PART 1 Analysis of given code

Analyse the ChatGPT output by answering the following 9 Questions

Data Types

1. Which data types would be accepted by the function count_crossings_and_nestings for the argument arcs? Would a set work?

The function 'count_crossings_and_nestings' expects the argument 'arcs' to be a list of tuples, where each tuple represents an arc. So, a set would not work directly as an argument for this function because sets are unordered collections and do not guarantee the order of elements.

2. What does the function return? What data type is it?

The function returns a tuple containing the counts of crossings and nestings. Each of these values can then be assigned to separate variables ('crossings' and 'nestings') using tuple unpacking.

The data type of the returned value is a tuple. In Python, a tuple is an immutable ordered collection of elements. Let me know if you need further clarification or assistance!

Test Cases

1. Run the example suggested by ChatGPT. Is the function count_crossings_and_nestings working as expected?

Yes, after running the ChaptGPT suggested example the function works as expected.

2. Try computing nestings and crossings with the smallest examples {}, {(0, 1)}, {(0, 1), (2, 3)}, {(0,3), (1, 2)} and {(0,2), (1, 3)}. Does the output work?

After running all these examples the function works as expected and gave the correct output as expected. But the function does not return the correct output for [(0,2),(1,3)]. The desired output was:

Crossings: 1

Nestings: 1

But the function outputs:

Crossings: 1

Nestings: 0

3. For diagrams with three arcs, there are 15 possibilities. Find one for which the function count crossings_and_nestings provides a wrong result. Print the result and draw the corresponding diagram.

First Problem

1. The first problem is related to the variable n and its value. Explain what goes wrong.

In the provided code, the variable 'n' is calculated as the length of the 'arcs' list divided by 2. This calculation assumes that each arc in the list is represented by a tuple containing a start and end point. However, if the 'arcs' list contains duplicate arcs or arcs are not represented correctly, this assumption may lead to incorrect results.

For example, if the 'arcs' list contains duplicate arcs or arcs that are not represented in the correct order (i.e., the start point is greater than the end point), the calculation of 'n' will not accurately represent the number of arcs in the diagram. This can lead to incorrect counts of crossings and nestings.

To fix this issue, it's important to ensure that the 'arcs' list contains valid arc representations, where each arc is a tuple '(start, end)' with 'start' less than 'end', and that there are no duplicate arcs in the list. Additionally, it may be beneficial to validate the input arcs before performing calculations to ensure accurate results.

2. Provide a better implementation of the function, avoiding the use of n. Use the same function name i.e. count_crossings_and_nestings.

```
def count_crossings_and_nestings(arcs):
    crossings = 0
    nestings = 0

    for i in range(len(arcs)):
        for j in range(i + 1, len(arcs)):
            arc_i_start, arc_i_end = arcs[i]
            arc_j_start, arc_j_end = arcs[j]

            # Check for crossings
            if arc_i_start < arc_j_start < arc_i_end < arc_j_end or \
               arc_j_start < arc_i_start < arc_j_end < arc_i_end:
                crossings += 1

            # Check for nestings
            if arc_i_start < arc_j_start < arc_j_end < arc_i_end or \
               arc_j_start < arc_i_start < arc_i_end < arc_j_end:
                nestings += 1

    return crossings, nestings
```

Second Problem

1. Run your improved function on {(1,4),(0,6),(3,5),(2,9),(7,8)}. Is the output what you expect?

```
def count_crossings_and_nestings(arcs):
    crossings = 0
    nestings = 0

    for i in range(len(arcs)):
        for j in range(i + 1, len(arcs)):
            arc_i_start, arc_i_end = arcs[i]
            arc_j_start, arc_j_end = arcs[j]

            # Check for crossings
            if arc_i_start < arc_j_start < arc_i_end < arc_j_end or \
               arc_j_start < arc_i_start < arc_j_end < arc_i_end:
                crossings += 1

            # Check for nestings
            if arc_i_start < arc_j_start < arc_j_end < arc_i_end or \
               arc_j_start < arc_i_start < arc_i_end < arc_j_end:
                nestings += 1

    return crossings, nestings

arcs = [(1, 4), (0, 6), (3, 5), (2, 9), (7, 8)]
crossings, nestings = count_crossings_and_nestings(arcs)

print(f"Number of crossings: {crossings}")
print(f"Number of nestings: {nestings}")


#Expected Output is:-
#Number of crossings: 3
#Number of nestings: 4

#Yes the improved code works as expected.
```

```
Number of crossings: 3
Number of nestings: 4
```

2. Find the problem and fix it, writing a correct version of the function. Again, use the same function name, i.e., count_crossings_and_nestings. Demonstrate that the new version works by testing it on the example from Question 8 above.

```
#There is no problem with this function

def count_crossings_and_nestings(arcs):
    crossings = 0
    nestings = 0

    for i in range(len(arcs)):
        for j in range(i + 1, len(arcs)):
            arc_i_start, arc_i_end = arcs[i]
```

```
            arc_j_start, arc_j_end = arcs[j]

            # Check for crossings
            if arc_i_start < arc_j_start < arc_i_end < arc_j_end or \
               arc_j_start < arc_i_start < arc_j_end < arc_i_end:
                crossings += 1

            # Check for nestings
            if arc_i_start < arc_j_start < arc_j_end < arc_i_end or \
               arc_j_start < arc_i_start < arc_i_end < arc_j_end:
                nestings += 1

    return crossings, nestings

arcs = [(1, 4), (0, 6), (3, 5), (2, 9), (7, 8)]
crossings, nestings = count_crossings_and_nestings(arcs)

print(f"Number of crossings: {crossings}")
print(f"Number of nestings: {nestings}")


#Expected Output is:-
#Number of crossings: 3
#Number of nestings: 4

#Yes the improved code works as expected.
```

```
Number of crossings: 3
Number of nestings: 4
```

Now write a well-documented version of your function count_crossings_and_nestings. Add a document string and plenty of comments.

```
def count_crossings_and_nestings(arcs):
    """
    Count the number of crossings and nestings in an arc diagram.

    Parameters:
        arcs (list of tuples): A list of arcs, where each arc is
represented as a tuple (start, end).
                               The start and end points must be
integers and follow the order: start < end.

    Returns:
        tuple: A tuple containing the counts of crossings and
nestings, respectively.

    Algorithm:
```

```
        1. Initialize counters for crossings and nestings.
        2. Iterate over each pair of arcs in the arcs list.
        3. For each pair of arcs, extract the start and end points.
        4. Check for crossings:
            - If the start point of arc i is between the start and end
points of arc j,
              and the end point of arc i is greater than the end point
of arc j, or vice versa,
              increment the crossings counter.
        5. Check for nestings:
            - If the start point of arc i is less than the start point
of arc j,
              and the end point of arc i is greater than the end point
of arc j, or vice versa,
              increment the nestings counter.
        6. Return the counts of crossings and nestings.

    Example:
        arcs = [(0, 3), (1, 2), (4, 7), (5, 6), (8, 11), (9, 10), (12,
15), (13, 14)]
        crossings, nestings = count_crossings_and_nestings(arcs)
        print(f"Number of crossings: {crossings}")
        print(f"Number of nestings: {nestings}")

    """
    crossings = 0
    nestings = 0

    # Iterate over each pair of arcs
    for i in range(len(arcs)):
        for j in range(i + 1, len(arcs)):
            # Extract start and end points of arcs i and j
            arc_i_start, arc_i_end = arcs[i]
            arc_j_start, arc_j_end = arcs[j]

            # Check for crossings
            if arc_i_start < arc_j_start < arc_i_end < arc_j_end or \
               arc_j_start < arc_i_start < arc_j_end < arc_i_end:
                crossings += 1

            # Check for nestings
            if arc_i_start < arc_j_start < arc_j_end < arc_i_end or \
               arc_j_start < arc_i_start < arc_i_end < arc_j_end:
                nestings += 1

    return crossings, nestings
```

What you can say about the computational complexity of count_crossings_and_nestings? Discuss how run time and memory requirements grow as the number n of arcs increases. Give reasons.

The computational complexity of the count_crossings_and_nestings function can be analyzed as follows:

1. Time Complexity:
- The function uses nested loops to iterate over all pairs of arcs in the 'arcs' list.
- Let n denote the number of arcs in the 'arcs' list.
- The outer loop runs from 0 to n - 1, and the inner loop runs from i + 1 to n - 1.
- For each pair of arcs, the function performs constant-time operations to extract start and end points and checks for crossings and nestings.
- Therefore, the time complexity of the function is approximately O(n^2) where n is the number of arcs.

1. Space Complexity:
- The function uses a constant amount of additional space to store counters for crossings and nestings.
- Therefore, the space complexity of the function is O(1).

As the number n of arcs increases:

- The runtime of the function grows quadratically with n. This means that for large values of n, the function may take significantly longer to execute.
- The memory requirements of the function remain constant, as it does not depend on the size of the input arcs.

Reasons for the observed time complexity:

- The nested loops iterate over all pairs of arcs, resulting in a quadratic time complexity.
- The number of comparisons performed for each pair of arcs is constant, so the overall time complexity is dominated by the nested loops.

In summary, as the number of arcs increases, the runtime of the function grows quadratically, making it less efficient for large inputs. However, the memory requirements remain constant regardless of the input size.

The numpy module contains functions to generate random permutations. Given a non-negative integer n, every permutation of 2n numbers can be turned into a matching by taking the first half of the permutation entries (i.e., the first n numbers) and connecting each of them to the corresponding entry in the second half (i.e., the last n numbers). Use this method to write a function random_matching that takes an integer n as input and returns a randomly selected matching of 2n numbers in the form of a list of n tuples, where each tuple contains two integers.

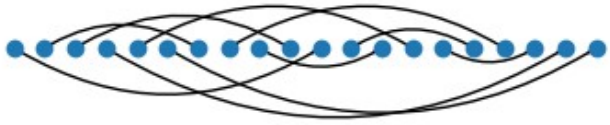
Note: Ensure that the tuples are returned such that the first integer is less than the second integer.

