**DAA Lab 4**

Name: Sarakshi Mamodia

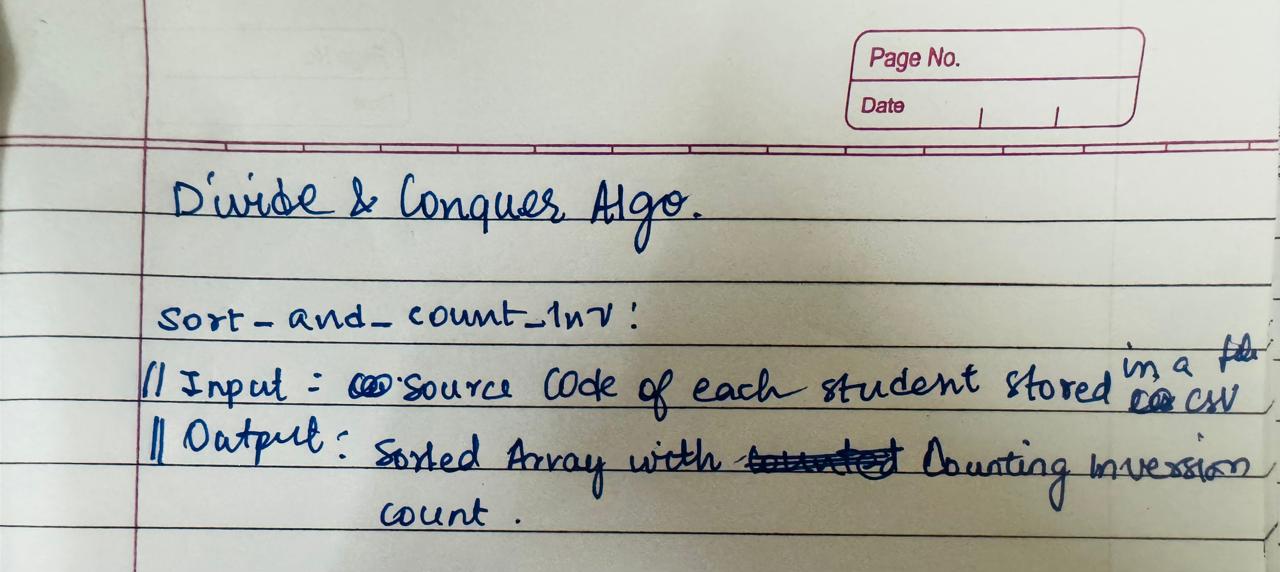
Reg No: 231071056

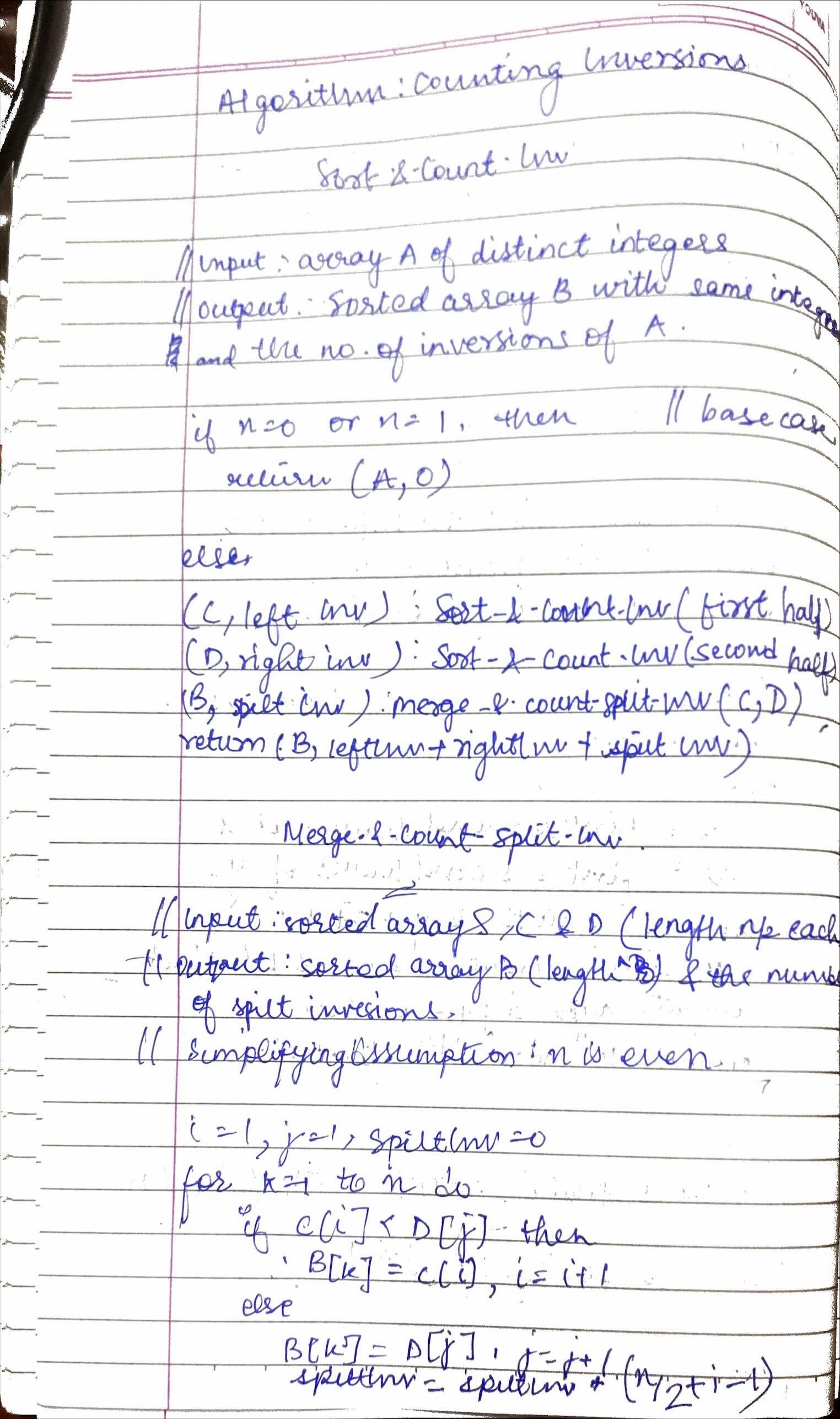
**Aim**:

Task 1: Consider first/second year course-code choices of 100 students.

