Triplet Sum (3Sum) CodeStudio

The problem is to find all unique triplets in an array that sum up to a given target value.

Input: 10 5 5 5 2, target = 12

Output: 5 5 2

Input: 1 2 3 1 2 3, target = 6

Output: 1 2 3

Input: 1 2 3 4, target = 4

Output: -1

Approach 1: Using Nested Loops

The function tripletSum takes the input array arr, its size n, and the target value as parameters.

It uses nested loops to iterate over all possible combinations of triplets.

For each triplet, it checks if the sum of the elements is equal to the target value.

If a triplet with the target sum is found, it creates a vector with the triplet elements, sorts it in ascending order, and inserts it into a set to eliminate duplicates.

Finally, it converts the set to a vector and returns the result.

Since it uses nested loops**, the time complexity is O(n^3), where n is the size of the input array.**

It uses a set to store unique triplets, **which can take up to O(n^3) space in the worst case when all possible combinations are unique.**

Approach 2: Using Two Pointer Approach (Optimized)

The function tripletSumOptimized follows an optimized approach using the two-pointer technique.

It first sorts the input array in ascending order.

Then, it iterates over the array using a single loop and maintains two pointers, left and right, initially pointing to the next and last elements of the array, respectively.

Inside the loop, it calculates the current sum of the three elements (the current element and the elements pointed by the two pointers).

If the current sum is equal to the target, it creates a triplet, inserts it into a set to eliminate duplicates, and moves the pointers to the next unique elements.

If the current sum is less than the target, it increments the left pointer to increase the sum.

If the current sum is greater than the target, it decrements the right pointer to decrease the sum.

The loop continues until the left and right pointers cross each other.

Finally, it converts the set to a vector and returns the result.

The array sorting step takes O(n log n) time.

Overall, the time complexity is dominated by the array sorting step, which is O(n log n).

It uses a set to store unique triplets, **which can take up to O(n^2) space in the worst case when all possible combinations are unique.**

Which Approach to Use:

Approach 2, using the two-pointer approach, is more efficient as it reduces the time complexity from O(n^3) to O(n^2) and optimizes the search process.