Data Structures and Algorithms (DSA)

Lab Report 3

Name:	Iqra Fatima
Reg. Number:	23-CP-62
Semester:	3rd
Department:	CPED

Submitted To:

Engineer Sheharyar Khan

Lab Report 3

Marks Obtained: Not evaluated yet

Total Marks: 08

Marks Distribution:

Total Lab Activity Marks: 4

Total Lab Report Marks: 4

Example Tasks

Task : To-Do List Application

```
class ToDoList:
    def __init__(self):
        self.tasks = []

    def add_task(self, task):
        self.tasks.append(task)

    def remove_task(self, task):
        if task in self.tasks:
            self.tasks.remove(task)
        else:
            print(f"Task '{task}' not found.")

    def view_tasks(self):
        print("Tasks:")
        for i, task in enumerate(self.tasks, 1):
            print(f"{i}. {task}")
to_do = ToDoList()
to_do.add_task("Buy groceries")
to_do.add_task("Complete homework")
to_do.view_tasks()
to_do.remove_task("Buy groceries")
to_do.view_tasks()
```

Output:

```
Tasks:
1. Buy groceries
2. Complete homework
Tasks:
1. Complete homework
```

Task 2: Expense Tracker

```
class ExpenseTracker:
    def __init__(self):
        self.expenses = []
    def add_expense(self, amount):
        self.expenses.append(amount)
    def total_expenses(self):
        return sum(self.expenses)
    def max_expense(self):
        return max(self.expenses) if self.expenses else 0
    def min_expense(self):
        return min(self.expenses) if self.expenses else 0
expense_tracker = ExpenseTracker()
expense_tracker.add_expense(20.5)
expense_tracker.add_expense(100.75)
print("Total Expenses:", expense_tracker.total_expenses())
print("Max Expense:", expense_tracker.max_expense())
print("Min Expense:", expense_tracker.min_expense())
```

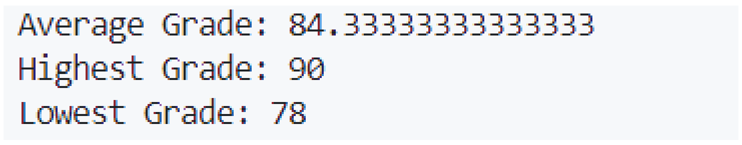
Output:

```
Total Expenses: 121.25
Max Expense: 100.75
Min Expense: 20.5
```

Task 3: Student Grade Tracker

```
class GradeTracker:
    def __init__(self):
        self.grades = []
    def add_grade(self, grade):
        self.grades.append(grade)

    def average_grade(self):
        return sum(self.grades) / len(self.grades) if self.grades
else 0

    def highest_grade(self):
        return max(self.grades) if self.grades else 0

    def lowest_grade(self):
        return min(self.grades) if self.grades else 0

grades = GradeTracker()
grades.add_grade(85)
grades.add_grade(90)
grades.add_grade(78)
print("Average Grade:", grades.average_grade())
print("Highest Grade:", grades.highest_grade())
print("Lowest Grade:", grades.lowest_grade())
```

Output:

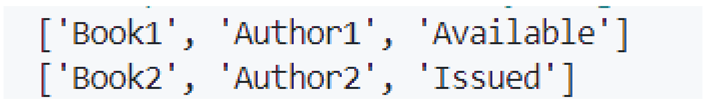


```
Average Grade: 84.33333333333333
Highest Grade: 90
Lowest Grade: 78
```

Task 4: Library Management System

```
class Library:
    def __init__(self):
        self.books = []

    def add_book(self, title, author, status):
        self.books.append([title, author, status])

    def search_book(self, title):
        for book in self.books:
            if book[0] == title:
                return book
        return None

    def display_books(self):
        for book in self.books:
            print(book)

library = Library()
library.add_book("Book1", "Author1", "Available")
library.add_book("Book2", "Author2", "Issued")
library.display_books()
```

Output:



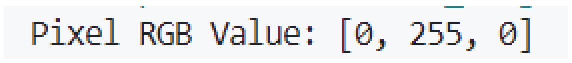
```
['Book1', 'Author1', 'Available']
['Book2', 'Author2', 'Issued']
```