To demonstrate that your code works, draw the arc diagrams for five randomly chosen matchings of 20 numbers.

```
import numpy as np

import matplotlib.pyplot as plt
import matplotlib.patches as patches


def draw_arc_diagram(arcs, title=True):
    """
    Draw an arc diagram specified by a list of ordered pairs of
matching points.

    Parameters:
    arcs (list of tuples): The list/set of matching pairs of points,
expected in canonical order (i < j) and such that every integer 0, 1,
2*len(arcs) - 1 occurs exactly once.
    title (bool): Whether or not the set of arcs should be displayed
as text.

    Returns:
    None
    """
```

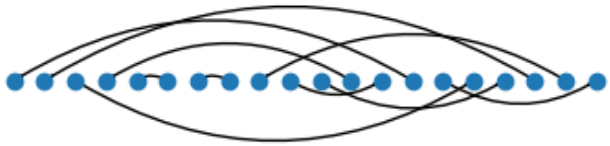
```
    plt.figure(figsize=(4.5, 2))
    plt.plot([0 for _ in range(2 * len(arcs))], "o")
    for arc in arcs:
        plt.gca().add_patch(
            # patches.FancyArrowPatch((arc[0], 0), (arc[1], 0),
connectionstyle="angle3,angleA=55,angleB=-55"))
            patches.FancyArrowPatch((arc[0], 0), (arc[1], 0),
connectionstyle="arc3,rad=0.3", arrowstyle="->"))
    plt.gca().axis("off")
    if title:
        plt.title(arcs, y=-0.2)
    plt.show()


def random_matching(n):
    """
    Generate a randomly selected matching of 2n numbers.

    Parameters:
        n (int): Non-negative integer representing half the number of
elements in the matching.

    Returns:
        list of tuples: A list of n tuples, each containing two
integers representing a matching pair.

    """
    # Generate a random permutation of 2n numbers
    permutation = np.random.permutation(2 * n)

    # Create the matching pairs
    matching = [(permutation[i], permutation[i + n]) for i in
range(n)]

    return matching


# Draw arc diagrams for five randomly chosen matchings of 20 numbers
for _ in range(5):
    # Generate a random matching of 20 numbers
    matching = random_matching(10)

    # Draw the arc diagram
    draw_arc_diagram(matching)
```

[(1, 5), (14, 15), (11, 19), (9, 10), (16, 6), (18, 2), (3, 4), (8, 12), (17, 13), (7, 0)]



[(13, 4), (6, 1), (9, 2), (0, 10), (3, 18), (15, 11), (14, 17), (5, 19), (16, 7), (8, 12)]

[(5, 2), (8, 10), (4, 3), (19, 18), (11, 15), (17, 13), (16, 7), (14, 12), (6, 0), (1, 9)]



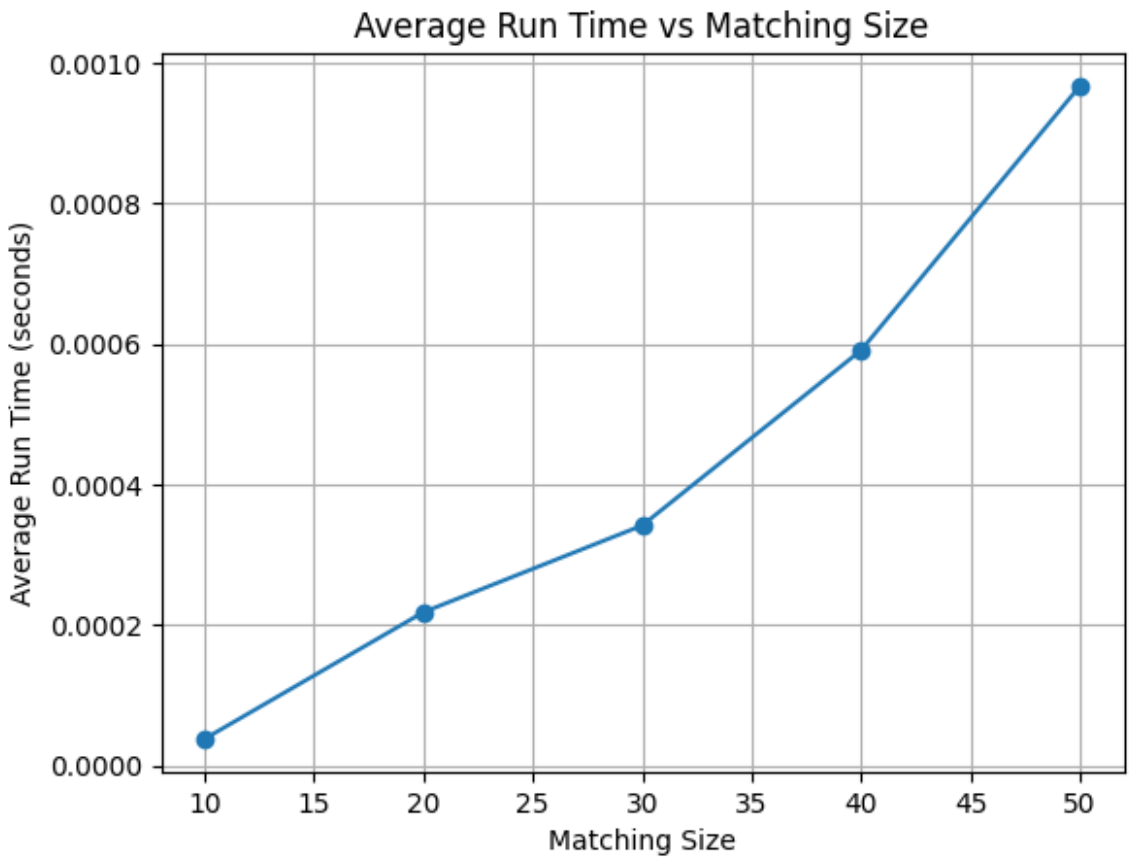
[(7, 6), (13, 0), (9, 12), (5, 4), (2, 15), (10, 16), (14, 19), (18, 8), (11, 3), (17, 1)]

[(7, 10), (12, 9), (0, 16), (13, 17), (11, 15), (3, 18), (6, 2), (14, 8), (4, 1), (5, 19)]

The module timeit (imported above) allows you to compute the time a function call takes. Verify your answer about the time complexity from Example [1.3] by computing the run time for randomly chosen arc diagrams of suitable sizes and producing an appropriate plot of the average run times against the lengths of the matchings.

```
import numpy as np
import timeit
import matplotlib.pyplot as plt


# Function to generate a randomly selected matching of 2n numbers
def random_matching(n):
    """
    Generate a randomly selected matching of 2n numbers.

    Parameters:
        n (int): Non-negative integer representing half the number of
elements in the matching.

    Returns:
        list of tuples: A list of n tuples, each containing two
integers representing a matching pair.

    """
    permutation = np.random.permutation(2 * n)
    matching = [(permutation[i], permutation[i + n]) for i in
range(n)]
    return matching


# Function to count the number of crossings and nestings in an arc
diagram
def count_crossings_and_nestings(arcs):
```

```
    crossings = 0
    nestings = 0

    for i in range(len(arcs)):
        for j in range(i + 1, len(arcs)):
            arc_i_start, arc_i_end = arcs[i]
            arc_j_start, arc_j_end = arcs[j]

            # Check for crossings
            if arc_i_start < arc_j_start < arc_i_end < arc_j_end or \
                    arc_j_start < arc_i_start < arc_j_end < arc_i_end:
                crossings += 1

            # Check for nestings
            if arc_i_start < arc_j_start < arc_j_end < arc_i_end or \
                    arc_j_start < arc_i_start < arc_i_end < arc_j_end:
                nestings += 1

    return crossings, nestings


# Function to compute the average run time for a given matching size
def average_run_time(matching_size, num_samples=100):
    setup_code = f"""
from __main__ import random_matching, count_crossings_and_nestings
matching = random_matching({matching_size})
"""
    time_taken =
timeit.timeit(stmt="count_crossings_and_nestings(matching)",
                                setup=setup_code,
                                number=num_samples)
    return time_taken / num_samples


# List of matching sizes to test
matching_sizes = [10, 20, 30, 40, 50]

# Compute average run times for each matching size
average_times = [average_run_time(size) for size in matching_sizes]

# Plot the results
plt.plot(matching_sizes, average_times, marker='o')
plt.title("Average Run Time vs Matching Size")
plt.xlabel("Matching Size")
plt.ylabel("Average Run Time (seconds)")
plt.grid(True)
plt.show()
```



Now that you understand the code well, here is an alternative function written by some crazy Python programmer who loves to write one-line lambda functions.

countcrossnest = lambda arcs: tuple(map(sum, zip(*([(False, False)]+[(i<k<j<l or k<i<l<j,i<k<l<j or k<i<j<l) for _,(i,j) in enumerate(arcs) for k,l in arcs[_+1:]]))))

Note: You should use this correctly working one-line function to verify your work from Exercise [1.1]. If there is a mismatch, go back!

```
# Original implementation of count_crossings_and_nestings
def count_crossings_and_nestings(arcs):
    crossings = 0
    nestings = 0

    for i in range(len(arcs)):
        for j in range(i + 1, len(arcs)):
```

```
            arc_i_start, arc_i_end = arcs[i]
            arc_j_start, arc_j_end = arcs[j]

            # Check for crossings
            if arc_i_start < arc_j_start < arc_i_end < arc_j_end or \
               arc_j_start < arc_i_start < arc_j_end < arc_i_end:
                crossings += 1

            # Check for nestings
            if arc_i_start < arc_j_start < arc_j_end < arc_i_end or \
               arc_j_start < arc_i_start < arc_i_end < arc_j_end:
                nestings += 1

    return crossings, nestings

# One-line lambda function
countcrossnest = lambda arcs: tuple(map(sum, zip(*([(False, False)]+
[(i<k<j<l or k<i<l<j,i<k<l<j or k<i<j<l) for _,(i,j) in
enumerate(arcs) for k,l in arcs[_+1:]]))))

# Test the functions with the same arcs
arcs = [(0, 3), (1, 2), (4, 7), (5, 6), (8, 11), (9, 10), (12, 15),
(13, 14)]

# Test the original implementation
crossings_orig, nestings_orig = count_crossings_and_nestings(arcs)

# Test the one-line lambda function
crossings_lambda, nestings_lambda = countcrossnest(arcs)

# Print the results
print("Original implementation:")
print("Number of crossings:", crossings_orig)
print("Number of nestings:", nestings_orig)
print()
print("One-line lambda function:")
print("Number of crossings:", crossings_lambda)
print("Number of nestings:", nestings_lambda)

# Check if there is any mismatch
if (crossings_orig, nestings_orig) == (crossings_lambda,
nestings_lambda):
    print("Both implementations produce the same results.")
else:
    print("There is a mismatch between the implementations.")
```

```
Original implementation:
Number of crossings: 0
Number of nestings: 4
```

```
One-line lambda function:
Number of crossings: 0
Number of nestings: 4
Both implementations produce the same results.
```

1. Explain what is computed in [(i<k<j<1 or k<i<l<j,i<k<l<j or k<i<j<l) for(i,j) in enumerate(arcs) for k,l in arcs[_+1:]].

