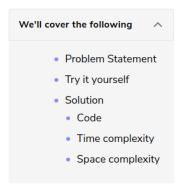




Triplet Sum Close to Target (medium)



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:



Solution |

This problem follows the ${\bf Two\ Pointers}$ pattern and is quite similar to ${\bf Triplet\ Sum\ to\ Zero.}$

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:

```
Python3
👙 Java
                                   ⊙ C++
                                                    JS JS
      import math
      def triplet sum close to target(arr, target sum):
        arr.sort()
         smallest_difference = math.inf
            right = len(arr) - 1
            while (left < right):</pre>
              target_diff = target_sum - arr[i] - arr[left] - arr[right]
if target_diff == 0:  # we've found a triplet with an exact sum
return target_sum - target_diff  # return sum of all the numbers
              if abs(target_diff) < abs(smallest_difference) or (abs(target_diff) == abs(smallest_difference) and
smallest_difference = target_diff # save the closest and the biggest difference</pre>
               if target_diff > 0:
                 right -= 1 # we need a triplet with a smaller sum
         return target_sum - smallest_difference
      def main():
        print(triplet_sum_close_to_target([-2, 0, 1, 2], 2))
print(triplet_sum_close_to_target([-3, -1, 1, 2], 1))
        print(triplet_sum_close_to_target([1, 0, 1, 1], 100))
     main()
Run
                                                                                                                                         Reset []
```

Time complexity

Sorting the array will take O(N * log N). Overall, the function will take $O(N * log N + N^2)$, 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.