Task 5: RGB Image Processing

```
class RGBImage:
    def __init__(self, rows, cols):
        self.image = [[[0, 0, 0] for _ in range(cols)] for _ in
range(rows)]

    def update_pixel(self, row, col, rgb):
        self.image[row][col] = rgb

    def get_pixel(self, row, col):
        return self.image[row][col]

image = RGBImage(2, 2)
image.update_pixel(0, 0, [255, 0, 0])
image.update_pixel(0, 1, [0, 255, 0])
```

```
print("Pixel RGB Value:", image.get_pixel(0, 1))
```

Output:



```
Pixel RGB Value: [0, 255, 0]
```

Exercise Problems

Easy Problems

1. To-Do List Enhancement

Create a to-do list program that allows users to mark tasks as "completed" and filter only completed tasks to display.

Hint: Use a 1-D array to store tasks and a parallel array to store their completion status (True/False).

Code:

```
class ToDoList:
    def __init__(self):
        self.tasks = []  # List of task names
        self.status = []  # Parallel list for task status (True = Completed, False = Pending)

    def add_task(self, task):
        """Add a new task with default status as False (Pending)."""
        self.tasks.append(task)
        self.status.append(False)

    def mark_completed(self, task):
        """Mark a task as completed if it exists."""
        if task in self.tasks:
            index = self.tasks.index(task)
            self.status[index] = True
            print(f"Task '{task}' marked as completed.")
        else:
            print(f"Task '{task}' not found.")

    def view_completed_tasks(self):
        """Display only completed tasks."""
        print("Completed Tasks:")
        found = False
        for i, (task, done) in enumerate(zip(self.tasks,
self.status), 1):
            if done:
                print(f"{i}. {task}")
                found = True
        if not found:
            print("No completed tasks yet.")

    def view_all_tasks(self):
        """Display all tasks with their status."""
        print("All Tasks:")
        for i, (task, done) in enumerate(zip(self.tasks,
self.status), 1):
            status_text = "✓ Completed" if done else "✗ Pending"
            print(f"{i}. {task} - {status_text}")

# Testing the To-Do List Enhancement
to_do = ToDoList()
to_do.add_task("Buy groceries")
to_do.add_task("Complete homework")
to_do.mark_completed("Buy groceries")
to_do.view_completed_tasks()
to_do.view_all_tasks()
```

Output:

```
Task 'Buy groceries' marked as completed.
Completed Tasks:
1. Buy groceries
All Tasks:
1. Buy groceries - ✓ Completed
2. Complete homework - X Pending
```

2. Daily Expense Calculator

Write a program to store daily expenses in an array and calculate the total expenses for the first seven days.

Hint: Use a for loop to sum up the first seven elements of the array.

Code:

```
class ExpenseCalculator:
    def __init__(self):
        self.expenses = []  # List to store daily expenses

    def add_expense(self, amount):
        """Add a daily expense amount."""
        self.expenses.append(amount)

    def total_first_week(self):
        """Calculate total expenses for the first 7 days."""
        return sum(self.expenses[:7])  # Sum of first 7 elements

    def view_expenses(self):
        """Display all stored expenses."""
        print("Daily Expenses:", self.expenses)

# Testing Daily Expense Calculator
tracker = ExpenseCalculator()
tracker.add_expense(200)
tracker.add_expense(150)
tracker.add_expense(300)
tracker.add_expense(400)
tracker.add_expense(500)
tracker.add_expense(250)
tracker.add_expense(350)
tracker.add_expense(600)  # Extra expense (8th day, should not be included)

tracker.view_expenses()
print("Total expenses for the first 7 days:", tracker.total_first_week())
```

Output:

```
Daily Expenses: [200, 150, 300, 400, 500, 250, 350, 600]
Total expenses for the first 7 days: 2150
```

3. Student Grade Summary

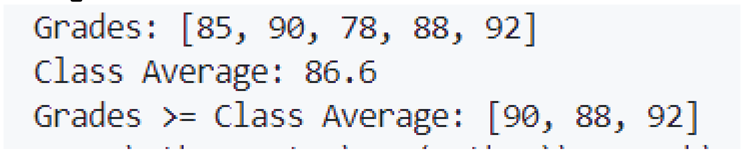
Develop a program to store grades of students for a single subject and display grades greater than or equal to the class average.