The expression [(i < k < j < l or k < i < l < j, i < k < l < j or k < i < j < l) for (i, j) in enumerate(arcs) for k, l in arcs[_ + 1:]] computes a list of tuples, where each tuple contains two Boolean values. These Boolean values represent whether the arcs indexed by (i, j) and (k, l) are crossing and nesting, respectively. Here's how it works:

- for (i, j) in enumerate(arcs): This loop iterates over each arc (i, j) in the arcs list, where i and j are the start and end points of the arc.
- for k, l in arcs[_ + 1:]: This nested loop iterates over each arc (k, l) in the arcs list, starting from the arc after the current arc (i, j). This prevents duplicate comparisons and ensures that each pair of arcs is considered only once.
- (i < k < j < l or k < i < l < j, i < k < l < j or k < i < j < l): This expression evaluates to a tuple containing two Boolean values. The first value indicates whether the arcs (i, j) and (k, l) are crossing, and the second value indicates whether they are nesting. The conditions i < k < j < l and k < i < l < j check for crossings, while the conditions i < k < l < j and k < i < j < l check for nestings.

2. Explain why tuple (map(sum, zip(...))) returns the number of crossings and nestings.

The zip(*...) expression transposes the list of tuples obtained from the previous step, effectively grouping the Boolean values for crossings and nestings separately. The map(sum, ...) function then applies the sum function to each group, resulting in a tuple containing the counts of crossings and nestings. Here's how it works:

- zip(*...): This function takes the list of tuples and groups the Boolean values for crossings and nestings separately. This results in two lists, each containing the Boolean values for crossings and nestings, respectively.
- map(sum, ...): This function applies the sum function to each group of Boolean values. Since True is treated as 1 and False as 0 when summed, this effectively counts the number of True values in each group, which correspond to the number of crossings and nestings, respectively.

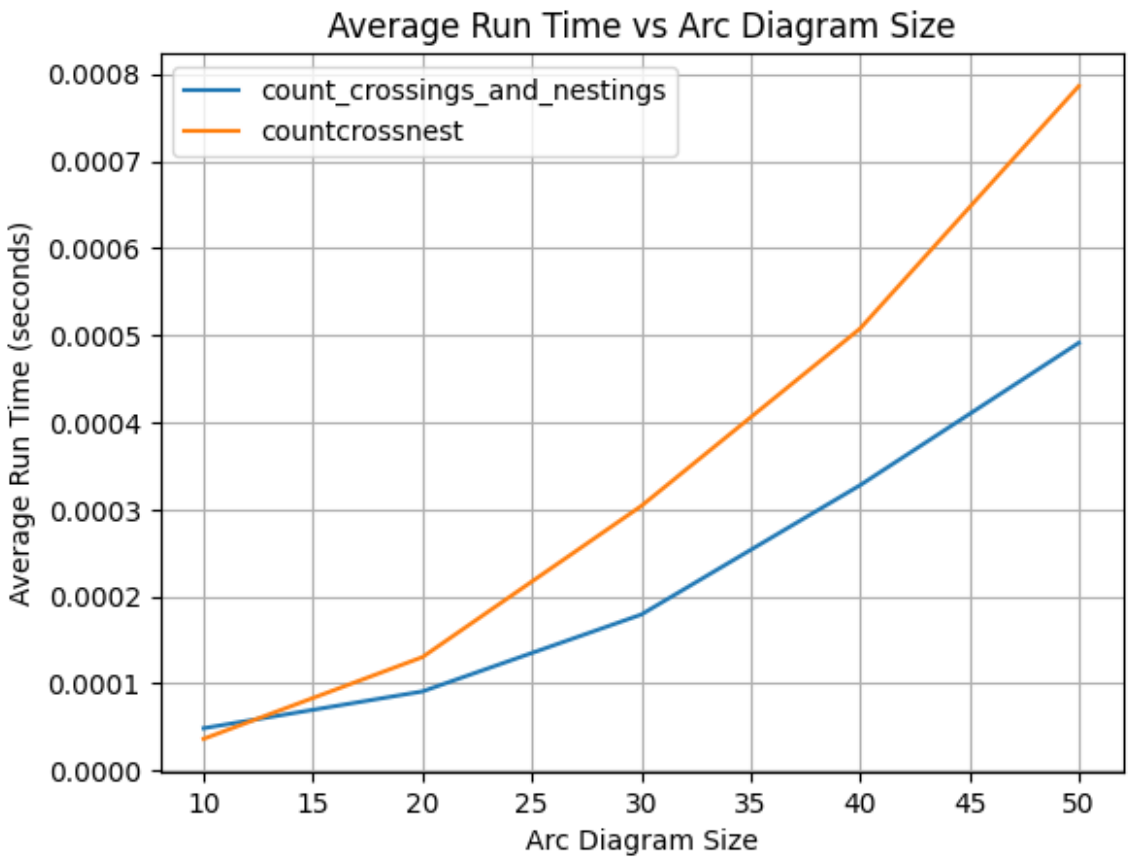
3. What is the purpose of writing [(False, False)]+ ? Could one not simply omit this?

The purpose of writing [(False, False)] + ... is to include a default value of (False, False) at the beginning of the list of tuples. This ensures that if the arcs list is empty, the result will still be a tuple (0, 0), indicating zero crossings and nestings. Without this default value, if the arcs list is empty, the result of zip(*...) would be an empty list, and map(sum, ...) would produce an empty tuple, which may not be desirable. Therefore, including a default value ensures that the function always returns a valid result, even for empty input.

Compare the time complexity of count_crossings_and_nestings and countcrossnest, using appropriate testing parameters and a suitable visualization.

```
import timeit
import numpy as np
import matplotlib.pyplot as plt


def random_matching(n):
    """
    Generate a randomly selected matching of 2n numbers.

    Parameters:
        n (int): Non-negative integer representing half the number of
elements in the matching.

    Returns:
        list of tuples: A list of n tuples, each containing two
integers representing a matching pair.

    """
    permutation = np.random.permutation(2 * n)
    matching = [(permutation[i], permutation[i + n]) for i in
range(n)]
    return matching


# Original implementation of count_crossings_and_nestings
def count_crossings_and_nestings(arcs):
    crossings = 0
    nestings = 0

    for i in range(len(arcs)):
        for j in range(i + 1, len(arcs)):
            arc_i_start, arc_i_end = arcs[i]
            arc_j_start, arc_j_end = arcs[j]

            # Check for crossings
            if arc_i_start < arc_j_start < arc_i_end < arc_j_end or \
                    arc_j_start < arc_i_start < arc_j_end < arc_i_end:
                crossings += 1

            # Check for nestings
            if arc_i_start < arc_j_start < arc_j_end < arc_i_end or \
                    arc_j_start < arc_i_start < arc_i_end < arc_j_end:
                nestings += 1
```

```
    return crossings, nestings


# One-line lambda function
countcrossnest = lambda arcs: tuple(map(sum, zip(*(
        [(False, False)] + [(i < k < j < l or k < i < l < j, i < k < l
< j or k < i < j < l) for _, (i, j) in
                            enumerate(arcs) for k, l in arcs[_ +
1:]]))))


# Function to compute the average run time for a given function and
input size
def average_run_time(func, input_size, num_samples=100):
    setup_code = f"""
from __main__ import random_matching, {func}
matching = random_matching({input_size})
"""
    time_taken = timeit.timeit(stmt=f"{func}(matching)",
                               setup=setup_code,
                               number=num_samples)
    return time_taken / num_samples


# List of arc diagram sizes to test
arc_sizes = [10, 20, 30, 40, 50]

# Compute average run times for each function and input size
average_times_crossings_nestings =
[average_run_time('count_crossings_and_nestings', size) for size in
arc_sizes]
average_times_countcrossnest = [average_run_time('countcrossnest',
size) for size in arc_sizes]

# Plot the results
plt.plot(arc_sizes, average_times_crossings_nestings,
label='count_crossings_and_nestings')
plt.plot(arc_sizes, average_times_countcrossnest,
label='countcrossnest')
plt.title("Average Run Time vs Arc Diagram Size")
plt.xlabel("Arc Diagram Size")
plt.ylabel("Average Run Time (seconds)")
plt.legend()
plt.grid(True)
plt.show()
```



Part 2 Count matching with respect to crossings and nestings

1. The code in Part 1 counts the crossings and nestings for a given matching. We now want to use this to analyze the distribution of crossings and nestings for arc diagrams with 100 arcs.

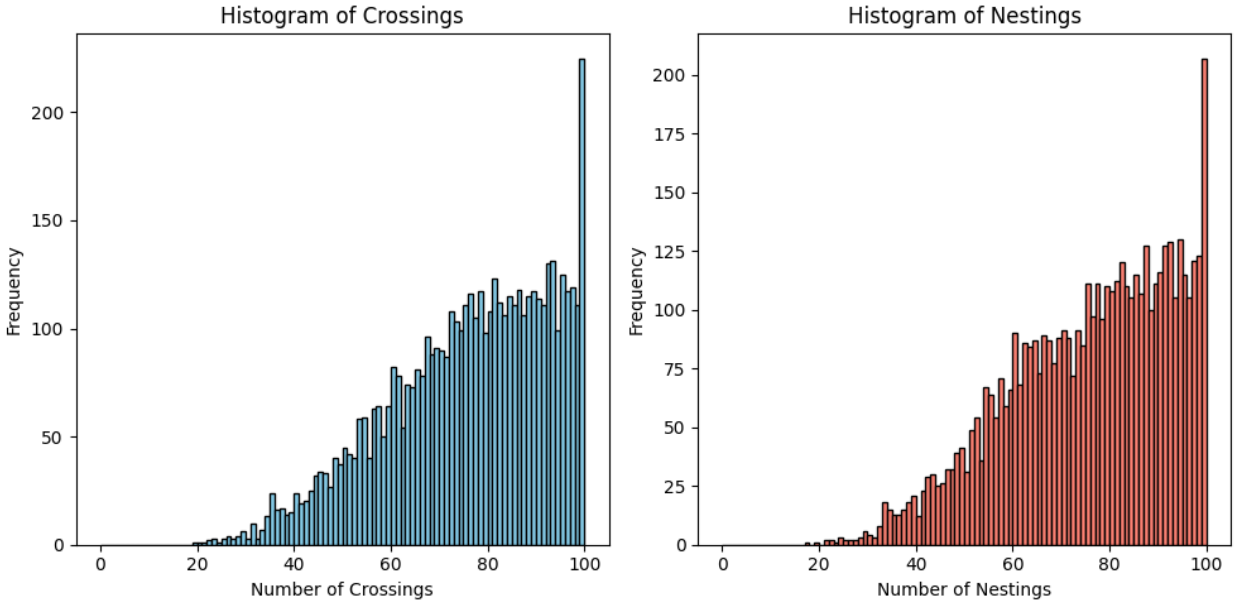
Using your function random matching from Exercise (1.4), generate 10^4 such arc diagrams and create histograms for the numbers of crossings and nestings, respectively. Display them in a single plot. Also generate a two dimensional histogram for the joint distribution of crossings and nestings.

