DS [Day – 4]

UID: 24MCI10204

Name: Rahul Saxena

Branch: 24MCA – AI & ML

Question 1: You are working with a large dataset representing **student scores** in an online exam platform. Each student has a unique **ID** and a score. The data is initially **unsorted**. Your task is to **sort** the student data by their scores, and then perform **search operations** to find specific students based on their **ID**. The dataset can contain up to **1,000,000 students**.
You are required to:

1. Sort the student data using different sorting algorithms.
2. Search for specific students using **linear search** and **binary search**.
3. Compare the performance of the sorting algorithms by measuring the time it takes to sort and search.

Code:

```
import random
import time
class Student:
    def __init__(self, student_id, score):
        self.student_id = student_id
        self.score = score
    def __repr__(self):
        return f"(ID: {self.student_id}, Score: {self.score})"
def generate_students(n):
    students = []
    for i in range(n):
        student_id = f"S{i+1:06d}"  # IDs like S000001
        score = random.randint(0, 100)
        students.append(Student(student_id, score))
    return students
def bubble_sort(students):
    n = len(students)
    for i in range(n):
        for j in range(0, n - i - 1):
            if students[j].score > students[j + 1].score:
                students[j], students[j + 1] = students[j + 1], students[j]
def quick_sort(students):
    if len(students) <= 1:
        return students
    pivot = students[len(students) // 2]
    left = [x for x in students if x.score < pivot.score]
    middle = [x for x in students if x.score == pivot.score]
    right = [x for x in students if x.score > pivot.score]
    return quick_sort(left) + middle + quick_sort(right)
def linear_search(students, target_id):
    for student in students:
        if student.student_id == target_id:
            return student
    return None
def binary_search(students, target_id):
    low = 0
```

```
    high = len(students) - 1
    while low <= high:
        mid = (low + high) // 2
        if students[mid].student_id == target_id:
            return students[mid]
        elif students[mid].student_id < target_id:
            low = mid + 1
        else:
            high = mid - 1
    return None
if __name__ == "__main__":
    NUM_STUDENTS = 10000
    student_data = generate_students(NUM_STUDENTS)
    bubble_students = student_data.copy()
    start = time.time()
    bubble_sort(bubble_students)
    end = time.time()
    print(f"Bubble Sort Time: {end - start:.4f} seconds")
    quick_students = student_data.copy()
    start = time.time()
    quick_students = quick_sort(quick_students)
    end = time.time()
    print(f"Quick Sort Time: {end - start:.4f} seconds")
    target = student_data[random.randint(0, NUM_STUDENTS-1)].student_id
    start = time.time()
    result = linear_search(student_data, target)
    end = time.time()
    print(f"Linear Search Time: {end - start:.6f} seconds → Found: {result}")
    sorted_by_id = sorted(student_data, key=lambda s: s.student_id)
    start = time.time()
    result = binary_search(sorted_by_id, target)
    end = time.time()
    print(f"Binary Search Time: {end - start:.6f} seconds → Found: {result}")
```

Question 2: You need to implement a **Dictionary** (also known as a **Hash Map**) using **hashing**. The dictionary will support the following operations efficiently:

- **Insert**: Insert a new key-value pair into the dictionary.
- **Delete**: Delete a key-value pair based on the key.
- **Lookup**: Retrieve the value associated with a given key.

Code:

```
class HashMap:
    def __init__(self):
        self.size = 1000
        self.table = [[] for _ in range(self.size)]S
    def _hash(self, key):
        return hash(key) % self.size
    def insert(self, key, value):
        index = self._hash(key)
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                self.table[index][i] = (key, value)
                return
        self.table[index].append((key, value))
    def delete(self, key):
        index = self._hash(key)
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                del self.table[index][i]
                return True
        return False
    def lookup(self, key):
        index = self._hash(key)
        for k, v in self.table[index]:
            if k == key:
                return v
        return None
if __name__ == "__main__":
    dictionary = HashMap()
    dictionary.insert("Rahul", 95)
    dictionary.insert("Anjali", 87)
    dictionary.insert("Priya", 90)
    print("Lookup 'Rahul':", dictionary.lookup("Rahul"))
    print("Lookup 'Anjali':", dictionary.lookup("Anjali"))
    dictionary.delete("Anjali")
    print("After deleting 'Anjali':", dictionary.lookup("Anjali"))
```