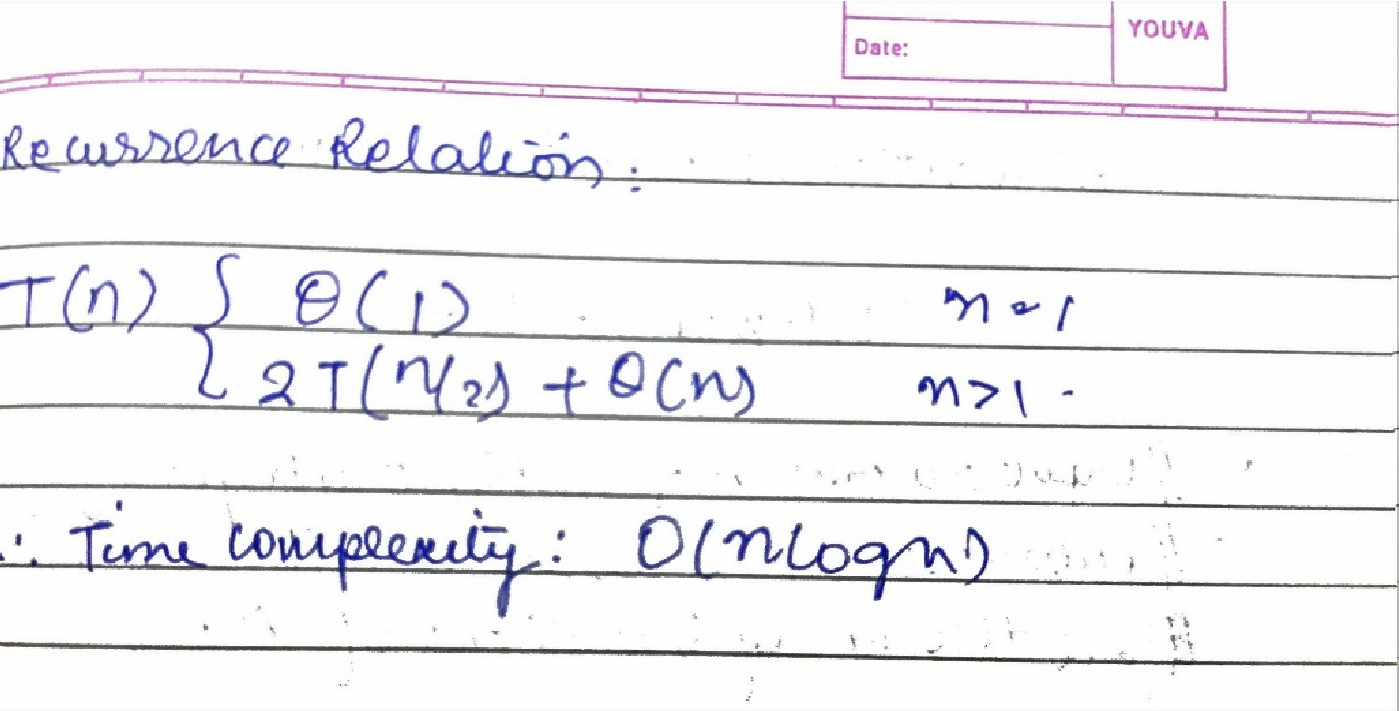
* Find inversion count of these choices.
* Find students with zero, one, two, three inversion counts comment on your result.

