



# 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 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++

```
def triplet_sum_close_to_target(arr, target_sum):  
    # TODO: Write your code here  
    return -1
```



Test

Save

Reset



## Solution#

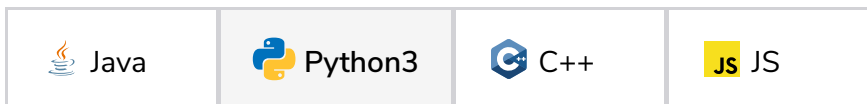
This problem follows the **Two Pointers** pattern and is quite similar to Triplet Sum to Zero

(<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:



```
import math

def triplet_sum_close_to_target(arr, target_sum):
    arr.sort()
    smallest_difference = math.inf
    for i in range(len(arr)-2):
        left = i + 1
        right = len(arr) - 1
        while (left < right):
            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

            # the second part of the following 'if' is to handle the smallest sum when
            if abs(target_diff) < abs(smallest_difference) or (abs(target_diff) == abs(
                smallest_difference = target_diff # save the closest and the biggest di

            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 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()
```

RunSaveReset

## 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.

[← Back](#)[Next →](#)[Triplet Sum to Zero \(medium\)](#)[Triplets with Smaller Sum \(medium\)](#)☒ Mark as Completed[? 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?open=true&ctag=grokking-the-coding-interview-patterns-for-coding-questions\\_\\_design-gurus&aid=5668639101419520&cid=5671464854355968&pid=6210874538721280](https://discuss.educative.io/tag/triplet-sum-close-to-target-medium__pattern-two-pointers__grokking-the-coding-interview-patterns-for-coding-questions?open=true&ctag=grokking-the-coding-interview-patterns-for-coding-questions__design-gurus&aid=5668639101419520&cid=5671464854355968&pid=6210874538721280))

[! Report an Issue](#)