Data Structure Assignment

Name: Yash Maan
Roll No.: 2401420019

Course: B. Tech CSE (DS)

Q1: Define the Weather Record ADT

Summary

The Weather Record ADT stores temperature data for multiple years and cities.
It supports the following:
- Attributes: years, cities, data (2D array)
- Methods: insert(year, city, temp), delete(year, city), retrieve(year, city)

Code:

```
class WeatherRecordADT:
    def __init__(self, years, cities):
        self.years = years
        self.cities = cities
        self.data = [[None for _ in range(cities)] for _ in range(years)]

    def insert(self, year, city, temp):
        self.data[year][city] = temp

    def delete(self, year, city):
        self.data[year][city] = None

    def retrieve(self, year, city):
        return self.data[year][city]
```

Q2: 2D Array-Based Storage

Summary

Rows → years, Columns → cities, Each cell data[year][city] holds the temperature.

Code with Example:

```
wr = WeatherRecordADT(3, 4)

wr.insert(0, 0, 25)
wr.insert(0, 1, 30)
wr.insert(1, 2, 28)
wr.insert(2, 3, 22)

print("2D Array Storage:")
for y in range(wr.years):
    print(f"Year {y}: {wr.data[y]}")
```

Output Example:

```
2D Array Storage:
Year 0: [25, 30, None, None]
Year 1: [None, None, 28, None]
Year 2: [None, None, None, 22]
```

Q3: Row-Major vs Column-Major Traversal

Summary

Row-Major Traversal: Traverses year by year.
Column-Major Traversal: Traverses city by city.
Both are O(Y × C), but row-major is faster in Python.

Code:

```
class WeatherRecordWithTraversal(WeatherRecordADT):
    def row_major(self):
        print("Row-major traversal:")
        for y in range(self.years):
            for c in range(self.cities):
                print(f"[{y},{c}] = {self.data[y][c]}", end=" ")
            print()

    def column_major(self):
        print("Column-major traversal:")
        for c in range(self.cities):
            for y in range(self.years):
                print(f"[{y},{c}] = {self.data[y][c]}", end=" ")
            print()
```

Q4: Sparse Data Handling

Summary

When most entries are missing, storing a full 2D array wastes memory. Dictionary stores only non-empty records.

Code:

```
class WeatherRecordSparse(WeatherRecordWithTraversal):
    def __init__(self, years, cities):
        super().__init__(years, cities)
        self.sparse = {}

    def insert(self, year, city, temp):
        super().insert(year, city, temp)
        self.sparse[(year, city)] = temp

    def delete(self, year, city):
        super().delete(year, city)
        if (year, city) in self.sparse:
            del self.sparse[(year, city)]

    def retrieve_sparse(self, year, city):
        return self.sparse.get((year, city), None)

    def display_sparse(self):
        print("Sparse Storage:", self.sparse)
```

Q5: Time & Space Complexity

Operation	Array Storage	Sparse Dict
Insert	O(1)	O(1) avg
Delete	O(1)	O(1) avg
Retrieve	O(1)	O(1) avg
Space Usage	O(Y × C)	O(k), k = non-empty records

Array is better for dense datasets, Sparse Dict saves memory when dataset is sparse.