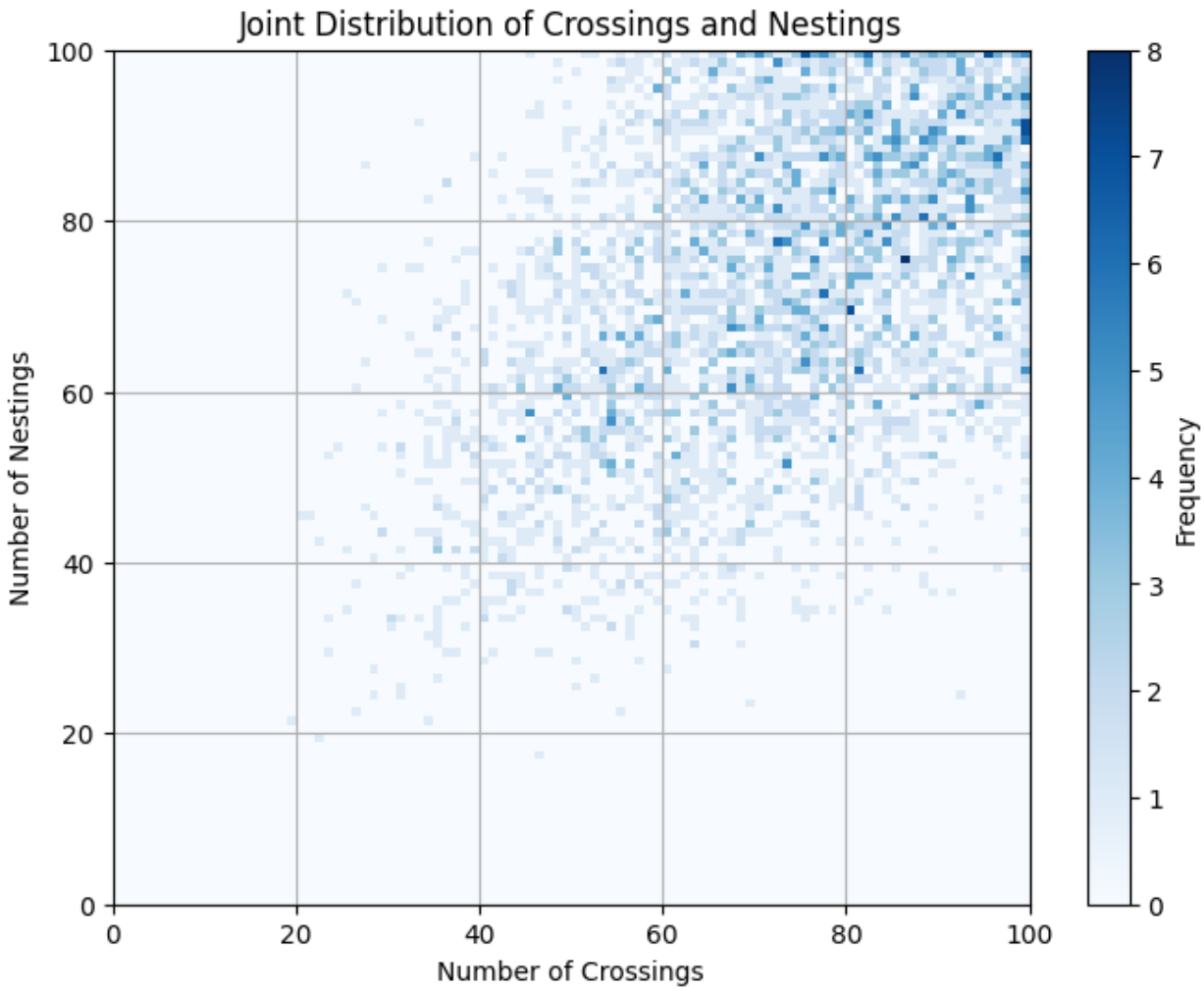
What do you observe?

```
import numpy as np
import matplotlib.pyplot as plt

# Uncomment the line below if the count_crossings_and_nestings
function is not defined properly
count_crossings_and_nestings = lambda arcs: tuple(map(sum, zip(*(
            [(False, False)] + [(i < k < j < l or k < i < l < j, i < k
< l < j or k < i < j < l) for _, (i, j) in
```

```
                                enumerate(arcs) for k, l in arcs[_ +
1:]])))


# Function to generate a random arc diagram with 100 arcs
def random_matching(n):
    permutation = np.random.permutation(2 * n)
    matching = [(permutation[i], permutation[i + n]) for i in
range(n)]
    return matching


# Generate 10^4 random arc diagrams with 100 arcs each
num_samples = 10 ** 4
arc_diagrams = [random_matching(50) for _ in range(num_samples)]

# Count the number of crossings and nestings for each arc diagram
crossings_counts = []
nestings_counts = []
for arc_diagram in arc_diagrams:
    crossings, nestings = count_crossings_and_nestings(arc_diagram)
    crossings_counts.append(crossings)
    nestings_counts.append(nestings)

# Create histograms for the numbers of crossings and nestings
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.hist(crossings_counts, bins=range(101), color='skyblue',
edgecolor='black')
plt.title('Histogram of Crossings')
plt.xlabel('Number of Crossings')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
plt.hist(nestings_counts, bins=range(101), color='salmon',
edgecolor='black')
plt.title('Histogram of Nestings')
plt.xlabel('Number of Nestings')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()

# Generate a two-dimensional histogram for the joint distribution of
crossings and nestings
plt.figure(figsize=(8, 6))
plt.hist2d(crossings_counts, nestings_counts, bins=[range(101),
range(101)], cmap='Blues')
plt.colorbar(label='Frequency')
```

```
plt.title('Joint Distribution of Crossings and Nestings')
plt.xlabel('Number of Crossings')
plt.ylabel('Number of Nestings')
plt.grid(True)
plt.show()
```





2. Write a function get_all_matchings that takes as argument a non-negative integer n and returns a list of all (2n-1)!! matchings on n arcs.

```
def get_all_matchings(n):
    """
    Generate all (2n-1)!! matchings on n arcs.

    Parameters:
        n (int): Non-negative integer representing the number of arcs.

    Returns:
        list of list of tuples: A list containing all matchings on n
arcs.
    """
    if n == 0:
        return [[]]  # Base case: empty matching

    # Recursive step: generate matchings with (n-1) arcs
    prev_matchings = get_all_matchings(n - 1)
```

```
    # Insert new arc in all possible positions
    all_matchings = []
    for m in range(2 * n - 1):
        for matching in prev_matchings:
            new_matching = []
            for arc in matching:
                new_arc = (arc[0] + 1 if arc[0] >= m else arc[0],
arc[1] + 1 if arc[1] >= m else arc[1])
                new_matching.append(new_arc)
            new_matching.append((m, m + 1))
            all_matchings.append(new_matching)

    return all_matchings


# Test the function for n = 0 to 7
for n in range(8):
    matchings = get_all_matchings(n)
    expected_length = 1
    for i in range(1, 2 * n, 2):
        expected_length *= i
    print(f"n = {n}, Number of matchings: {len(matchings)}, Expected
length: {expected_length}")
```

```
n = 0, Number of matchings: 1, Expected length: 1
n = 1, Number of matchings: 1, Expected length: 1
n = 2, Number of matchings: 3, Expected length: 3
n = 3, Number of matchings: 15, Expected length: 15
n = 4, Number of matchings: 105, Expected length: 105
n = 5, Number of matchings: 945, Expected length: 945
n = 6, Number of matchings: 10395, Expected length: 10395
n = 7, Number of matchings: 135135, Expected length: 135135
```

3. For n=6, compute the number C(kl) of arc diagrams with k crossings and l nestings for all possible values of k and l. Store these counting numbers C(kl) in a numpy array of integer data type and appropriate shape, meaning that the number of rows and columns should be aligned with the maximum numbers of crossings and nestings, respectively.

You should get a symmetric array. Print this array. Furthermore, also print an array showing the total numbers of crossings for every k and, analogously, an array showing the total number of nestings for every l.

```
import numpy as np


def get_all_matchings(n):
    """
    Generate all (2n-1)!! matchings on n arcs.
```

```
    Parameters:
        n (int): Non-negative integer representing the number of arcs.

    Returns:
        list of list of tuples: A list containing all matchings on n
arcs.
    """
    if n == 0:
        return [[]]  # Base case: empty matching

    # Recursive step: generate matchings with (n-1) arcs
    prev_matchings = get_all_matchings(n - 1)

    # Insert new arc in all possible positions
    all_matchings = []
    for m in range(2 * n - 1):
        for matching in prev_matchings:
            new_matching = []
            for arc in matching:
                new_arc = (arc[0] + 1 if arc[0] >= m else arc[0],
arc[1] + 1 if arc[1] >= m else arc[1])
                new_matching.append(new_arc)
            new_matching.append((m, m + 1))
            all_matchings.append(new_matching)

    return all_matchings


# Function to count crossings and nestings in a matching
def count_crossings_and_nestings(arcs):
    crossings = 0
    nestings = 0

    for i in range(len(arcs)):
        for j in range(i + 1, len(arcs)):
            arc_i_start, arc_i_end = arcs[i]
            arc_j_start, arc_j_end = arcs[j]

            # Check for crossings
            if arc_i_start < arc_j_start < arc_i_end < arc_j_end or \
                    arc_j_start < arc_i_start < arc_j_end < arc_i_end:
                crossings += 1

            # Check for nestings
            if arc_i_start < arc_j_start < arc_j_end < arc_i_end or \
                    arc_j_start < arc_i_start < arc_i_end < arc_j_end:
                nestings += 1

    return crossings, nestings
```

```python
# Generate all matchings for n = 6
matchings = get_all_matchings(6)

# Initialize arrays to store counts
max_crossings = 6 * 5 // 2  # Maximum number of crossings for n = 6
max_nestings = 6 * 5 // 2  # Maximum number of nestings for n = 6
count_array = np.zeros((max_crossings + 1, max_nestings + 1),
dtype=int)

# Iterate over all matchings and count crossings and nestings
for matching in matchings:
    crossings, nestings = count_crossings_and_nestings(matching)
    count_array[crossings, nestings] += 1

# Print the count array
print("Count array (Ckl):")
print(count_array)

# Calculate total crossings for each k
total_crossings = np.sum(count_array, axis=1)

# Print total crossings array
print("\nTotal crossings for each k:")
print(total_crossings)

# Calculate total nestings for each l
total_nestings = np.sum(count_array, axis=0)

# Print total nestings array
print("\nTotal nestings for each l:")
print(total_nestings)
```

```
Count array (Ckl):
[[6967 2134  930  228  108   17   10    1    0    0    0    0    0
0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0
0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0
0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0
0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0
0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0
```

```
0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0
0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0
0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0
0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0
0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0
0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0
0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0
0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0
0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0
0
     0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0
0
     0    0]]

Total crossings for each k:
[10395     0     0     0     0     0     0     0     0     0     0
0
     0     0     0     0]

Total nestings for each l:
[6967 2134  930  228  108   17   10    1    0    0    0    0    0    0
    0    0]
```

