Distance Calculation Between Lists

Write code to solve the following programming puzzle:

"You haven't even left yet and the group of Elvish Senior Historians has already hit a problem: their list of locations to check is currently empty. Eventually, someone decides that the best place to check first would be the Chief Historian's office.

Upon pouring into the office, everyone confirms that the Chief Historian is indeed nowhere to be found. Instead, the Elves discover an assortment of notes and lists of historically significant locations! This seems to be the planning the Chief Historian was doing before he left. Perhaps these notes can be used to determine which locations to search?

Throughout the Chief's office, the historically significant locations are listed not by name but by a unique number called the location ID. To make sure they don't miss anything, The Historians split into two groups, each searching the office and trying to create their own complete list of location IDs.

There's just one problem: by holding the two lists up side by side (your puzzle input), it quickly becomes clear that the lists aren't very similar. Maybe you can help The Historians reconcile their lists?

For example:

3   4
4   3
2   5
1   3
3   9
3   3

Maybe the lists are only off by a small amount! To find out, pair up the numbers and measure how far apart they are. Pair up the smallest number in the left list with the smallest number in the right list, then the second-smallest left number with the second-smallest right number, and so on.

Within each pair, figure out how far apart the two numbers are; you'll need to add up all of those distances. For example, if you pair up a 3 from the left list with a 7 from the right list, the distance apart is 4; if you pair up a 9 with a 3, the distance apart is 6.

In the example list above, the pairs and distances would be as follows:

The smallest number in the left list is 1, and the smallest number in the right list is 3. The distance between them is 2.
The second-smallest number in the left list is 2, and the second-smallest number in the right list is another 3. The distance between them is 1.
The third-smallest number in both lists is 3, so the distance between them is 0.

The next numbers to pair up are 3 and 4, a distance of 1.
The fifth-smallest numbers in each list are 3 and 5, a distance of 2.
Finally, the largest number in the left list is 4, while the largest number in the right list is 9; these are a distance 5 apart.

To find the total distance between the left list and the right list, add up the distances between all of the pairs you found. In the example above, this is 2 + 1 + 0 + 1 + 2 + 5, a total distance of 11!

Your actual left and right lists contain many location IDs. What is the total distance between your lists?"

Here's the Python code to solve the puzzle:

Problem Explanation:

1. **Input:** Two lists of integers (left and right).
2. **Task:** Compute the total distance between the lists by:
    - Sorting both lists in ascending order.
    - Pairing the smallest element in one list with the smallest in the other, the second smallest with the second smallest, and so on.
    - Calculating the absolute difference for each pair and summing these differences.

Solution Code:

python

```python
def calculate_total_distance(left_list, right_list):
    # Sort both lists in ascending order
    left_sorted = sorted(left_list)
    right_sorted = sorted(right_list)

    # Calculate the total distance by summing up absolute differences
    total_distance = sum(abs(l - r) for l, r in zip(left_sorted, right_sorted))

    return total_distance

# Example Input
left_list = [3, 4, 2, 1, 3, 3]
right_list = [4, 3, 5, 3, 9, 3]

# Calculate the total distance
result = calculate_total_distance(left_list, right_list)

# Output the result
print("Total distance:", result)
```

Explanation of the Code:

1. **Sorting**:
    - Both lists are sorted to ensure that the smallest numbers are paired together.
2. **Zipping and Pairing**:

    - The `` `zip` `` function pairs corresponding elements from the sorted lists.
3. **Distance Calculation**:
    - For each pair, the absolute difference (`` `abs(l - r)` ``) is calculated and summed using a generator expression in the `` `sum` `` function.
4. **Output**:
    - The function returns the total distance.

Example Walkthrough:

Input:

```plaintext
left_list = [3, 4, 2, 1, 3, 3]
right_list = [4, 3, 5, 3, 9, 3]
```

Steps:

1. Sort the lists:
    - `` `left_sorted = [1, 2, 3, 3, 3, 4]` ``
    - `` `right_sorted = [3, 3, 3, 4, 5, 9]` ``
2. Pair the elements:
    - `` `(1, 3), (2, 3), (3, 3), (3, 4), (3, 5), (4, 9)` ``
3. Calculate distances:
    - `` `|1 - 3| = 2` ``
    - `` `|2 - 3| = 1` ``
    - `` `|3 - 3| = 0` ``
    - `` `|3 - 4| = 1` ``
    - `` `|3 - 5| = 2` ``
    - `` `|4 - 9| = 5` ``
4. Total Distance:
    - `` `2 + 1 + 0 + 1 + 2 + 5 = 11` ``

Output:

```plaintext
Total distance: 11
```

Complexity:

1. **Time Complexity**:
    - Sorting both lists: $O(n \log n)$, where n is the length of the lists.
    - Calculating distances: $O(n)$.
    - Overall: $O(n \log n)$.
2. **Space Complexity**:

  - $O(n)$ due to sorting.