Hint: Calculate the average first, then use a loop to filter grades that meet the condition.

Code:

```
class GradeSummary:
    def __init__(self):
        self.grades = []  # List to store grades

    def add_grade(self, grade):
        """Add a new grade to the list."""
        self.grades.append(grade)

    def calculate_average(self):
        """Calculate the average grade."""
        return sum(self.grades) / len(self.grades) if self.grades else 0

    def filter_above_average(self):
        """Return grades that are greater than or equal to the class
average."""
        avg = self.calculate_average()
        return [grade for grade in self.grades if grade >= avg]

    def view_grades(self):
        """Display all stored grades."""
        print("Grades:", self.grades)

# Testing Student Grade Summary
summary = GradeSummary()
summary.add_grade(85)
summary.add_grade(90)
summary.add_grade(78)
summary.add_grade(88)
summary.add_grade(92)

summary.view_grades()
print("Class Average:", summary.calculate_average())
print("Grades >= Class Average:", summary.filter_above_average())
```

Output:



```
Grades: [85, 90, 78, 88, 92]
Class Average: 86.6
Grades >= Class Average: [90, 88, 92]
```

4. Find the Maximum Element

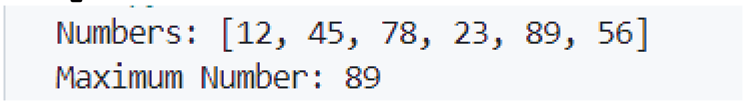
Create a program to find the maximum number in a list of positive integers entered by the user.

Hint: Use a max() function or iterate through the array with a for loop.

Code:

```
# Problem 4: Find the Maximum Element
class MaxFinder:
    def __init__(self):
        self.numbers = []

    def add_number(self, number):
        self.numbers.append(number)

    def find_max(self):
        return max(self.numbers) if self.numbers else None
    def view_numbers(self):
        print("Numbers:", self.numbers)
# Testing MaxFinder
max_finder = MaxFinder()
numbers = [12, 45, 78, 23, 89, 56]
for num in numbers:
    max_finder.add_number(num)

max_finder.view_numbers()
print("Maximum Number:", max_finder.find_max())
```

Output:



```
Numbers: [12, 45, 78, 23, 89, 56]
Maximum Number: 89
```

5. Simple Library Search

Write a program to store book names in a library and allow a user to search for a specific book by its name.

Hint: Use the in keyword to check if the book is in the array.

Code:

```
# Problem 5: Simple Library Search

class Library:

    def __init__(self):

        self.books = []


    def add_book(self, book_name):

        self.books.append(book_name)


    def search_book(self, book_name):

        return book_name in self.books


    def view_books(self):

        print("Books in Library:", self.books)


# Testing Library Search

library = Library()

books = ["Python Basics", "Data Science Handbook", "Machine Learning
Guide"]

for book in books:
    library.add_book(book)


library.view_books()

search_query = "Data Science Handbook"

print(f"Is '{search_query}' available?:",
library.search_book(search_query))
```

Output:

```
Books in Library: ['Python Basics', 'Data Science Handbook', 'Machine Learning Guide']
Is 'Data Science Handbook' available?: True
```

Intermediate Problems

1. Expense Breakdown by Category

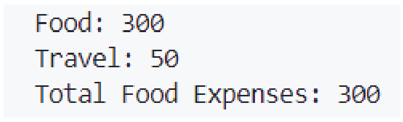
Develop a program to track expenses for different categories (food, travel, utilities, etc.) using a 2-D array. Calculate the total expenses for each category.

Hint: Use a nested list where each row corresponds to a category, and each column is an expense.

