



This is a copy of a conversation between ChatGPT & Anonymous.

[Report content](#)

Please write a Python script to solve the following problem:

The Chief Historian is always present for the big Christmas sleigh launch, but nobody has seen him in months! Last anyone heard, he was visiting locations that are historically significant to the North Pole; a group of Senior Historians has asked you to accompany them as they check the places they think he was most likely to visit.

As each location is checked, they will mark it on their list with a star. They figure the Chief Historian must be in one of the first fifty places they'll look, so in order to save Christmas, you need to help them get fifty stars on their list before Santa takes off on December 25th.

Collect stars by solving puzzles. Two puzzles will be made available on each day in the Advent calendar; the second puzzle is unlocked when you complete the first. Each puzzle grants one star. Good luck!

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 is a Python script to calculate the total distance between two lists as described in your problem:

python

Copy code

```
def calculate_total_distance(left_list, right_list):
    """
    Calculate the total distance between two lists of numbers
    by pairing the smallest elements and summing their absolute differences.

    Args:
    left_list (list): First list of numbers.
    right_list (list): Second list of numbers.

    Returns:
    int: Total distance between the two lists.
    """
    # Sort both lists
    left_sorted = sorted(left_list)
    right_sorted = sorted(right_list)

    # Calculate the sum of absolute differences between paired elements
```

```

total_distance = sum(abs(a - b) for a, b in zip(left_sorted, right_sorted))

return total_distance

# Example usage
if __name__ == "__main__":
    # Replace these lists with your actual input data
    left_list = [3, 4, 2, 1, 3, 3]
    right_list = [4, 3, 5, 3, 9, 3]

    total_distance = calculate_total_distance(left_list, right_list)
    print(f"Total distance: {total_distance}")

```

## How It Works:

1. **Sorting:** Both input lists are sorted in ascending order.
2. **Pairing:** The elements are paired by their sorted positions, ensuring the smallest values are compared first.
3. **Calculating Differences:** The absolute differences between paired elements are calculated.
4. **Summing Up:** The total distance is obtained by summing all the absolute differences.

## Example Output:

For the input lists:

plaintext

 Copy code

```

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

```

The output will be:

plaintext

 Copy code

```
Total distance: 11
```

You can replace `left_list` and `right_list` with your actual input data to solve the problem.

Please can you add a function to read the input lists from file?



Here's the updated Python script with a function to read the input lists from a file. The file should have two lines: the first line for the left list and the second line for the right list, with numbers separated by spaces.

python

 Copy code

```

def read_lists_from_file(file_path):
    """
    Reads two lists of integers from a file.

```

Args:  
file\_path (str): Path to the input file. The file should contain two lines:  
the first line for the left list, and the second line for the right list.

```
Returns:
tuple: Two lists of integers (left_list, right_list).
"""

with open(file_path, 'r') as file:
    lines = file.readlines()
    # Parse each line into a list of integers
    left_list = list(map(int, lines[0].strip().split()))
    right_list = list(map(int, lines[1].strip().split()))
    return left_list, right_list

def calculate_total_distance(left_list, right_list):
    """
    Calculate the total distance between two lists of numbers
    by pairing the smallest elements and summing their absolute differences.

    Args:
    left_list (list): First list of numbers.
    right_list (list): Second list of numbers.

    Returns:
    int: Total distance between the two lists.
    """
    # Sort both lists
    left_sorted = sorted(left_list)
    right_sorted = sorted(right_list)

    # Calculate the sum of absolute differences between paired elements
    total_distance = sum(abs(a - b) for a, b in zip(left_sorted, right_sorted))

    return total_distance

# Example usage
if __name__ == "__main__":
    # Path to the input file
    file_path = "input.txt" # Replace with your actual file path

    # Read the lists from the file
    left_list, right_list = read_lists_from_file(file_path)


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

    print(f"Total distance: {total_distance}")
```

## File Format

The input file should be formatted as follows:

plaintext

 Copy code

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

## How to Use

1. Save the input data into a file, e.g., `input.txt`.
2. Update the `file_path` variable in the script to the path of your file.
3. Run the script, and it will read the lists from the file, compute the total distance, and print the result.

This approach separates file handling from computation, making the code easier to maintain and test.