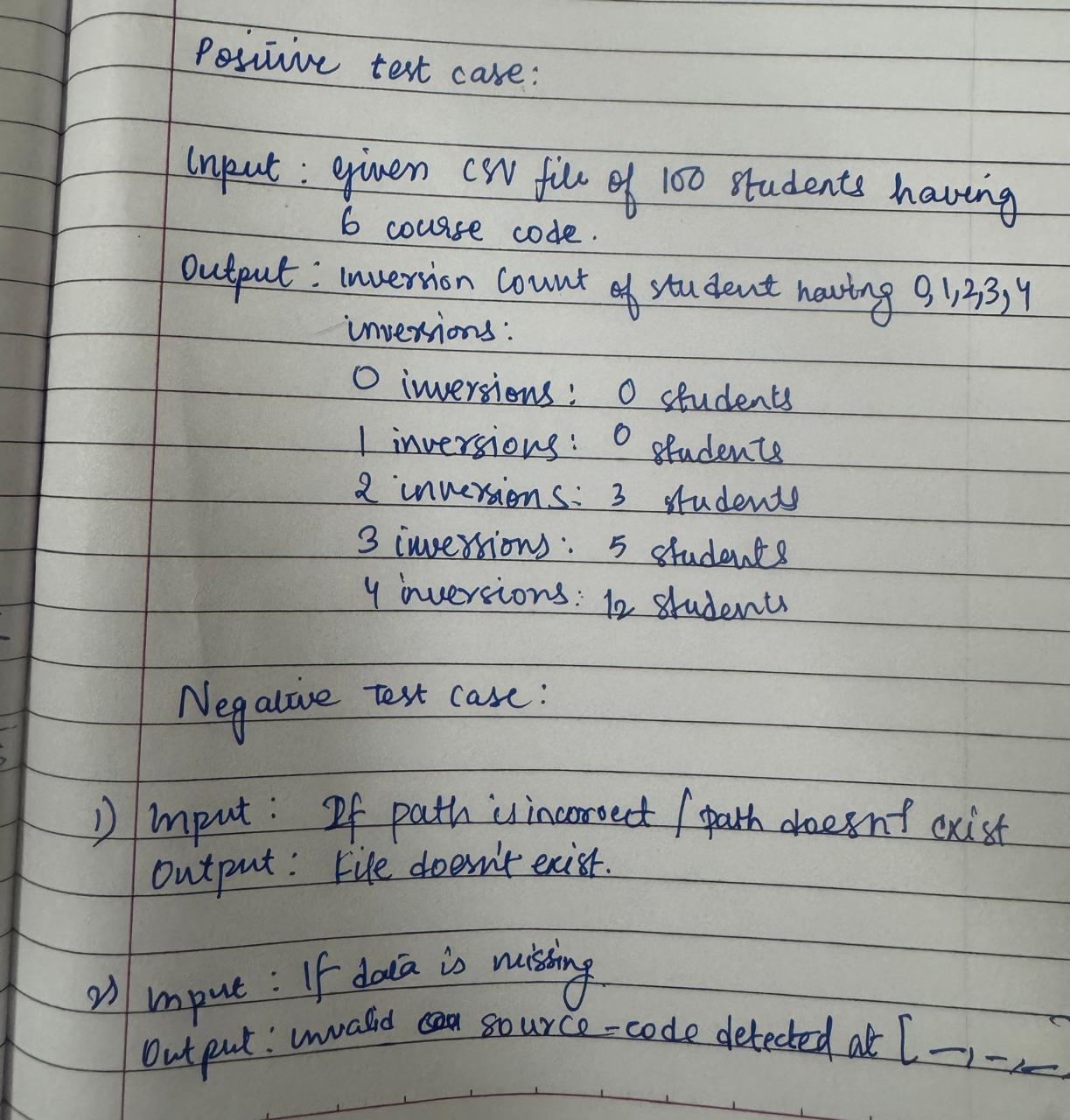
**Algorithm:**

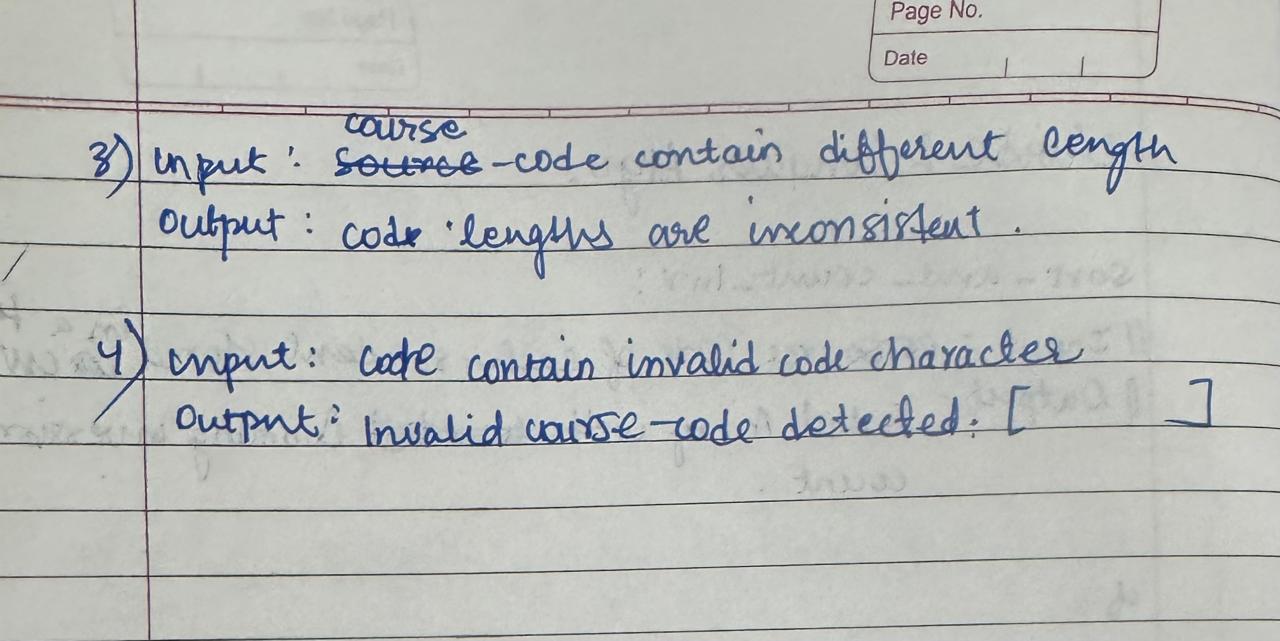




**Time Complexity:**

**Test Cases:**





**Code:**

import pandas as pd

class InversionCount:

    def \_\_init\_\_(s):

        s.df = None

    def get\_data(s, filePath):

        if not s.file\_existence(filePath):

            return None

        s.df = pd.read\_csv(filePath)

        return s.df

    def file\_existence(s, filepath):

        try:

            pd.read\_csv(filepath)

            return True

        except FileNotFoundError:

            print("File does not exist.")

            return False

    def validate\_courses(s, courses):

        # Check if all course codes are integers and non-negative

        if not all(isinstance(course, int) and course >= 0 for course in courses):

            print(f"Invalid course codes detected: {courses}")

            return False

        return True

    def check\_course\_length(s, courses):

        # Check if all course codes have the same length

        course\_lengths = [len(str(course)) for course in courses]

        if len(set(course\_lengths)) > 1:

            print(f"Course lengths are inconsistent: {courses}")

            return False

        return True

    def combine\_count(s, l, r):

        combined = []

        larr = rarr = countI = 0

        while larr < len(l) and rarr < len(r):

            if l[larr] <= r[rarr]:

                combined.append(l[larr])

                larr += 1

            else:

                combined.append(r[rarr])

                countI += len(l) - larr

                rarr += 1

        combined.extend(l[larr:])

        combined.extend(r[rarr:])

        return combined, countI

    def count\_inversion(s, arr):

        if len(arr) < 2:

            return arr, 0

        mid = len(arr) // 2

        left, Lcount = s.count\_inversion(arr[:mid])

        right, Rcount = s.count\_inversion(arr[mid:])

        combined, split\_count = s.combine\_count(left, right)

        return combined, Lcount + Rcount + split\_count

    def inversion\_count(s):

        if s.df is None:

            print("Dataframe is empty. Load data first.")

            return

        inversion\_counts = []

        for i, row in s.df.iterrows():

            courses = row[['course1', 'course2', 'course3', 'course4', 'course5', 'course6']].tolist()

            # Check if the courses are valid

            if not s.validate\_courses(courses):

                inversion\_counts.append(None)

                continue

            # Check for inconsistent course lengths

            if not s.check\_course\_length(courses):

                inversion\_counts.append(None)

                continue

            \_, count = s.count\_inversion(courses)

            inversion\_counts.append(count)

        s.df['inversion\_count'] = inversion\_counts

        inversion\_summary = s.df['inversion\_count'].value\_counts().reindex(range(5), fill\_value=0)

        print("Inversion Count Summary:")

        for count in range(5):

            print(f"{count} inversion(s): {inversion\_summary.get(count, 0)} student(s)")

# Usage

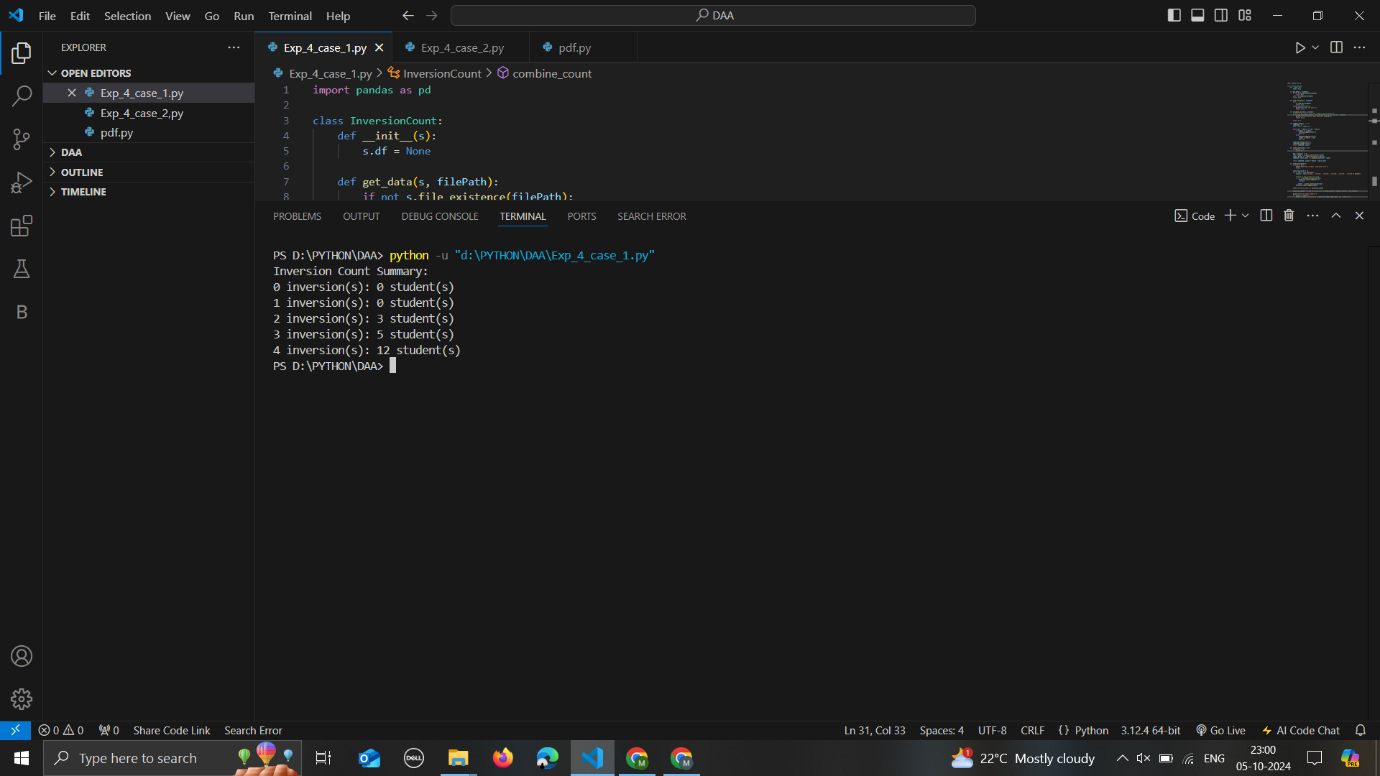
ca = InversionCount()