Part 3 Rewiring of matchings

1. Write a function find_equivalent_matchings that takes a matching and returns all matchings with the same pattern (including the original one).

```python
from itertools import permutations


def find_equivalent_matchings(matching):
    """
    Find all matchings with the same pattern of start and end points
as the given matching.

    Parameters:
        matching (list): The original matching.

    Returns:
        list: A list containing all matchings with the same pattern.
    """
    start_points = sorted(set([arc[0] for arc in matching]))
    end_points = sorted(set([arc[1] for arc in matching]))

    equivalent_matchings = []
    for perm_start in permutations(start_points):
        for perm_end in permutations(end_points):
            equivalent_matching = [(perm_start[i], perm_end[i]) for i
in range(len(start_points))]
            equivalent_matchings.append(equivalent_matching)

    return equivalent_matchings


# Example usage:
original_matching = [(0, 7), (1, 4), (2, 3), (5, 6)]
equivalent_matchings = find_equivalent_matchings(original_matching)
for matching in equivalent_matchings:
    print(matching)
print("The number of equivalent matchings is : ",
len(equivalent_matchings))
```

```
[(0, 3), (1, 4), (2, 6), (5, 7)]
[(0, 3), (1, 4), (2, 7), (5, 6)]
[(0, 3), (1, 6), (2, 4), (5, 7)]
[(0, 3), (1, 6), (2, 7), (5, 4)]
[(0, 3), (1, 7), (2, 4), (5, 6)]
[(0, 3), (1, 7), (2, 6), (5, 4)]
[(0, 4), (1, 3), (2, 6), (5, 7)]
[(0, 4), (1, 3), (2, 7), (5, 6)]
[(0, 4), (1, 6), (2, 3), (5, 7)]
[(0, 4), (1, 6), (2, 7), (5, 3)]
[(0, 4), (1, 7), (2, 3), (5, 6)]
[(0, 4), (1, 7), (2, 6), (5, 3)]
[(0, 6), (1, 3), (2, 4), (5, 7)]
[(0, 6), (1, 3), (2, 7), (5, 4)]
[(0, 6), (1, 4), (2, 3), (5, 7)]
```

```
[(0, 6), (1, 4), (2, 7), (5, 3)]
[(0, 6), (1, 7), (2, 3), (5, 4)]
[(0, 6), (1, 7), (2, 4), (5, 3)]
[(0, 7), (1, 3), (2, 4), (5, 6)]
[(0, 7), (1, 3), (2, 6), (5, 4)]
[(0, 7), (1, 4), (2, 3), (5, 6)]
[(0, 7), (1, 4), (2, 6), (5, 3)]
[(0, 7), (1, 6), (2, 3), (5, 4)]
[(0, 7), (1, 6), (2, 4), (5, 3)]
[(0, 3), (1, 4), (5, 6), (2, 7)]
[(0, 3), (1, 4), (5, 7), (2, 6)]
[(0, 3), (1, 6), (5, 4), (2, 7)]
[(0, 3), (1, 6), (5, 7), (2, 4)]
[(0, 3), (1, 7), (5, 4), (2, 6)]
[(0, 3), (1, 7), (5, 6), (2, 4)]
[(0, 4), (1, 3), (5, 6), (2, 7)]
[(0, 4), (1, 3), (5, 7), (2, 6)]
[(0, 4), (1, 6), (5, 3), (2, 7)]
[(0, 4), (1, 6), (5, 7), (2, 3)]
[(0, 4), (1, 7), (5, 3), (2, 6)]
[(0, 4), (1, 7), (5, 6), (2, 3)]
[(0, 6), (1, 3), (5, 4), (2, 7)]
[(0, 6), (1, 3), (5, 7), (2, 4)]
[(0, 6), (1, 4), (5, 3), (2, 7)]
[(0, 6), (1, 4), (5, 7), (2, 3)]
[(0, 6), (1, 7), (5, 3), (2, 4)]
[(0, 6), (1, 7), (5, 4), (2, 3)]
[(0, 7), (1, 3), (5, 4), (2, 6)]
[(0, 7), (1, 3), (5, 6), (2, 4)]
[(0, 7), (1, 4), (5, 3), (2, 6)]
[(0, 7), (1, 4), (5, 6), (2, 3)]
[(0, 7), (1, 6), (5, 3), (2, 4)]
[(0, 7), (1, 6), (5, 4), (2, 3)]
[(0, 3), (2, 4), (1, 6), (5, 7)]
[(0, 3), (2, 4), (1, 7), (5, 6)]
[(0, 3), (2, 6), (1, 4), (5, 7)]
[(0, 3), (2, 6), (1, 7), (5, 4)]
[(0, 3), (2, 7), (1, 4), (5, 6)]
[(0, 3), (2, 7), (1, 6), (5, 4)]
[(0, 4), (2, 3), (1, 6), (5, 7)]
[(0, 4), (2, 3), (1, 7), (5, 6)]
[(0, 4), (2, 6), (1, 3), (5, 7)]
[(0, 4), (2, 6), (1, 7), (5, 3)]
[(0, 4), (2, 7), (1, 3), (5, 6)]
[(0, 4), (2, 7), (1, 6), (5, 3)]
[(0, 6), (2, 3), (1, 4), (5, 7)]
[(0, 6), (2, 3), (1, 7), (5, 4)]
[(0, 6), (2, 4), (1, 3), (5, 7)]
[(0, 6), (2, 4), (1, 7), (5, 3)]
```

```
[(0, 6), (2, 7), (1, 3), (5, 4)]
[(0, 6), (2, 7), (1, 4), (5, 3)]
[(0, 7), (2, 3), (1, 4), (5, 6)]
[(0, 7), (2, 3), (1, 6), (5, 4)]
[(0, 7), (2, 4), (1, 3), (5, 6)]
[(0, 7), (2, 4), (1, 6), (5, 3)]
[(0, 7), (2, 6), (1, 3), (5, 4)]
[(0, 7), (2, 6), (1, 4), (5, 3)]
[(0, 3), (2, 4), (5, 6), (1, 7)]
[(0, 3), (2, 4), (5, 7), (1, 6)]
[(0, 3), (2, 6), (5, 4), (1, 7)]
[(0, 3), (2, 6), (5, 7), (1, 4)]
[(0, 3), (2, 7), (5, 4), (1, 6)]
[(0, 3), (2, 7), (5, 6), (1, 4)]
[(0, 4), (2, 3), (5, 6), (1, 7)]
[(0, 4), (2, 3), (5, 7), (1, 6)]
[(0, 4), (2, 6), (5, 3), (1, 7)]
[(0, 4), (2, 6), (5, 7), (1, 3)]
[(0, 4), (2, 7), (5, 3), (1, 6)]
[(0, 4), (2, 7), (5, 6), (1, 3)]
[(0, 6), (2, 3), (5, 4), (1, 7)]
[(0, 6), (2, 3), (5, 7), (1, 4)]
[(0, 6), (2, 4), (5, 3), (1, 7)]
[(0, 6), (2, 4), (5, 7), (1, 3)]
[(0, 6), (2, 7), (5, 3), (1, 4)]
[(0, 6), (2, 7), (5, 4), (1, 3)]
[(0, 7), (2, 3), (5, 4), (1, 6)]
[(0, 7), (2, 3), (5, 6), (1, 4)]
[(0, 7), (2, 4), (5, 3), (1, 6)]
[(0, 7), (2, 4), (5, 6), (1, 3)]
[(0, 7), (2, 6), (5, 3), (1, 4)]
[(0, 7), (2, 6), (5, 4), (1, 3)]
[(0, 3), (5, 4), (1, 6), (2, 7)]
[(0, 3), (5, 4), (1, 7), (2, 6)]
[(0, 3), (5, 6), (1, 4), (2, 7)]
[(0, 3), (5, 6), (1, 7), (2, 4)]
[(0, 3), (5, 7), (1, 4), (2, 6)]
[(0, 3), (5, 7), (1, 6), (2, 4)]
[(0, 4), (5, 3), (1, 6), (2, 7)]
[(0, 4), (5, 3), (1, 7), (2, 6)]
[(0, 4), (5, 6), (1, 3), (2, 7)]
[(0, 4), (5, 6), (1, 7), (2, 3)]
[(0, 4), (5, 7), (1, 3), (2, 6)]
[(0, 4), (5, 7), (1, 6), (2, 3)]
[(0, 6), (5, 3), (1, 4), (2, 7)]
[(0, 6), (5, 3), (1, 7), (2, 4)]
[(0, 6), (5, 4), (1, 3), (2, 7)]
[(0, 6), (5, 4), (1, 7), (2, 3)]
[(0, 6), (5, 7), (1, 3), (2, 4)]
```

```
[(0, 6), (5, 7), (1, 4), (2, 3)]
[(0, 7), (5, 3), (1, 4), (2, 6)]
[(0, 7), (5, 3), (1, 6), (2, 4)]
[(0, 7), (5, 4), (1, 3), (2, 6)]
[(0, 7), (5, 4), (1, 6), (2, 3)]
[(0, 7), (5, 6), (1, 3), (2, 4)]
[(0, 7), (5, 6), (1, 4), (2, 3)]
[(0, 3), (5, 4), (2, 6), (1, 7)]
[(0, 3), (5, 4), (2, 7), (1, 6)]
