Pair Sum CodeStudio

Given an array of integers and a target sum, find all pairs of elements in the array that add up to the target sum. Return the pairs in sorted order.

Input: arr = {1, 2, 3, 4, 5}, s = 5

Output: {1, 4}, {2, 3}

Input: arr = {2, -3, 3, 3, -2}, s = 0

Output: {-3, 3}, {-3, 3}, {-2, 2}

Input: arr = {2, 3, 2, 4, 2}, s = 4

Output: {2, 2}, {2, 2}, {2, 2}

Approach 1: Using Nested loop to find pairs.

This approach uses nested loops to iterate through all possible pairs of elements in the array.

If the sum of the current pair is equal to the target sum, the pair is added to the ans vector after sorting it in ascending order.

Time Complexity:

pairSum function: O(N^2 * log(N))

The nested loops iterate through all possible pairs of elements, resulting in O(N^2) iterations.

The sorting step at the end takes O(N log N) time complexity.

Space Complexity:

This approach has **a space complexity of O(N)** as the resulting pairs are stored in a vector.

Approach 2: Using HashMap

utilizes a hash map to efficiently track the frequencies of elements in the array.

It iterates through each element and calculates the complement (target minus the current element) to find the pair that sums up to the target.

If the complement is present in the hash map, the pairs are added to the ans vector, considering the minimum and maximum values.

Finally, the ans vector is sorted in ascending order before returning the result.

Time Complexity:

pairSumHashMap function: O(N log N)

The loop iterating over the array elements takes O(N) time complexity.

The hash map lookup and insertion operations have an average time complexity of O(1) but can go up to O(N) in the worst case.

The sorting step at the end takes O(N log N) time complexity.

Space Complexity:

The pairSumHashMap function additionally uses a **hash map to store the frequencies of elements, which can take up to O(N) space in the worst case.**