ca.get\_data("D:\\PYTHON\\DAA\\updated\_student\_course\_choices\_5\_digit-1.csv")

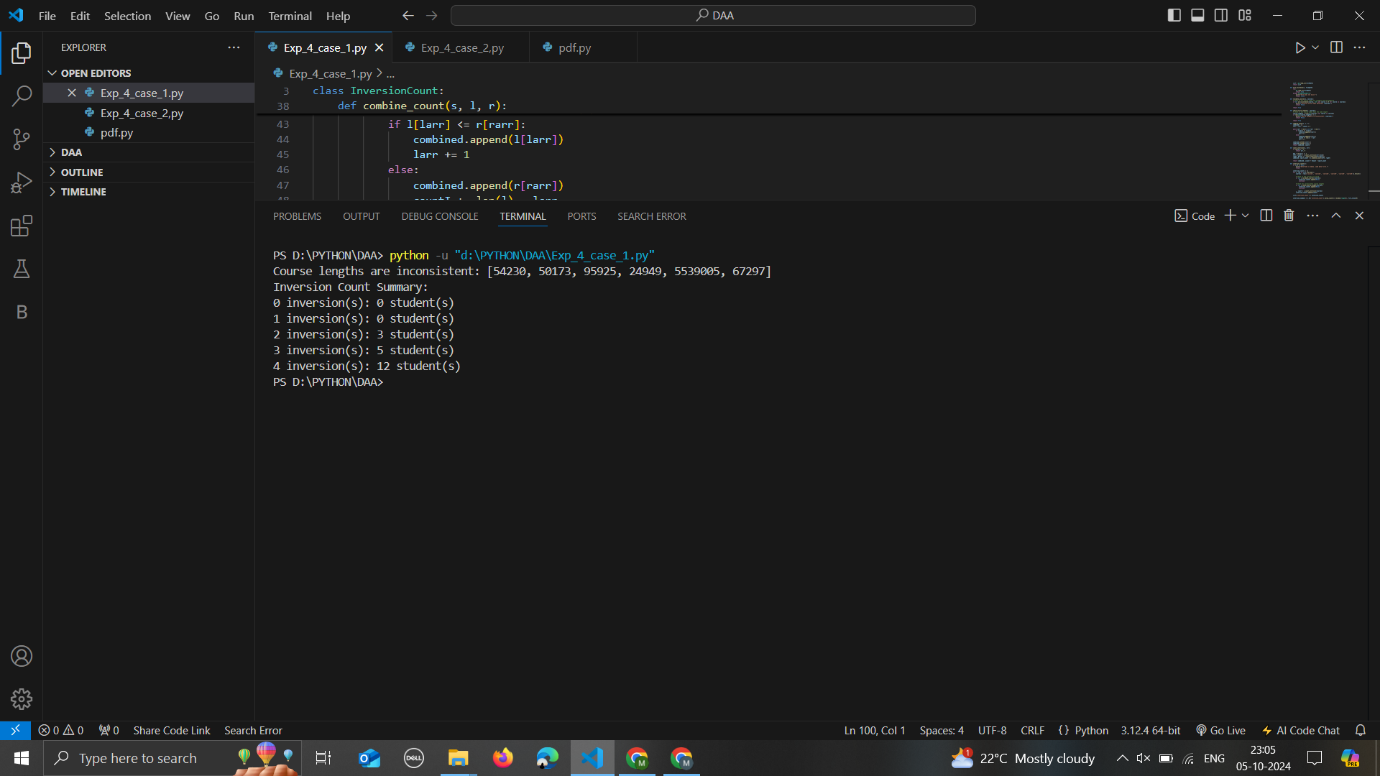
ca.inversion\_count()