Code:

```
# Problem 1: Expense Breakdown by Category

class ExpenseTracker:

    def __init__(self):

        self.categories = {}

    def add_expense(self, category, amount):

        if category not in self.categories:

            self.categories[category] = []

        self.categories[category].append(amount)

    def total_expense(self, category):

        return sum(self.categories.get(category, []))

    def view_expenses(self):

        for category, expenses in self.categories.items():

            print(f"{category}: {sum(expenses)}")

# Testing ExpenseTracker

expense_tracker = ExpenseTracker()

expense_tracker.add_expense("Food", 100)

expense_tracker.add_expense("Travel", 50)
expense_tracker.add_expense("Food", 200)

expense_tracker.view_expenses()

print("Total Food Expenses:", expense_tracker.total_expense("Food"))
```

Output:



```
Food: 300
Travel: 50
Total Food Expenses: 300
```

2. Attendance Tracker

Create a program to track attendance for 5 employees over 5 days using a 2-D array.

Calculate the attendance percentage for each employee.

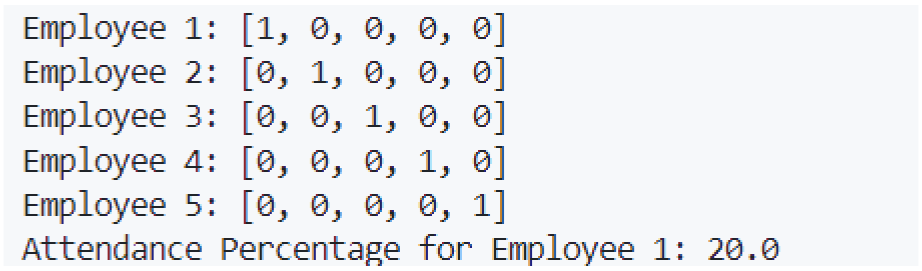
Hint: Use a loop to count the number of 1s in each row and divide by the total days.

Code:

```
# Problem 2: Attendance Tracker

class AttendanceTracker:

    def __init__(self, employees, days):

        self.attendance = [[0] * days for _ in range(employees)]

    def mark_attendance(self, employee, day):

        self.attendance[employee][day] = 1

    def attendance_percentage(self, employee):

        return (sum(self.attendance[employee]) /
len(self.attendance[employee])) * 100

    def view_attendance(self):

        for emp, record in enumerate(self.attendance):

            print(f"Employee {emp + 1}: {record}")

# Testing AttendanceTracker

tracker = AttendanceTracker(5, 5)

tracker.mark_attendance(0, 0)

tracker.mark_attendance(1, 1)

tracker.mark_attendance(2, 2)

tracker.mark_attendance(3, 3)

tracker.mark_attendance(4, 4)

tracker.view_attendance()
```

```
print("Attendance Percentage for Employee 1:",
tracker.attendance_percentage(0))
```

Output:



```
Employee 1: [1, 0, 0, 0, 0]
Employee 2: [0, 1, 0, 0, 0]
Employee 3: [0, 0, 1, 0, 0]
Employee 4: [0, 0, 0, 1, 0]
Employee 5: [0, 0, 0, 0, 1]
Attendance Percentage for Employee 1: 20.0
```

3. Matrix Addition

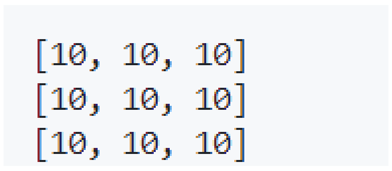
Write a program to add two 2-D matrices (3x3) and display the resulting matrix.

Hint: Use nested loops to add corresponding elements from two matrices.

Code:

```
# Problem 3: Matrix Addition

class Matrix:

    def __init__(self, matrix):

        self.matrix = matrix

    def add(self, other):

        result = [[self.matrix[i][j] + other.matrix[i][j] for j in
range(len(self.matrix[0]))] for i in range(len(self.matrix))]

        return Matrix(result)

    def display(self):

        for row in self.matrix:

            print(row)

# Testing Matrix Addition

matrix1 = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

matrix2 = Matrix([[9, 8, 7], [6, 5, 4], [3, 2, 1]])

result_matrix = matrix1.add(matrix2)

result_matrix.display()
```

Output:



```
[10, 10, 10]
[10, 10, 10]
[10, 10, 10]
```

4. Sort Grades

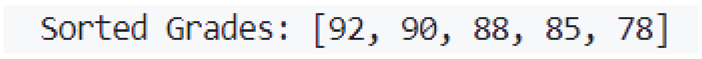
Develop a program to store student grades for five students and sort them in descending order.

Hint: Use a sorting algorithm or Python's sorted() function with the reverse parameter set to True.