This code will work for any valid input lists of integers.

Now write code to solve part two of the puzzle:

"Your analysis only confirmed what everyone feared: the two lists of location IDs are indeed very different.

Or are they?

The Historians can't agree on which group made the mistakes or how to read most of the Chief's handwriting, but in the commotion you notice an interesting detail: a lot of location IDs appear in both lists! Maybe the other numbers aren't location IDs at all but rather misinterpreted handwriting.

This time, you'll need to figure out exactly how often each number from the left list appears in the right list. Calculate a total similarity score by adding up each number in the left list after multiplying it by the number of times that number appears in the right list.

Here are the same example lists again:

3   4
4   3
2   5
1   3
3   9
3   3

For these example lists, here is the process of finding the similarity score:

The first number in the left list is 3. It appears in the right list three times, so the similarity score increases by 3 * 3 = 9.
The second number in the left list is 4. It appears in the right list once, so the similarity score increases by 4 * 1 = 4.
The third number in the left list is 2. It does not appear in the right list, so the similarity score does not increase (2 * 0 = 0).
The fourth number, 1, also does not appear in the right list.
The fifth number, 3, appears in the right list three times; the similarity score increases by 9.
The last number, 3, appears in the right list three times; the similarity score again increases by 9.

So, for these example lists, the similarity score at the end of this process is 31 (9 + 4 + 0 + 0 + 9 + 9).

Once again consider your left and right lists. What is their similarity score?"

Here’s the Python code to solve **Part Two** of the puzzle:

Solution Code:

python

```python
from collections import Counter

def calculate_similarity_score(left_list, right_list):
    # Count occurrences of each number in the right list
    right_count = Counter(right_list)

    # Calculate the similarity score
    similarity_score = sum(num * right_count[num] for num in left_list)

    return similarity_score

# Example Input
left_list = [3, 4, 2, 1, 3, 3]
right_list = [4, 3, 5, 3, 9, 3]

# Calculate the similarity score
result = calculate_similarity_score(left_list, right_list)

# Output the result
print("Similarity score:", result)
```

Explanation of the Code:

1. **Counting Occurrences**:
    - Use `collections.Counter` to count how often each number appears in the `right_list`.
    - Example: For `right_list = [4, 3, 5, 3, 9, 3]`, the count is:

      python

      ```python
      right_count = {4: 1, 3: 3, 5: 1, 9: 1}
      ```

2. **Calculating the Similarity Score**:
    - Iterate over each number in the `left_list`.
    - Multiply the number by its count from the `right_count` dictionary (defaulting to 0 if the number is not in the dictionary).
    - Accumulate the results into the `similarity_score`.
3. **Output**:
    - Return the final similarity score.

Example Walkthrough:

Input:

plaintext

```plaintext
left_list = [3, 4, 2, 1, 3, 3]
```

```
right_list = [4, 3, 5, 3, 9, 3]
```

Steps:

1. Count occurrences in the `right_list`:

   ```
   python
   
   right_count = {4: 1, 3: 3, 5: 1, 9: 1}
   ```

2. Process each number in the `left_list`:
   - **First number (3)**: Appears 3 times in `right_list`. Add $3 \times 3 = 9$.
   - **Second number (4)**: Appears 1 time. Add $4 \times 1 = 4$.
   - **Third number (2)**: Appears 0 times. Add $2 \times 0 = 0$.
   - **Fourth number (1)**: Appears 0 times. Add $1 \times 0 = 0$.
   - **Fifth number (3)**: Appears 3 times. Add $3 \times 3 = 9$.
   - **Sixth number (3)**: Appears 3 times. Add $3 \times 3 = 9$.
3. Total Similarity Score:

   ```
   plaintext
   
   9 + 4 + 0 + 0 + 9 + 9 = 31
   ```

Output:

```
plaintext

Similarity score: 31
```

Complexity:

1. **Time Complexity**:
   - Counting occurrences in the `right_list`: $O(n)$, where n is the length of the `right_list`.
   - Iterating over the `left_list`: $O(m)$, where m is the length of the `left_list`.
   - Overall: $O(n + m)$.
2. **Space Complexity**:
   - Storage for the `Counter` dictionary: $O(k)$, where k is the number of unique elements in the `right_list`.

This code is efficient and works for any valid input lists of integers.