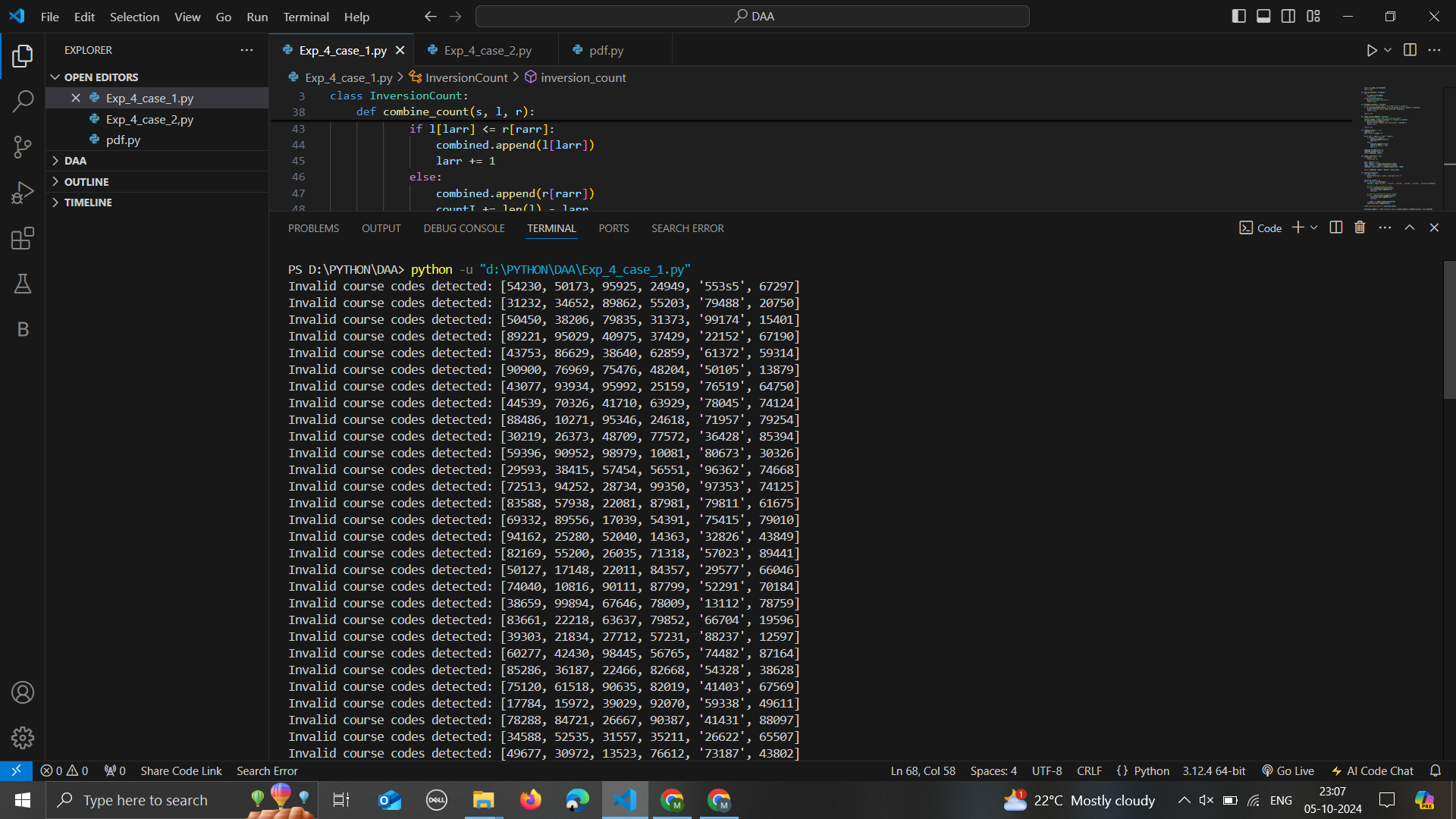
**Output:**

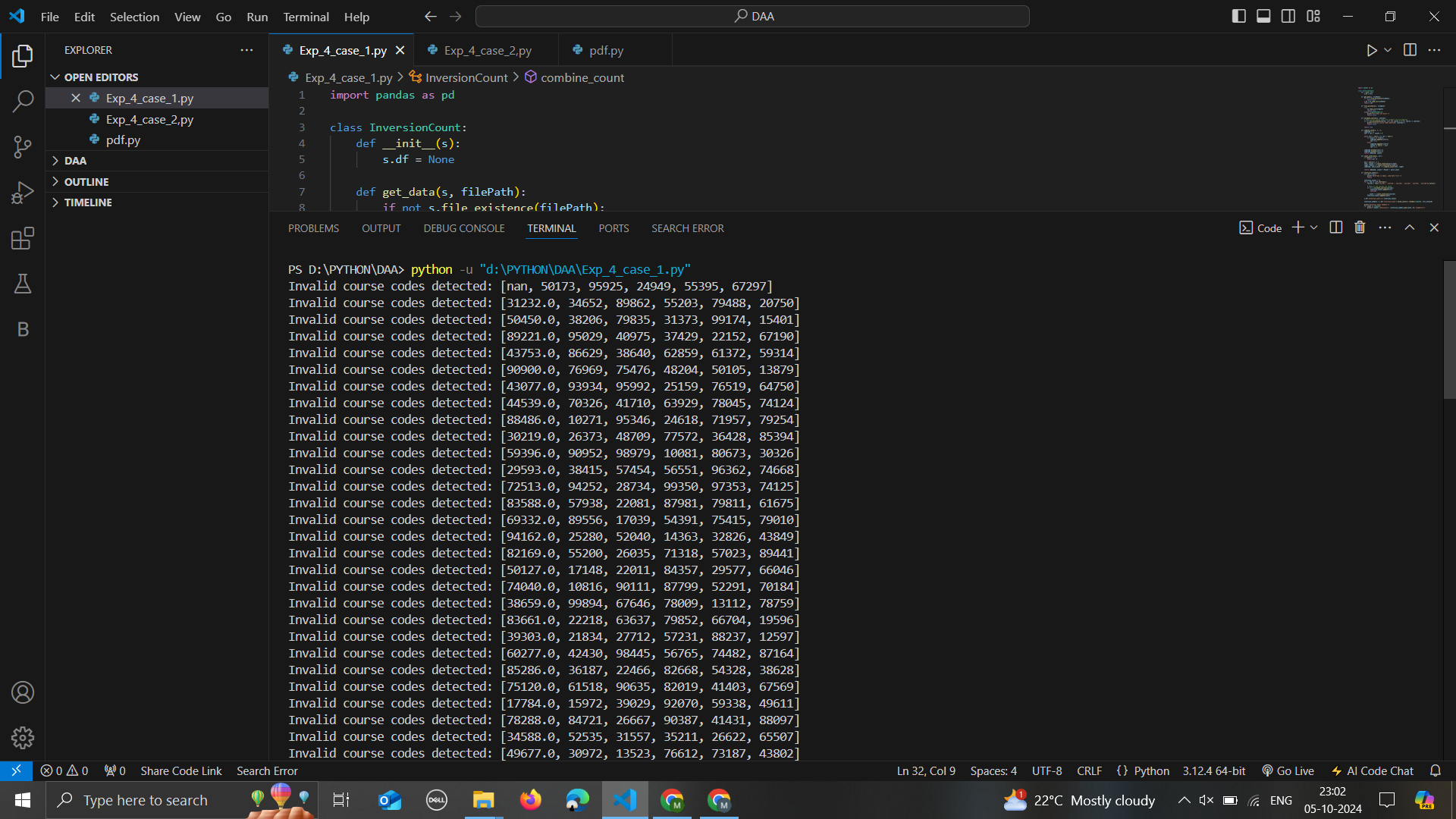
**Positive Case:**

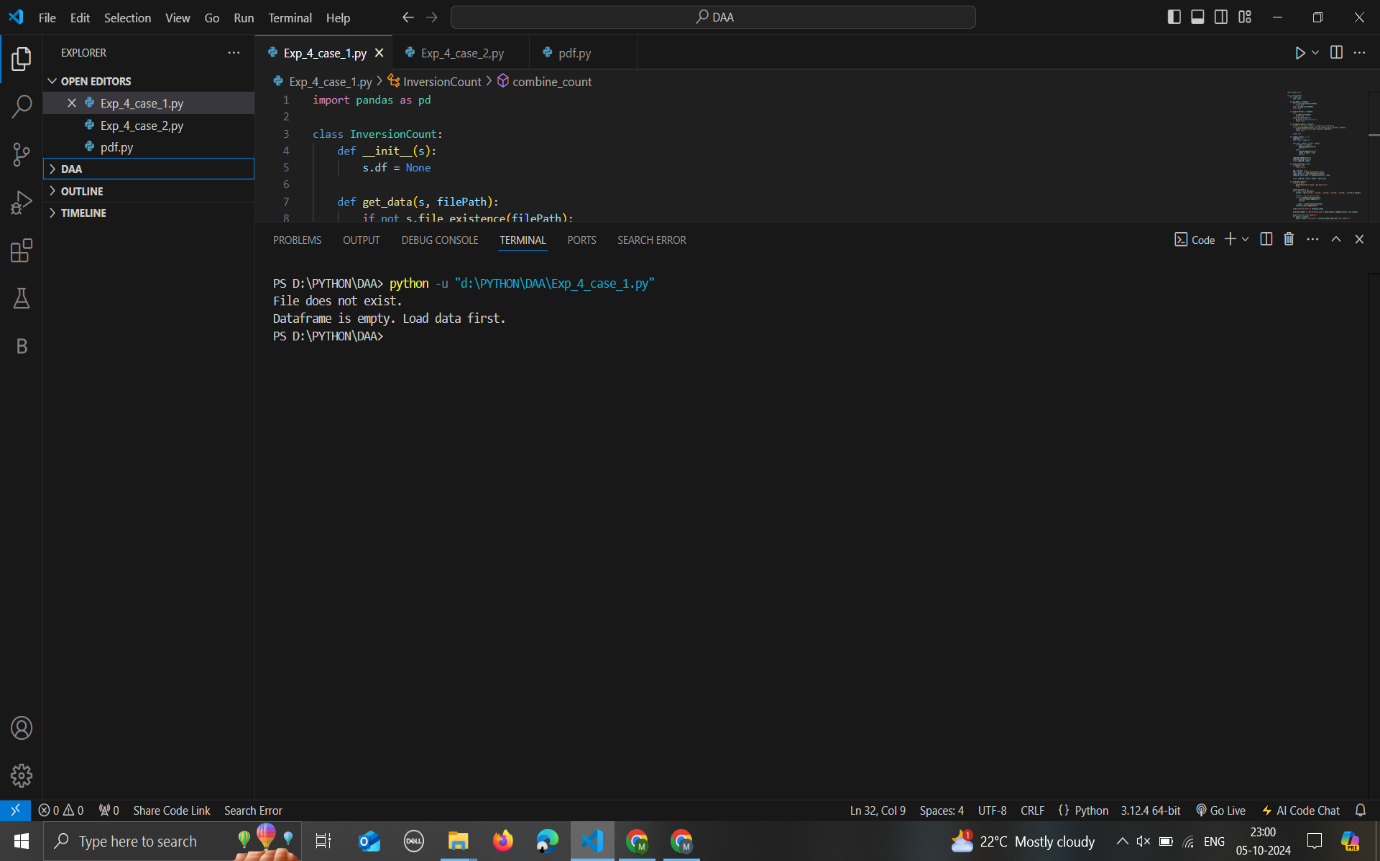


**Negative Case:**





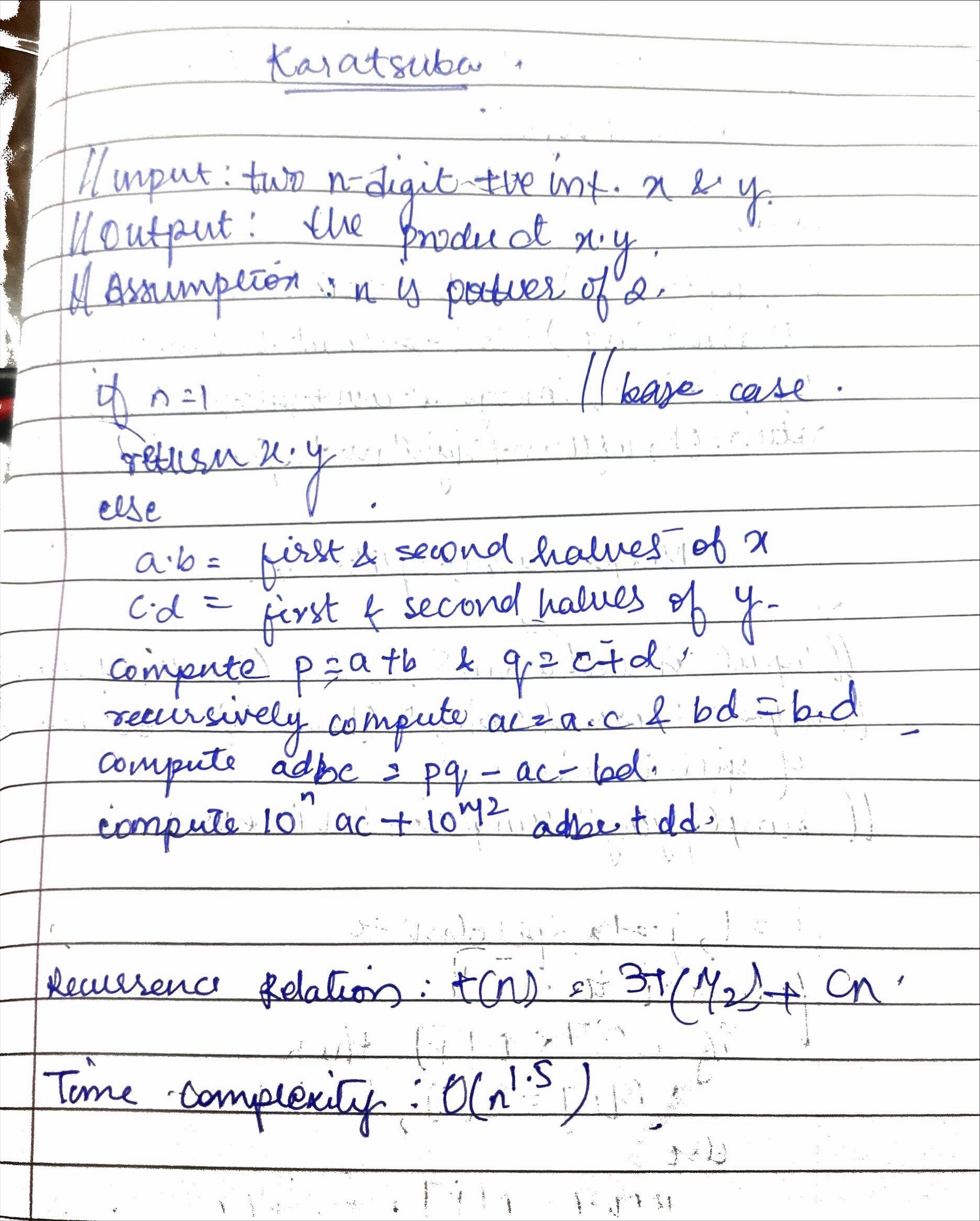




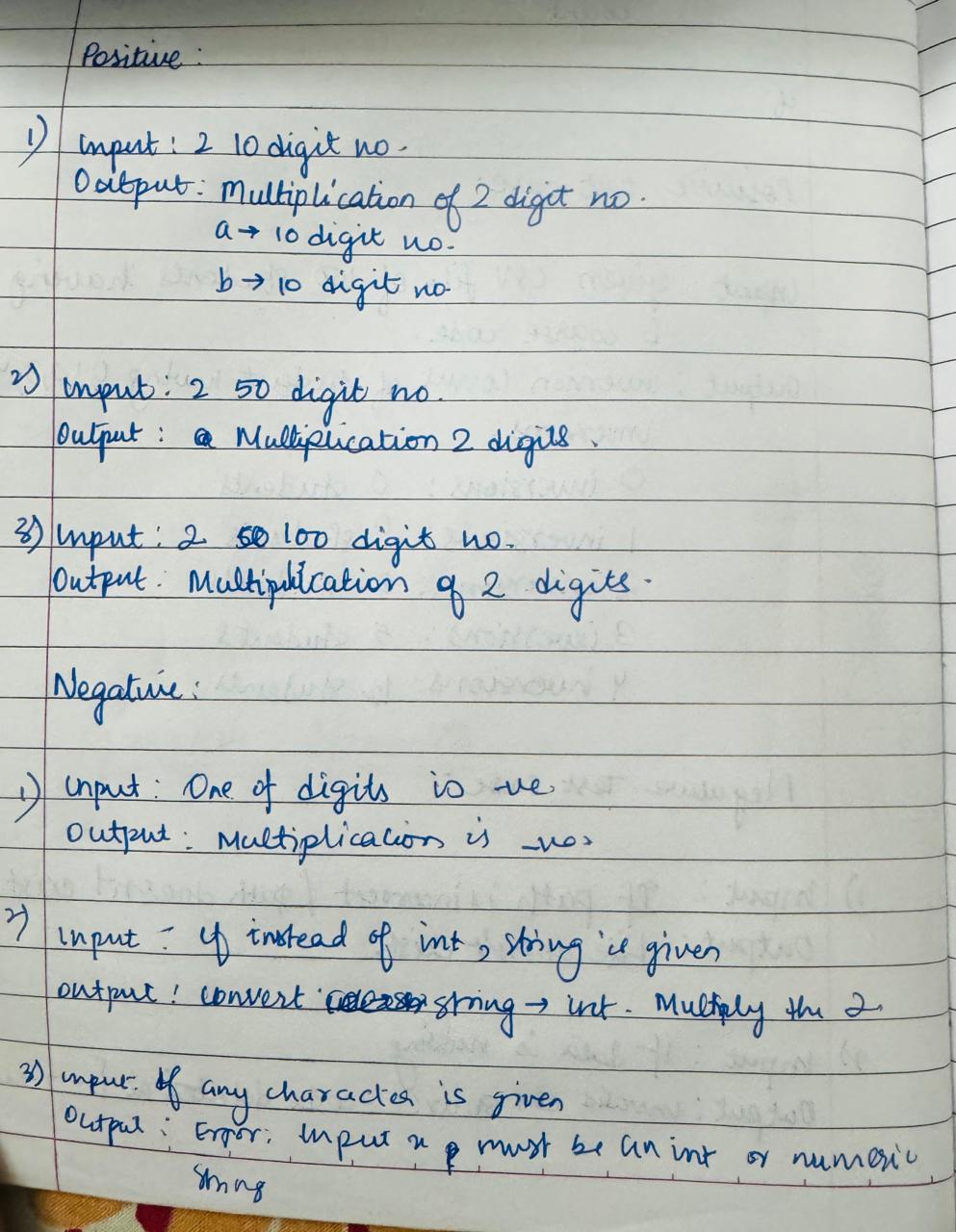
Task 2: Consider large integers of size 10, 50, 100, 500 and 1000 digits.

* Write integer multiplication program
* Write integer multiplication program using divide and conquer technique.

**Algorithm:**



**Test Cases:**



Code:

def multiply(a, b):

    return a \* b

def karatsuba(x, y):

    if isinstance(x, str):

        try:

            x = int(x)  #N\_case-1

        except ValueError:

            raise ValueError("Input x must be an integer or a numeric string.")

    if isinstance(y, str):

        try:

            y = int(y) #N\_case-1

        except ValueError:

            raise ValueError("Input y must be an integer or a numeric string.")

    #N\_case-2