[(0, 3), (5, 6), (2, 4), (1, 7)]
[(0, 3), (5, 6), (2, 7), (1, 4)]
[(0, 3), (5, 7), (2, 4), (1, 6)]
[(0, 3), (5, 7), (2, 6), (1, 4)]
[(0, 4), (5, 3), (2, 6), (1, 7)]
[(0, 4), (5, 3), (2, 7), (1, 6)]
[(0, 4), (5, 6), (2, 3), (1, 7)]
[(0, 4), (5, 6), (2, 7), (1, 3)]
[(0, 4), (5, 7), (2, 3), (1, 6)]
[(0, 4), (5, 7), (2, 6), (1, 3)]
[(0, 6), (5, 3), (2, 4), (1, 7)]
[(0, 6), (5, 3), (2, 7), (1, 4)]
[(0, 6), (5, 4), (2, 3), (1, 7)]
[(0, 6), (5, 4), (2, 7), (1, 3)]
[(0, 6), (5, 7), (2, 3), (1, 4)]
[(0, 6), (5, 7), (2, 4), (1, 3)]
[(0, 7), (5, 3), (2, 4), (1, 6)]
[(0, 7), (5, 3), (2, 6), (1, 4)]
[(0, 7), (5, 4), (2, 3), (1, 6)]
[(0, 7), (5, 4), (2, 6), (1, 3)]
[(0, 7), (5, 6), (2, 3), (1, 4)]
[(0, 7), (5, 6), (2, 4), (1, 3)]
[(1, 3), (0, 4), (2, 6), (5, 7)]
[(1, 3), (0, 4), (2, 7), (5, 6)]
[(1, 3), (0, 6), (2, 4), (5, 7)]
[(1, 3), (0, 6), (2, 7), (5, 4)]
[(1, 3), (0, 7), (2, 4), (5, 6)]
[(1, 3), (0, 7), (2, 6), (5, 4)]
[(1, 4), (0, 3), (2, 6), (5, 7)]
[(1, 4), (0, 3), (2, 7), (5, 6)]
[(1, 4), (0, 6), (2, 3), (5, 7)]
[(1, 4), (0, 6), (2, 7), (5, 3)]
[(1, 4), (0, 7), (2, 3), (5, 6)]
[(1, 4), (0, 7), (2, 6), (5, 3)]
[(1, 6), (0, 3), (2, 4), (5, 7)]
[(1, 6), (0, 3), (2, 7), (5, 4)]
[(1, 6), (0, 4), (2, 3), (5, 7)]
[(1, 6), (0, 4), (2, 7), (5, 3)]
[(1, 6), (0, 7), (2, 3), (5, 4)]
[(1, 6), (0, 7), (2, 4), (5, 3)]
```

```
[(1, 7), (0, 3), (2, 4), (5, 6)]
[(1, 7), (0, 3), (2, 6), (5, 4)]
[(1, 7), (0, 4), (2, 3), (5, 6)]
[(1, 7), (0, 4), (2, 6), (5, 3)]
[(1, 7), (0, 6), (2, 3), (5, 4)]
[(1, 7), (0, 6), (2, 4), (5, 3)]
[(1, 3), (0, 4), (5, 6), (2, 7)]
[(1, 3), (0, 4), (5, 7), (2, 6)]
[(1, 3), (0, 6), (5, 4), (2, 7)]
[(1, 3), (0, 6), (5, 7), (2, 4)]
[(1, 3), (0, 7), (5, 4), (2, 6)]
[(1, 3), (0, 7), (5, 6), (2, 4)]
[(1, 4), (0, 3), (5, 6), (2, 7)]
[(1, 4), (0, 3), (5, 7), (2, 6)]
[(1, 4), (0, 6), (5, 3), (2, 7)]
[(1, 4), (0, 6), (5, 7), (2, 3)]
[(1, 4), (0, 7), (5, 3), (2, 6)]
[(1, 4), (0, 7), (5, 6), (2, 3)]
[(1, 6), (0, 3), (5, 4), (2, 7)]
[(1, 6), (0, 3), (5, 7), (2, 4)]
[(1, 6), (0, 4), (5, 3), (2, 7)]
[(1, 6), (0, 4), (5, 7), (2, 3)]
[(1, 6), (0, 7), (5, 3), (2, 4)]
[(1, 6), (0, 7), (5, 4), (2, 3)]
[(1, 7), (0, 3), (5, 4), (2, 6)]
[(1, 7), (0, 3), (5, 6), (2, 4)]
[(1, 7), (0, 4), (5, 3), (2, 6)]
[(1, 7), (0, 4), (5, 6), (2, 3)]
[(1, 7), (0, 6), (5, 3), (2, 4)]
[(1, 7), (0, 6), (5, 4), (2, 3)]
[(1, 3), (2, 4), (0, 6), (5, 7)]
[(1, 3), (2, 4), (0, 7), (5, 6)]
[(1, 3), (2, 6), (0, 4), (5, 7)]
[(1, 3), (2, 6), (0, 7), (5, 4)]
[(1, 3), (2, 7), (0, 4), (5, 6)]
[(1, 3), (2, 7), (0, 6), (5, 4)]
[(1, 4), (2, 3), (0, 6), (5, 7)]
[(1, 4), (2, 3), (0, 7), (5, 6)]
[(1, 4), (2, 6), (0, 3), (5, 7)]
[(1, 4), (2, 6), (0, 7), (5, 3)]
[(1, 4), (2, 7), (0, 3), (5, 6)]
[(1, 4), (2, 7), (0, 6), (5, 3)]
[(1, 6), (2, 3), (0, 4), (5, 7)]
[(1, 6), (2, 3), (0, 7), (5, 4)]
[(1, 6), (2, 4), (0, 3), (5, 7)]
[(1, 6), (2, 4), (0, 7), (5, 3)]
[(1, 6), (2, 7), (0, 3), (5, 4)]
[(1, 6), (2, 7), (0, 4), (5, 3)]
[(1, 7), (2, 3), (0, 4), (5, 6)]
```

```
[(1, 7), (2, 3), (0, 6), (5, 4)]
[(1, 7), (2, 4), (0, 3), (5, 6)]
[(1, 7), (2, 4), (0, 6), (5, 3)]
[(1, 7), (2, 6), (0, 3), (5, 4)]
[(1, 7), (2, 6), (0, 4), (5, 3)]
[(1, 3), (2, 4), (5, 6), (0, 7)]
[(1, 3), (2, 4), (5, 7), (0, 6)]
[(1, 3), (2, 6), (5, 4), (0, 7)]
[(1, 3), (2, 6), (5, 7), (0, 4)]
[(1, 3), (2, 7), (5, 4), (0, 6)]
[(1, 3), (2, 7), (5, 6), (0, 4)]
[(1, 4), (2, 3), (5, 6), (0, 7)]
[(1, 4), (2, 3), (5, 7), (0, 6)]
[(1, 4), (2, 6), (5, 3), (0, 7)]
[(1, 4), (2, 6), (5, 7), (0, 3)]
[(1, 4), (2, 7), (5, 3), (0, 6)]
[(1, 4), (2, 7), (5, 6), (0, 3)]
[(1, 6), (2, 3), (5, 4), (0, 7)]
[(1, 6), (2, 3), (5, 7), (0, 4)]
[(1, 6), (2, 4), (5, 3), (0, 7)]
[(1, 6), (2, 4), (5, 7), (0, 3)]
[(1, 6), (2, 7), (5, 3), (0, 4)]
[(1, 6), (2, 7), (5, 4), (0, 3)]
[(1, 7), (2, 3), (5, 4), (0, 6)]
[(1, 7), (2, 3), (5, 6), (0, 4)]
[(1, 7), (2, 4), (5, 3), (0, 6)]
[(1, 7), (2, 4), (5, 6), (0, 3)]
[(1, 7), (2, 6), (5, 3), (0, 4)]
[(1, 7), (2, 6), (5, 4), (0, 3)]
[(1, 3), (5, 4), (0, 6), (2, 7)]
[(1, 3), (5, 4), (0, 7), (2, 6)]
[(1, 3), (5, 6), (0, 4), (2, 7)]
[(1, 3), (5, 6), (0, 7), (2, 4)]
[(1, 3), (5, 7), (0, 4), (2, 6)]
[(1, 3), (5, 7), (0, 6), (2, 4)]
[(1, 4), (5, 3), (0, 6), (2, 7)]
[(1, 4), (5, 3), (0, 7), (2, 6)]
[(1, 4), (5, 6), (0, 3), (2, 7)]
[(1, 4), (5, 6), (0, 7), (2, 3)]
[(1, 4), (5, 7), (0, 3), (2, 6)]
[(1, 4), (5, 7), (0, 6), (2, 3)]
[(1, 6), (5, 3), (0, 4), (2, 7)]
[(1, 6), (5, 3), (0, 7), (2, 4)]
[(1, 6), (5, 4), (0, 3), (2, 7)]
[(1, 6), (5, 4), (0, 7), (2, 3)]
[(1, 6), (5, 7), (0, 3), (2, 4)]
[(1, 6), (5, 7), (0, 4), (2, 3)]
[(1, 7), (5, 3), (0, 4), (2, 6)]
[(1, 7), (5, 3), (0, 6), (2, 4)]
```

```
[(1, 7), (5, 4), (0, 3), (2, 6)]
[(1, 7), (5, 4), (0, 6), (2, 3)]
[(1, 7), (5, 6), (0, 3), (2, 4)]
[(1, 7), (5, 6), (0, 4), (2, 3)]
[(1, 3), (5, 4), (2, 6), (0, 7)]
[(1, 3), (5, 4), (2, 7), (0, 6)]
[(1, 3), (5, 6), (2, 4), (0, 7)]
[(1, 3), (5, 6), (2, 7), (0, 4)]
[(1, 3), (5, 7), (2, 4), (0, 6)]
[(1, 3), (5, 7), (2, 6), (0, 4)]
[(1, 4), (5, 3), (2, 6), (0, 7)]
[(1, 4), (5, 3), (2, 7), (0, 6)]
[(1, 4), (5, 6), (2, 3), (0, 7)]
[(1, 4), (5, 6), (2, 7), (0, 3)]
[(1, 4), (5, 7), (2, 3), (0, 6)]
[(1, 4), (5, 7), (2, 6), (0, 3)]
[(1, 6), (5, 3), (2, 4), (0, 7)]
[(1, 6), (5, 3), (2, 7), (0, 4)]
[(1, 6), (5, 4), (2, 3), (0, 7)]
[(1, 6), (5, 4), (2, 7), (0, 3)]
[(1, 6), (5, 7), (2, 3), (0, 4)]