The file has the list in two columns



If your input file has the lists in two columns, with one value from the left list and one value from the right list per line, you can modify the script to read the lists in that format. Here's the updated script:

## Updated Script:

python

 Copy code

```
def read_lists_from_file(file_path):
    """
    Reads two lists of integers from a file with two columns.
    Each line contains a pair of integers: one for the left list and one for the right
    list.

    Args:
        file_path (str): Path to the input file.

    Returns:
        tuple: Two lists of integers (left_list, right_list).
    """
    left_list = []
    right_list = []

    with open(file_path, 'r') as file:
        for line in file:
            left, right = map(int, line.strip().split())
            left_list.append(left)
            right_list.append(right)

    return left_list, right_list
```

```
def calculate_total_distance(left_list, right_list):
    """
    Calculate the total distance between two lists of numbers
    by pairing the smallest elements and summing their absolute differences.

    Args:
    left_list (list): First list of numbers.
    right_list (list): Second list of numbers.

    Returns:
    int: Total distance between the two lists.
    """
    # Sort both lists
    left_sorted = sorted(left_list)
    right_sorted = sorted(right_list)

    # Calculate the sum of absolute differences between paired elements
    total_distance = sum(abs(a - b) for a, b in zip(left_sorted, right_sorted))

    return total_distance

# Example usage
if __name__ == "__main__":
    # Path to the input file
    file_path = "input.txt" # Replace with your actual file path

    # Read the lists from the file
    left_list, right_list = read_lists_from_file(file_path)

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

    print(f"Total distance: {total_distance}")
```

## File Format:

The input file should have two columns, one for each list, like this:

plaintext

 Copy code

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


## How It Works:

1. **Reading the File:** Each line of the file is split into two integers, with the first integer added to `left_list` and the second to `right_list`.
2. **Sorting and Calculation:** The script processes the lists as before, sorting and calculating the total distance.

## Example Output:

For the input file:

plaintext

 Copy code

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

The output will be:

plaintext

 Copy code

```
Total distance: 11
```

This format ensures that your data is parsed correctly, with each column contributing to one of the two lists.

Please update the script for part 2 of the problem:

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 updated script to compute the similarity score for part 2:

## Updated Script:

python

Copy code

```
from collections import Counter

def read_lists_from_file(file_path):
    """
    Reads two lists of integers from a file with two columns.
    Each line contains a pair of integers: one for the left list and one for the right
    list.

    Args:
    file_path (str): Path to the input file.

    Returns:
    tuple: Two lists of integers (left_list, right_list).
    """
    left_list = []
    right_list = []

    with open(file_path, 'r') as file:
        for line in file:
            left, right = map(int, line.strip().split())
            left_list.append(left)
            right_list.append(right)

    return left_list, right_list
```



```
def calculate_similarity_score(left_list, right_list):
    """
    Calculate the similarity score between two lists.
    The similarity score is the sum of each number in the left list multiplied
    by the number of times it appears in the right list.

    Args:
    left_list (list): First list of numbers.
    right_list (list): Second list of numbers.

    Returns:
    int: Total similarity score.
    """
    # Count occurrences of each number in the right list
    right_counts = Counter(right_list)

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

    return similarity_score

# Example usage
if __name__ == "__main__":
    # Path to the input file
    file_path = "input.txt" # Replace with your actual file path

    # Read the lists from the file
    left_list, right_list = read_lists_from_file(file_path)

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

    print(f"Similarity score: {similarity_score}")
```


## How It Works:

1. **Reading Input:** The `read_lists_from_file` function parses the file into two lists of integers, one for each column.
2. **Counting Occurrences:** The `Counter` from the `collections` module is used to count the occurrences of each number in the right list.
3. **Computing Similarity:** For each number in the left list, its count in the right list is looked up, and the product of the number and its count is added to the total similarity score.

## Example Input:

File `input.txt`:

plaintext


 Copy code

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

### Example Output:

For the input above, the script will output:

plaintext

 Copy code

```
Similarity score: 31
```

### Explanation of the Example:

- **3** in the left list appears 3 times in the right list:  $3 \times 3 = 9$
- **4** in the left list appears 1 time in the right list:  $4 \times 1 = 4$
- **2** in the left list appears 0 times in the right list:  $2 \times 0 = 0$
- **1** in the left list appears 0 times in the right list:  $1 \times 0 = 0$
- The remaining **3s** in the left list add  $9 + 9$ .

Total score:  $9 + 4 + 0 + 0 + 9 + 9 = 31$ .

This script calculates the similarity score efficiently, even for larger lists.