

Triplets with Smaller Sum (medium)

We'll cover the following ^

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

Problem Statement

Given an array `arr` of unsorted numbers and a target sum, **count all triplets** in it such that **`arr[i] + arr[j] + arr[k] < target`** where `i`, `j`, and `k` are three different indices. Write a function to return the count of such triplets.

Example 1:

Input: `[-1, 0, 2, 3]`, `target=3`

Output: 2

Explanation: There are two triplets whose sum is less than the target: `[-1, 0, 3]`, `[-1, 0, 2]`

Example 2:

Input: `[-1, 4, 2, 1, 3]`, `target=5`

Output: 4

Explanation: There are four triplets whose sum is less than the target: `[-1, 1, 4]`, `[-1, 1, 3]`, `[-1, 1, 2]`, `[-1, 2, 3]`

Try it yourself

Try solving this question here:

 Java

 Python3

 JS

 C++

```
1 def triplet_with_smaller_sum(arr, target):
```

```
2     count = 0
```

```
3     # TODO: Write your code here
```



educative

```
4     return count
5
```



Solution

This problem follows the **Two Pointers** pattern and shares similarities with Triplet Sum to Zero (<https://www.educative.io/collection/page/5668639101419520/5671464854355968/5679549973004288/>). The only difference is that, in this problem, we need to find the triplets whose sum is less than the given target. To meet the condition $i \neq j \neq k$ we need to make sure that each number is not used more than once.

Following a similar approach, first, we can sort the array and then iterate through it, taking one number at a time. Let's say during our iteration we are at number 'X', so we need to find 'Y' and 'Z' such that $X + Y + Z < target$. At this stage, our problem translates into finding a pair whose sum is less than " $target - X$ " (as from the above equation $Y + Z == target - X$). We can use a similar approach as discussed in Triplet Sum to Zero (<https://www.educative.io/collection/page/5668639101419520/5671464854355968/5679549973004288/>).

Code

Here is what our algorithm will look like:

Java

Python3

C++

JS

```
3     count = 0
4     for i in range(len(arr)-2):
5         count += search_pair(arr, target - arr[i], i)
6     return count
7
8
9 def search_pair(arr, target_sum, first):
10     count = 0
11     left, right = first + 1, len(arr) - 1
12     while (left < right):
13         if arr[left] + arr[right] < target_sum: # found the triplet
14             # since arr[right] >= arr[left], therefore, we can replace arr[right] by any number
15             # left and right to get a sum less than the target sum
16             count += right - left
17             left += 1
18         else:
19             right -= 1 # we need a pair with a smaller sum
20     return count
21
22
23 def main():
24     print(triplet_with_smaller_sum([-1, 0, 2, 3], 3))
25     print(triplet_with_smaller_sum([-1, 4, 2, 1, 3], 5))
26
27
28 main()
```



Time complexity

Sorting the array will take $O(N * \log N)$. The `searchPair()` will take $O(N)$. So, overall `searchTriplets()` will take $O(N * \log N + N^2)$, 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 if we are not using an in-place sorting algorithm.

Similar Problems

Problem: Write a function to return the list of all such triplets instead of the count. How will the time complexity change in this case?

Solution: Following a similar approach we can create a list containing all the triplets. Here is the code - only the highlighted lines have changed:

Java

Python3

C++

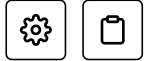
JS JS

```
3 triplets = []
4 for i in range(len(arr)-2):
5     search_pair(arr, target - arr[i], i, triplets)
6 return triplets
7
8
9 def search_pair(arr, target_sum, first, triplets):
10     left = first + 1
11     right = len(arr) - 1
12     while (left < right):
13         if arr[left] + arr[right] < target_sum: # found the triplet
14             # since arr[right] >= arr[left], therefore, we can replace arr[right] by any number
15             # left and right to get a sum less than the target sum
16             for i in range(right, left, -1):
17                 triplets.append([arr[first], arr[left], arr[i]])
18             left += 1
19         else:
20             right -= 1 # we need a pair with a smaller sum
21
22
23 def main():
24     print(triplet_with_smaller_sum([-1, 0, 2, 3], 3))
25     print(triplet_with_smaller_sum([-1, 4, 2, 1, 3], 5))
26
27
28 main()
29
```



Another simpler approach could be to check every triplet of the array with three nested loops and create a list of triplets that meet the required condition.

Time complexity



Sorting the array will take $O(N * \log N)$. The `searchPair()`, in this case, will take $O(N^2)$; the main `while` loop will run in $O(N)$ but the nested `for` loop can also take $O(N)$ - this will happen when the target sum is bigger than every triplet in the array.

So, overall `searchTriplets()` will take $O(N * \log N + N^3)$, which is asymptotically equivalent to $O(N^3)$.

Space complexity

Ignoring the space required for the output array, the space complexity of the above algorithm will be $O(N)$ which is required for sorting.

[< Back](#)[Next >](#)

Triplet Sum Close to Target (medium)

Subarrays with Product Less than a Ta...

☒ Mark as Completed



Report an
Issue



Ask a Question

(https://discuss.educative.io/tag/triplets-with-smaller-sum-medium__pattern-two-pointers__grokking-the-coding-interview-patterns-for-coding-questions)