[(1, 6), (5, 7), (2, 4), (0, 3)]
[(1, 7), (5, 3), (2, 4), (0, 6)]
[(1, 7), (5, 3), (2, 6), (0, 4)]
[(1, 7), (5, 4), (2, 3), (0, 6)]
[(1, 7), (5, 4), (2, 6), (0, 3)]
[(1, 7), (5, 6), (2, 3), (0, 4)]
[(1, 7), (5, 6), (2, 4), (0, 3)]
[(2, 3), (0, 4), (1, 6), (5, 7)]
[(2, 3), (0, 4), (1, 7), (5, 6)]
[(2, 3), (0, 6), (1, 4), (5, 7)]
[(2, 3), (0, 6), (1, 7), (5, 4)]
[(2, 3), (0, 7), (1, 4), (5, 6)]
[(2, 3), (0, 7), (1, 6), (5, 4)]
[(2, 4), (0, 3), (1, 6), (5, 7)]
[(2, 4), (0, 3), (1, 7), (5, 6)]
[(2, 4), (0, 6), (1, 3), (5, 7)]
[(2, 4), (0, 6), (1, 7), (5, 3)]
[(2, 4), (0, 7), (1, 3), (5, 6)]
[(2, 4), (0, 7), (1, 6), (5, 3)]
[(2, 6), (0, 3), (1, 4), (5, 7)]
[(2, 6), (0, 3), (1, 7), (5, 4)]
[(2, 6), (0, 4), (1, 3), (5, 7)]
[(2, 6), (0, 4), (1, 7), (5, 3)]
[(2, 6), (0, 7), (1, 3), (5, 4)]
[(2, 6), (0, 7), (1, 4), (5, 3)]
[(2, 7), (0, 3), (1, 4), (5, 6)]
[(2, 7), (0, 3), (1, 6), (5, 4)]
[(2, 7), (0, 4), (1, 3), (5, 6)]
```

```
[(2, 7), (0, 4), (1, 6), (5, 3)]
[(2, 7), (0, 6), (1, 3), (5, 4)]
[(2, 7), (0, 6), (1, 4), (5, 3)]
[(2, 3), (0, 4), (5, 6), (1, 7)]
[(2, 3), (0, 4), (5, 7), (1, 6)]
[(2, 3), (0, 6), (5, 4), (1, 7)]
[(2, 3), (0, 6), (5, 7), (1, 4)]
[(2, 3), (0, 7), (5, 4), (1, 6)]
[(2, 3), (0, 7), (5, 6), (1, 4)]
[(2, 4), (0, 3), (5, 6), (1, 7)]
[(2, 4), (0, 3), (5, 7), (1, 6)]
[(2, 4), (0, 6), (5, 3), (1, 7)]
[(2, 4), (0, 6), (5, 7), (1, 3)]
[(2, 4), (0, 7), (5, 3), (1, 6)]
[(2, 4), (0, 7), (5, 6), (1, 3)]
[(2, 6), (0, 3), (5, 4), (1, 7)]
[(2, 6), (0, 3), (5, 7), (1, 4)]
[(2, 6), (0, 4), (5, 3), (1, 7)]
[(2, 6), (0, 4), (5, 7), (1, 3)]
[(2, 6), (0, 7), (5, 3), (1, 4)]
[(2, 6), (0, 7), (5, 4), (1, 3)]
[(2, 7), (0, 3), (5, 4), (1, 6)]
[(2, 7), (0, 3), (5, 6), (1, 4)]
[(2, 7), (0, 4), (5, 3), (1, 6)]
[(2, 7), (0, 4), (5, 6), (1, 3)]
[(2, 7), (0, 6), (5, 3), (1, 4)]
[(2, 7), (0, 6), (5, 4), (1, 3)]
[(2, 3), (1, 4), (0, 6), (5, 7)]
[(2, 3), (1, 4), (0, 7), (5, 6)]
[(2, 3), (1, 6), (0, 4), (5, 7)]
[(2, 3), (1, 6), (0, 7), (5, 4)]
[(2, 3), (1, 7), (0, 4), (5, 6)]
[(2, 3), (1, 7), (0, 6), (5, 4)]
[(2, 4), (1, 3), (0, 6), (5, 7)]
[(2, 4), (1, 3), (0, 7), (5, 6)]
[(2, 4), (1, 6), (0, 3), (5, 7)]
[(2, 4), (1, 6), (0, 7), (5, 3)]
[(2, 4), (1, 7), (0, 3), (5, 6)]
[(2, 4), (1, 7), (0, 6), (5, 3)]
[(2, 6), (1, 3), (0, 4), (5, 7)]
[(2, 6), (1, 3), (0, 7), (5, 4)]
[(2, 6), (1, 4), (0, 3), (5, 7)]
[(2, 6), (1, 4), (0, 7), (5, 3)]
[(2, 6), (1, 7), (0, 3), (5, 4)]
[(2, 6), (1, 7), (0, 4), (5, 3)]
[(2, 7), (1, 3), (0, 4), (5, 6)]
[(2, 7), (1, 3), (0, 6), (5, 4)]
[(2, 7), (1, 4), (0, 3), (5, 6)]
[(2, 7), (1, 4), (0, 6), (5, 3)]
```

```
[(2, 7), (1, 6), (0, 3), (5, 4)]
[(2, 7), (1, 6), (0, 4), (5, 3)]
[(2, 3), (1, 4), (5, 6), (0, 7)]
[(2, 3), (1, 4), (5, 7), (0, 6)]
[(2, 3), (1, 6), (5, 4), (0, 7)]
[(2, 3), (1, 6), (5, 7), (0, 4)]
[(2, 3), (1, 7), (5, 4), (0, 6)]
[(2, 3), (1, 7), (5, 6), (0, 4)]
[(2, 4), (1, 3), (5, 6), (0, 7)]
[(2, 4), (1, 3), (5, 7), (0, 6)]
[(2, 4), (1, 6), (5, 3), (0, 7)]
[(2, 4), (1, 6), (5, 7), (0, 3)]
[(2, 4), (1, 7), (5, 3), (0, 6)]
[(2, 4), (1, 7), (5, 6), (0, 3)]
[(2, 6), (1, 3), (5, 4), (0, 7)]
[(2, 6), (1, 3), (5, 7), (0, 4)]
[(2, 6), (1, 4), (5, 3), (0, 7)]
[(2, 6), (1, 4), (5, 7), (0, 3)]
[(2, 6), (1, 7), (5, 3), (0, 4)]
[(2, 6), (1, 7), (5, 4), (0, 3)]
[(2, 7), (1, 3), (5, 4), (0, 6)]
[(2, 7), (1, 3), (5, 6), (0, 4)]
[(2, 7), (1, 4), (5, 3), (0, 6)]
[(2, 7), (1, 4), (5, 6), (0, 3)]
[(2, 7), (1, 6), (5, 3), (0, 4)]
[(2, 7), (1, 6), (5, 4), (0, 3)]
[(2, 3), (5, 4), (0, 6), (1, 7)]
[(2, 3), (5, 4), (0, 7), (1, 6)]
[(2, 3), (5, 6), (0, 4), (1, 7)]
[(2, 3), (5, 6), (0, 7), (1, 4)]
[(2, 3), (5, 7), (0, 4), (1, 6)]
[(2, 3), (5, 7), (0, 6), (1, 4)]
[(2, 4), (5, 3), (0, 6), (1, 7)]
[(2, 4), (5, 3), (0, 7), (1, 6)]
[(2, 4), (5, 6), (0, 3), (1, 7)]
[(2, 4), (5, 6), (0, 7), (1, 3)]
[(2, 4), (5, 7), (0, 3), (1, 6)]
[(2, 4), (5, 7), (0, 6), (1, 3)]
[(2, 6), (5, 3), (0, 4), (1, 7)]
[(2, 6), (5, 3), (0, 7), (1, 4)]
[(2, 6), (5, 4), (0, 3), (1, 7)]
[(2, 6), (5, 4), (0, 7), (1, 3)]
[(2, 6), (5, 7), (0, 3), (1, 4)]
[(2, 6), (5, 7), (0, 4), (1, 3)]
[(2, 7), (5, 3), (0, 4), (1, 6)]
[(2, 7), (5, 3), (0, 6), (1, 4)]
[(2, 7), (5, 4), (0, 3), (1, 6)]
[(2, 7), (5, 4), (0, 6), (1, 3)]
[(2, 7), (5, 6), (0, 3), (1, 4)]
```

```
[(2, 7), (5, 6), (0, 4), (1, 3)]
[(2, 3), (5, 4), (1, 6), (0, 7)]
[(2, 3), (5, 4), (1, 7), (0, 6)]
[(2, 3), (5, 6), (1, 4), (0, 7)]
[(2, 3), (5, 6), (1, 7), (0, 4)]
[(2, 3), (5, 7), (1, 4), (0, 6)]
[(2, 3), (5, 7), (1, 6), (0, 4)]
[(2, 4), (5, 3), (1, 6), (0, 7)]
[(2, 4), (5, 3), (1, 7), (0, 6)]
[(2, 4), (5, 6), (1, 3), (0, 7)]
[(2, 4), (5, 6), (1, 7), (0, 3)]
[(2, 4), (5, 7), (1, 3), (0, 6)]
[(2, 4), (5, 7), (1, 6), (0, 3)]
[(2, 6), (5, 3), (1, 4), (0, 7)]
[(2, 6), (5, 3), (1, 7), (0, 4)]
[(2, 6), (5, 4), (1, 3), (0, 7)]
[(2, 6), (5, 4), (1, 7), (0, 3)]
[(2, 6), (5, 7), (1, 3), (0, 4)]
[(2, 6), (5, 7), (1, 4), (0, 3)]
[(2, 7), (5, 3), (1, 4), (0, 6)]
[(2, 7), (5, 3), (1, 6), (0, 4)]
[(2, 7), (5, 4), (1, 3), (0, 6)]
[(2, 7), (5, 4), (1, 6), (0, 3)]
[(2, 7), (5, 6), (1, 3), (0, 4)]
[(2, 7), (5, 6), (1, 4), (0, 3)]
[(5, 3), (0, 4), (1, 6), (2, 7)]
[(5, 3), (0, 4), (1, 7), (2, 6)]
[(5, 3), (0, 6), (1, 4), (2, 7)]
[(5, 3), (0, 6), (1, 7), (2, 4)]
[(5, 3), (0, 7), (1, 4), (2, 6)]
[(5, 3), (0, 7), (1, 6), (2, 4)]
[(5, 4), (0, 3), (1, 6), (2, 7)]
[(5, 4), (0, 3), (1, 7), (2, 6)]
[(5, 4), (0, 6), (1, 3), (2, 7)]
