    // TODO: Write your code here
    return -1;
  }
}
```

Solution

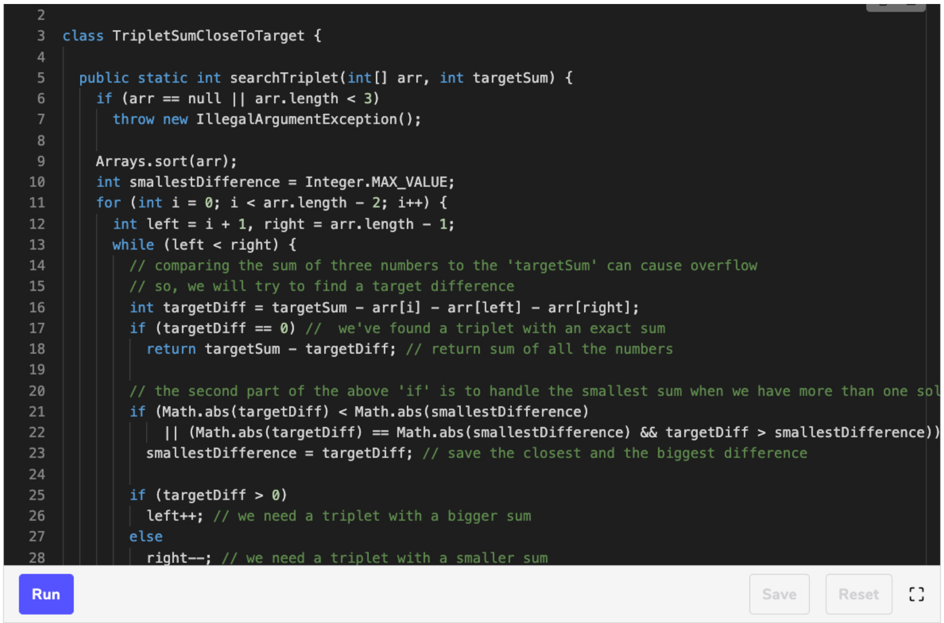
This problem follows the **Two Pointers** pattern and is quite similar to Triplet Sum to Zero.

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
import java.util.*;
```



```
class TripletSumCloseToTarget {

  public static int searchTriplet(int[] arr, int targetSum) {
    if (arr == null || arr.length < 3)
      throw new IllegalArgumentException();

    Arrays.sort(arr);
    int smallestDifference = Integer.MAX_VALUE;
    for (int i = 0; i < arr.length - 2; i++) {
      int left = i + 1, right = arr.length - 1;
      while (left < right) {
        // comparing the sum of three numbers to the 'targetSum' can cause overflow
        // so, we will try to find a target difference
        int targetDiff = targetSum - arr[i] - arr[left] - arr[right];
        if (targetDiff == 0) //  we've found a triplet with an exact sum
          return targetSum - targetDiff; // return sum of all the numbers

        // the second part of the above 'if' is to handle the smallest sum when we have more than one sol
        if (Math.abs(targetDiff) < Math.abs(smallestDifference)
            || (Math.abs(targetDiff) == Math.abs(smallestDifference) && targetDiff > smallestDifference))
          smallestDifference = targetDiff; // save the closest and the biggest difference

        if (targetDiff > 0)
          left++; // we need a triplet with a bigger sum
        else
          right--; // we need a triplet with a smaller sum
```

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)

✓ Completed

Report an Issue Ask a Question