    if x < 0 and y < 0:

        return karatsuba(-x, -y)

    elif x < 0:

        return -karatsuba(-x, y)

    elif y < 0:

        return -karatsuba(x, -y)

    # Base case

    if x < 10 or y < 10:

        return x \* y

    n = max(len(str(x)), len(str(y)))

    # Divide step

    a, b = divmod(x, 10\*\*(n // 2))

    c, d = divmod(y, 10\*\*(n // 2))

    # Recursively call

    ac = karatsuba(a, c)

    bd = karatsuba(b, d)

    ad\_bc = karatsuba(a + b, c + d) - ac - bd

    # Combine the results

    product = ac \* 10\*(2 \* (n // 2)) + (ad\_bc \* 10\*(n // 2)) + bd

    return product

# 10-digit

a = 1234567890

b = 9876543210

print(f"Multiplication of the 10-digits number using Karatsuba : {karatsuba(a, b)}\n\n")

# 50-digit

a = 1234567890123456789012345678901234567890

b = 9876543210987654321098765432109876543210

print(f"Multiplication of the 50-digits number using Karatsuba : {karatsuba(a, b)}\n\n")

# 100-digit

a = 1234567890123456789012345678901234567890123456789012345678901234567890

b = 9876543210987654321098765432109876543210987654321098765432109876543210

print(f"Multiplication of the 100-digits number using Karatsuba : {karatsuba(a, b)}\n\n")

# 1000-digit

a = 1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890

b = 9876543210987654321098765432109876543210987654321098765432109876543210987654321098765432109876543210987654321098765432109876543210

print(f"Multiplication of the 1000-digits number using Karatsuba : {karatsuba(a, b)}\n\n")

# Negative test cases

a = 1234567890

b = 9876543210

print(f"Negative Case\nMultiplication if one num is negative : {karatsuba(-a, b)}")

print(f"Multiplying {a} and -{b} using Karatsuba\nERROR : Integer can't be negative")

#Numeric string test cases

a = "1234567890"

b = "9876543210"

print(f"\n\n#If string is given\nMultiplying {a} and {b} using Karatsuba \nERROR:Enter the valid data type\n\n")