[(5, 4), (0, 6), (1, 7), (2, 3)]
[(5, 4), (0, 7), (1, 3), (2, 6)]
[(5, 4), (0, 7), (1, 6), (2, 3)]
[(5, 6), (0, 3), (1, 4), (2, 7)]
[(5, 6), (0, 3), (1, 7), (2, 4)]
[(5, 6), (0, 4), (1, 3), (2, 7)]
[(5, 6), (0, 4), (1, 7), (2, 3)]
[(5, 6), (0, 7), (1, 3), (2, 4)]
[(5, 6), (0, 7), (1, 4), (2, 3)]
[(5, 7), (0, 3), (1, 4), (2, 6)]
[(5, 7), (0, 3), (1, 6), (2, 4)]
[(5, 7), (0, 4), (1, 3), (2, 6)]
[(5, 7), (0, 4), (1, 6), (2, 3)]
[(5, 7), (0, 6), (1, 3), (2, 4)]
[(5, 7), (0, 6), (1, 4), (2, 3)]
```

```
[(5, 3), (0, 4), (2, 6), (1, 7)]
[(5, 3), (0, 4), (2, 7), (1, 6)]
[(5, 3), (0, 6), (2, 4), (1, 7)]
[(5, 3), (0, 6), (2, 7), (1, 4)]
[(5, 3), (0, 7), (2, 4), (1, 6)]
[(5, 3), (0, 7), (2, 6), (1, 4)]
[(5, 4), (0, 3), (2, 6), (1, 7)]
[(5, 4), (0, 3), (2, 7), (1, 6)]
[(5, 4), (0, 6), (2, 3), (1, 7)]
[(5, 4), (0, 6), (2, 7), (1, 3)]
[(5, 4), (0, 7), (2, 3), (1, 6)]
[(5, 4), (0, 7), (2, 6), (1, 3)]
[(5, 6), (0, 3), (2, 4), (1, 7)]
[(5, 6), (0, 3), (2, 7), (1, 4)]
[(5, 6), (0, 4), (2, 3), (1, 7)]
[(5, 6), (0, 4), (2, 7), (1, 3)]
[(5, 6), (0, 7), (2, 3), (1, 4)]
[(5, 6), (0, 7), (2, 4), (1, 3)]
[(5, 7), (0, 3), (2, 4), (1, 6)]
[(5, 7), (0, 3), (2, 6), (1, 4)]
[(5, 7), (0, 4), (2, 3), (1, 6)]
[(5, 7), (0, 4), (2, 6), (1, 3)]
[(5, 7), (0, 6), (2, 3), (1, 4)]
[(5, 7), (0, 6), (2, 4), (1, 3)]
[(5, 3), (1, 4), (0, 6), (2, 7)]
[(5, 3), (1, 4), (0, 7), (2, 6)]
[(5, 3), (1, 6), (0, 4), (2, 7)]
[(5, 3), (1, 6), (0, 7), (2, 4)]
[(5, 3), (1, 7), (0, 4), (2, 6)]
[(5, 3), (1, 7), (0, 6), (2, 4)]
[(5, 4), (1, 3), (0, 6), (2, 7)]
[(5, 4), (1, 3), (0, 7), (2, 6)]
[(5, 4), (1, 6), (0, 3), (2, 7)]
[(5, 4), (1, 6), (0, 7), (2, 3)]
[(5, 4), (1, 7), (0, 3), (2, 6)]
[(5, 4), (1, 7), (0, 6), (2, 3)]
[(5, 6), (1, 3), (0, 4), (2, 7)]
[(5, 6), (1, 3), (0, 7), (2, 4)]
[(5, 6), (1, 4), (0, 3), (2, 7)]
[(5, 6), (1, 4), (0, 7), (2, 3)]
[(5, 6), (1, 7), (0, 3), (2, 4)]
[(5, 6), (1, 7), (0, 4), (2, 3)]
[(5, 7), (1, 3), (0, 4), (2, 6)]
[(5, 7), (1, 3), (0, 6), (2, 4)]
[(5, 7), (1, 4), (0, 3), (2, 6)]
[(5, 7), (1, 4), (0, 6), (2, 3)]
[(5, 7), (1, 6), (0, 3), (2, 4)]
[(5, 7), (1, 6), (0, 4), (2, 3)]
[(5, 3), (1, 4), (2, 6), (0, 7)]
[(5, 3), (1, 4), (2, 7), (0, 6)]
```

```
[(5, 3), (1, 6), (2, 4), (0, 7)]
[(5, 3), (1, 6), (2, 7), (0, 4)]
[(5, 3), (1, 7), (2, 4), (0, 6)]
[(5, 3), (1, 7), (2, 6), (0, 4)]
[(5, 4), (1, 3), (2, 6), (0, 7)]
[(5, 4), (1, 3), (2, 7), (0, 6)]
[(5, 4), (1, 6), (2, 3), (0, 7)]
[(5, 4), (1, 6), (2, 7), (0, 3)]
[(5, 4), (1, 7), (2, 3), (0, 6)]
[(5, 4), (1, 7), (2, 6), (0, 3)]
[(5, 6), (1, 3), (2, 4), (0, 7)]
[(5, 6), (1, 3), (2, 7), (0, 4)]
[(5, 6), (1, 4), (2, 3), (0, 7)]
[(5, 6), (1, 4), (2, 7), (0, 3)]
[(5, 6), (1, 7), (2, 3), (0, 4)]
[(5, 6), (1, 7), (2, 4), (0, 3)]
[(5, 7), (1, 3), (2, 4), (0, 6)]
[(5, 7), (1, 3), (2, 6), (0, 4)]
[(5, 7), (1, 4), (2, 3), (0, 6)]
[(5, 7), (1, 4), (2, 6), (0, 3)]
[(5, 7), (1, 6), (2, 3), (0, 4)]
[(5, 7), (1, 6), (2, 4), (0, 3)]
[(5, 3), (2, 4), (0, 6), (1, 7)]
[(5, 3), (2, 4), (0, 7), (1, 6)]
[(5, 3), (2, 6), (0, 4), (1, 7)]
[(5, 3), (2, 6), (0, 7), (1, 4)]
[(5, 3), (2, 7), (0, 4), (1, 6)]
[(5, 3), (2, 7), (0, 6), (1, 4)]
[(5, 4), (2, 3), (0, 6), (1, 7)]
[(5, 4), (2, 3), (0, 7), (1, 6)]
[(5, 4), (2, 6), (0, 3), (1, 7)]
[(5, 4), (2, 6), (0, 7), (1, 3)]
[(5, 4), (2, 7), (0, 3), (1, 6)]
[(5, 4), (2, 7), (0, 6), (1, 3)]
[(5, 6), (2, 3), (0, 4), (1, 7)]
[(5, 6), (2, 3), (0, 7), (1, 4)]
[(5, 6), (2, 4), (0, 3), (1, 7)]
[(5, 6), (2, 4), (0, 7), (1, 3)]
[(5, 6), (2, 7), (0, 3), (1, 4)]
[(5, 6), (2, 7), (0, 4), (1, 3)]
[(5, 7), (2, 3), (0, 4), (1, 6)]
[(5, 7), (2, 3), (0, 6), (1, 4)]
[(5, 7), (2, 4), (0, 3), (1, 6)]
[(5, 7), (2, 4), (0, 6), (1, 3)]
[(5, 7), (2, 6), (0, 3), (1, 4)]
[(5, 7), (2, 6), (0, 4), (1, 3)]
[(5, 3), (2, 4), (1, 6), (0, 7)]
[(5, 3), (2, 4), (1, 7), (0, 6)]
[(5, 3), (2, 6), (1, 4), (0, 7)]
```

```
[(5, 3), (2, 6), (1, 7), (0, 4)]
[(5, 3), (2, 7), (1, 4), (0, 6)]
[(5, 3), (2, 7), (1, 6), (0, 4)]
[(5, 4), (2, 3), (1, 6), (0, 7)]
[(5, 4), (2, 3), (1, 7), (0, 6)]
[(5, 4), (2, 6), (1, 3), (0, 7)]
[(5, 4), (2, 6), (1, 7), (0, 3)]
[(5, 4), (2, 7), (1, 3), (0, 6)]
[(5, 4), (2, 7), (1, 6), (0, 3)]
[(5, 6), (2, 3), (1, 4), (0, 7)]
[(5, 6), (2, 3), (1, 7), (0, 4)]
[(5, 6), (2, 4), (1, 3), (0, 7)]
[(5, 6), (2, 4), (1, 7), (0, 3)]
[(5, 6), (2, 7), (1, 3), (0, 4)]
[(5, 6), (2, 7), (1, 4), (0, 3)]
[(5, 7), (2, 3), (1, 4), (0, 6)]
[(5, 7), (2, 3), (1, 6), (0, 4)]
[(5, 7), (2, 4), (1, 3), (0, 6)]
[(5, 7), (2, 4), (1, 6), (0, 3)]
[(5, 7), (2, 6), (1, 3), (0, 4)]
[(5, 7), (2, 6), (1, 4), (0, 3)]
The number of equivalent matchings is :  576
```

2. Execute the following code cell to create your matching. You must replace the number "123456789" with your 9-digit student ID number.

```
from project import create_partial_matching

#Replace "123456789" in the following line with your 9-digit student
ID number
my_partial_matching=create_partial_matching(123456789)

#Replace "123456789" in the previous Line with your 9-digit student ID
number
```

3. For your personal partial matching, print out the following information:

- **a list of the pairs with missing data**
- **a list of all missing numbers**
- **the number of ways the partial matching can be completed to a valid (1) matching (but not the completed matchings themselves).**

Note: Do not print out anything else.

```
#
```