Code:

```
# Problem 4: Sort Grades
class GradeSorter:
    def __init__(self, grades):
        self.grades = grades

    def sort_grades(self):
        return sorted(self.grades, reverse=True)

    def display(self):
        print("Sorted Grades:", self.sort_grades())

# Testing GradeSorter
grades = GradeSorter([85, 92, 78, 90, 88])
grades.display()
```

Output:



```
Sorted Grades: [92, 90, 88, 85, 78]
```

5. 2-D Tic-Tac-Toe Enhancements

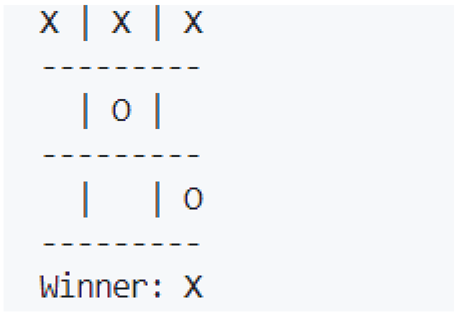
Extend the tic-tac-toe game to announce the winner (player "X" or "O") or declare it as a draw after all moves are completed.

Hint: Check rows, columns, and diagonals for identical values to determine the winner.

Code:

```
class TicTacToe:
    def __init__(self):
        self.board = [[' ' for _ in range(3)] for _ in range(3)]
    def make_move(self, row, col, player):
        if self.board[row][col] == ' ':
            self.board[row][col] = player
        else:
            print("Invalid Move!")
    def check_winner(self):
        for row in self.board:
            if row[0] == row[1] == row[2] != ' ':
                return row[0]
        for col in range(3):
            if self.board[0][col] == self.board[1][col] ==
self.board[2][col] != ' ':
                return self.board[0][col]
        if self.board[0][0] == self.board[1][1] == self.board[2][2]
!= ' ':
            return self.board[0][0]
        if self.board[0][2] == self.board[1][1] == self.board[2][0]
!= ' ':
            return self.board[0][2]
        return None
    def display(self):
        for row in self.board:
            print(" | ".join(row))
            print("-" * 9)
# Testing TicTacToe
game = TicTacToe()
game.make_move(0, 0, 'X')
game.make_move(1, 1, 'O')
game.make_move(0, 1, 'X')
game.make_move(2, 2, 'O')
game.make_move(0, 2, 'X')
game.display()
winner = game.check_winner()
if winner:
    print("Winner:", winner)
else:
    print("No Winner Yet!")
```

Output:



```
 X | X | X
---------
   | O |
---------
   |   | O
---------
Winner: X
```

Advanced Problems

1. Facebook Notifications System

Implement a notifications queue to manage and display sequential updates, allowing

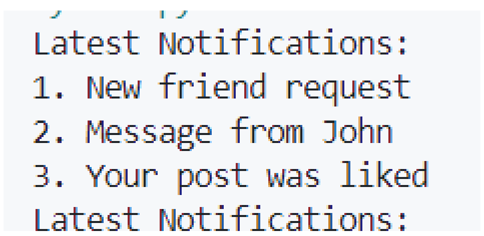
users to "clear all" or view the latest five notifications.

Hint: Use a 1-D array as a queue and maintain a size limit for the array.

Code:

```
class NotificationSystem:
    def __init__(self, limit=5):
        self.notifications = []
        self.limit = limit

    def add_notification(self, notification):
        if len(self.notifications) >= self.limit:
            self.notifications.pop(0)  # Remove the oldest notification
        self.notifications.append(notification)

    def clear_all(self):
        self.notifications = []

    def view_latest_notifications(self):
        print("Latest Notifications:")
        for i, notification in enumerate(self.notifications[-5:], 1):  #
Display only the latest 5 notifications
            print(f"{i}. {notification}")


# Example usage
notif_system = NotificationSystem()
notif_system.add_notification("New friend request")
notif_system.add_notification("Message from John")
notif_system.add_notification("Your post was liked")
notif_system.view_latest_notifications()
notif_system.clear_all()
notif_system.view_latest_notifications()
```

Output:



```
Latest Notifications:
1. New friend request
2. Message from John
3. Your post was liked
Latest Notifications:
```

2. Instagram Image Filter

Write a program to apply a grayscale filter on a 3-D array representing RGB pixel values of an image. Convert each pixel to grayscale using the formula:

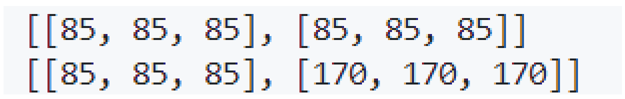
Gray = (R + G + B) / 3.