# Invalid string test cases

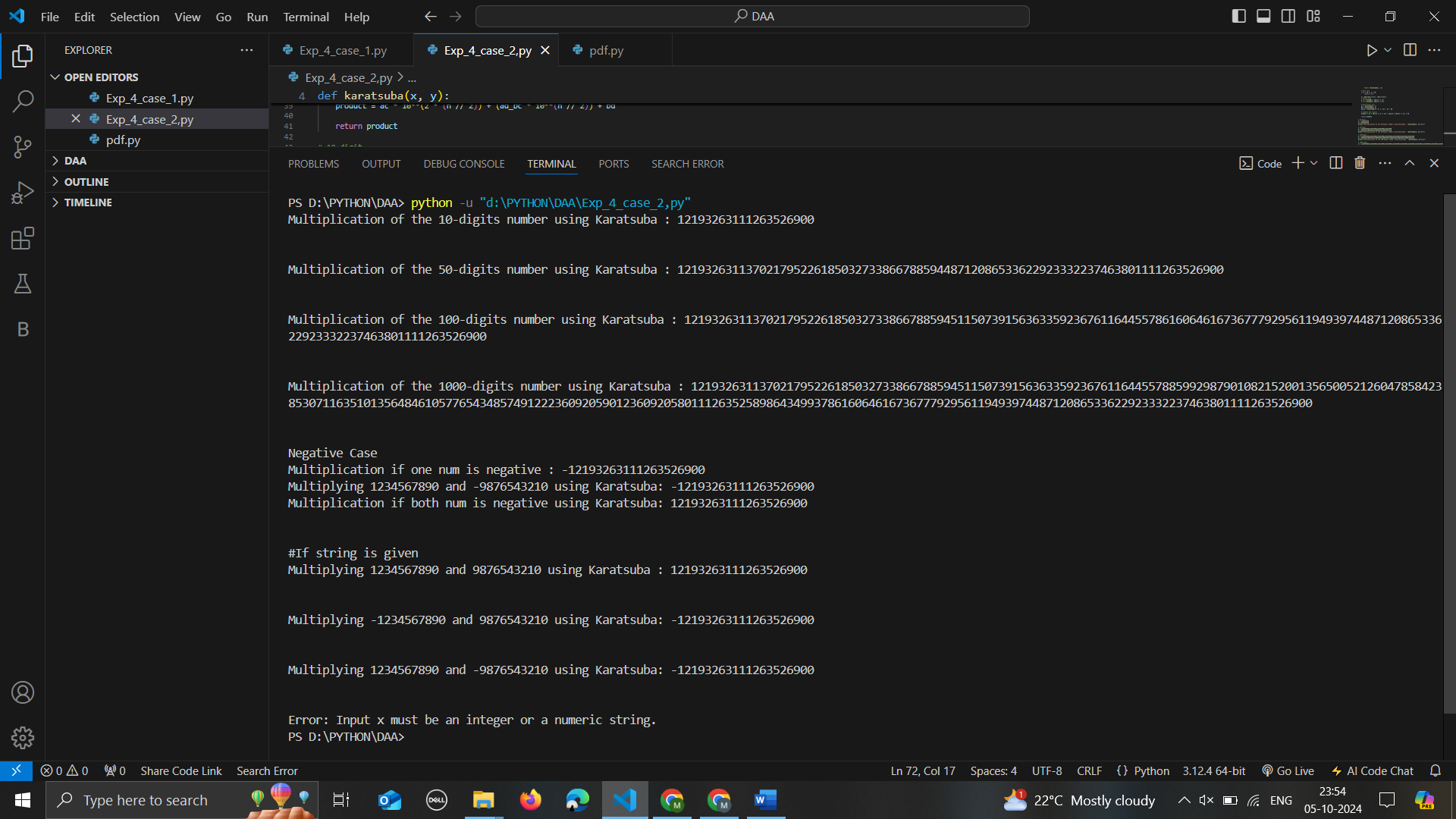
try:

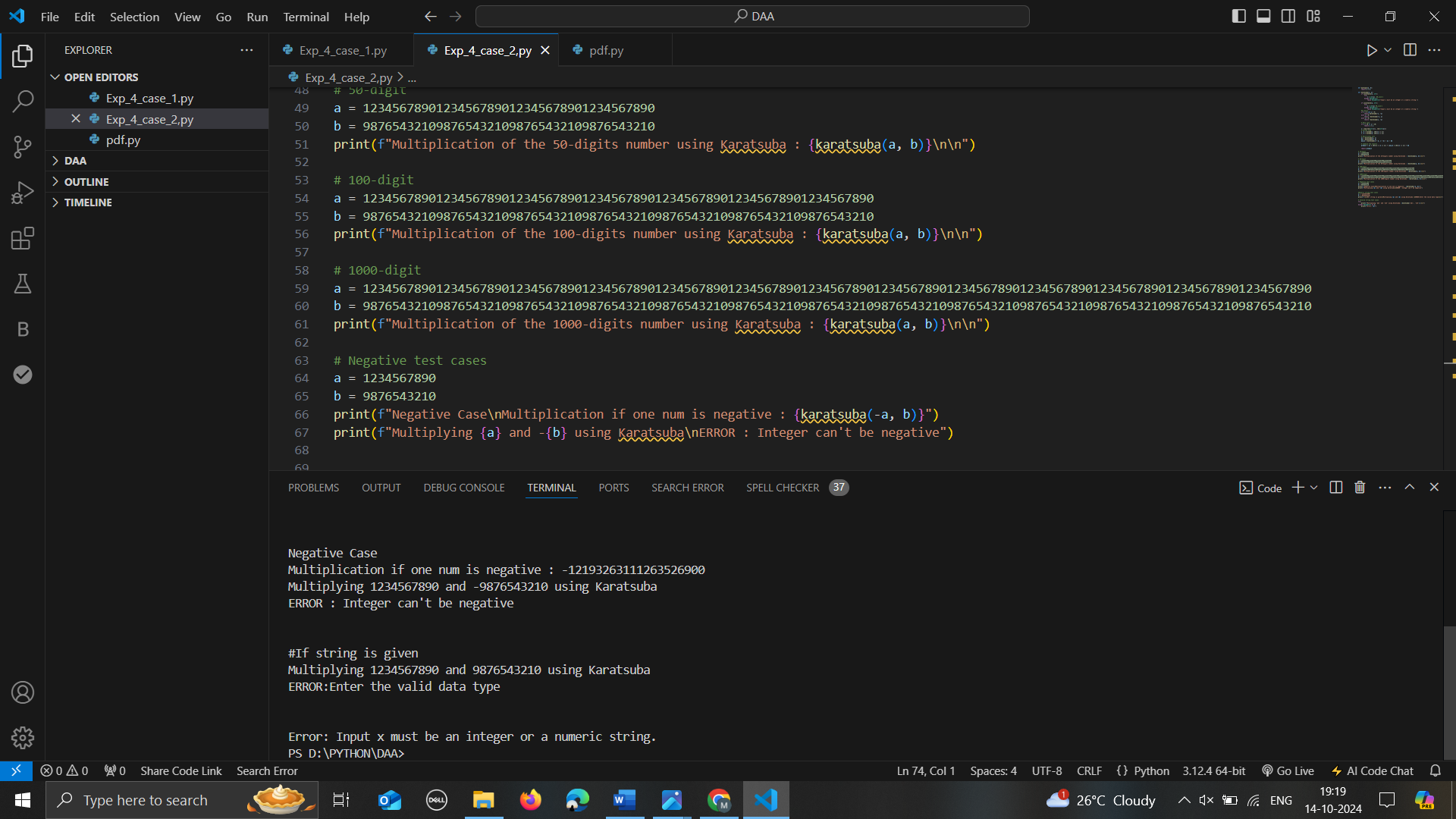
    print(f"Multiplying 'abc' and '123' using Karatsuba: {karatsuba('abc', '123')}\n\n")

except ValueError as e:

    print(f"Error: {e}")

**Output:**





**Conclusion:**

* In this experiment we have learnt about counting inversions. The time complexity using Brute force algorithm is O(n^2) and by divide and conquer algorithm the time complexity is O(nlogn). The time complexity using divide and conquer is less and hence divide and conquer technique is better for this kind of problems.
* The multiplication using Karatsuba multiplication. In this the time complexity by Karatsuba is O(n^1.585) which is lesser than integer multiplication i.e O(n^2). Hence Karatsuba multiplication is preferred.