Hint: Iterate through each pixel (row and column) and apply the formula to update the pixel values.

Code:

```
class ImageProcessor:
    def __init__(self, image):
        self.image = image  # 3D array representing RGB pixels

    def apply_grayscale(self):
        for i in range(len(self.image)):
            for j in range(len(self.image[i])):
                r, g, b = self.image[i][j]
                gray = (r + g + b) // 3
                self.image[i][j] = [gray, gray, gray]  # Apply grayscale

    def display_image(self):
        for row in self.image:
            print(row)


# Example usage
image = [
    [[255, 0, 0], [0, 255, 0]],
    [[0, 0, 255], [255, 255, 0]]
]
processor = ImageProcessor(image)
processor.apply_grayscale()
processor.display_image()
```

Output:



```
[[85, 85, 85], [85, 85, 85]]
[[85, 85, 85], [170, 170, 170]]
```

3. Snapchat Streak Tracker

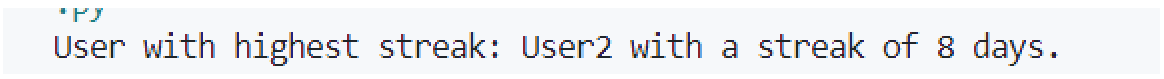
Develop a program to store a 2-D array of streak counts between users over a week. Calculate the highest streak for each user and display the user with the longest streak.

Hint: Iterate through each row to find the maximum streak and use it to identify the user.

Code:

```python
class StreakTracker:
    def __init__(self):
        self.streaks = {}

    def add_streak(self, user, streak_count):
        if user in self.streaks:
            self.streaks[user].append(streak_count)
        else:
            self.streaks[user] = [streak_count]

    def highest_streak(self):
        max_streak_user = None
        max_streak = 0
        for user, streak_list in self.streaks.items():
            highest = max(streak_list)
            if highest > max_streak:
                max_streak = highest
                max_streak_user = user
        return max_streak_user, max_streak


# Example usage
streak_tracker = StreakTracker()
streak_tracker.add_streak("User1", 5)
streak_tracker.add_streak("User1", 7)
streak_tracker.add_streak("User2", 8)
streak_tracker.add_streak("User2", 6)

user, streak = streak_tracker.highest_streak()
print(f"User with highest streak: {user} with a streak of {streak} days.")
```

Output:



```
User with highest streak: User2 with a streak of 8 days.
```

4. Twitter Hashtag Tracker

Create a program to count the occurrences of hashtags in a given list of tweets.

Display the top three most-used hashtags.

Hint: Use a dictionary to store hashtag counts and sort the dictionary by values to find

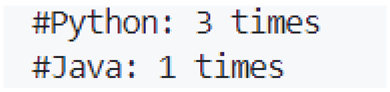
the top three.

Code:

```python
class HashtagTracker:
    def __init__(self):
        self.hashtags = {}

    def add_tweet(self, tweet):
        words = tweet.split()
        for word in words:
            if word.startswith("#"):
                self.hashtags[word] = self.hashtags.get(word, 0) + 1

    def top_hashtags(self, top_n=3):
        sorted_hashtags = sorted(self.hashtags.items(), key=lambda x: x[1], 
reverse=True)
        return sorted_hashtags[:top_n]
```

```
# Example usage
hashtag_tracker = HashtagTracker()
hashtag_tracker.add_tweet("I love #Python programming")
hashtag_tracker.add_tweet("Learning #Python with fun")
hashtag_tracker.add_tweet("Hello #Java and #Python fans")

top_hashtags = hashtag_tracker.top_hashtags()
for tag, count in top_hashtags:
    print(f"{tag}: {count} times")
```

Output:



```
#Python: 3 times
#Java: 1 times
```