Problem-Solving Activity 3

GCIS123 – Fall 2024

Activity Title: Creating a Search and Sort Pipeline

Total Marks: 100 (Rubrics given below)

Objective:
This activity will guide students through a structured search and sort process, where each phase builds on the previous one. The assignment aims to strengthen students' understanding of data structures, search algorithms, sorting algorithms, and performance analysis which perfectly aligns with the Course Learning Objective 3. **Note: All sort and search algorithms must have different implementations than algorithms developed during the class lectures.**

Phases and Requirements

Phase 1: Data Generation and Initial Sorting (Insertion Sort)

- **Task**: Write a function generate_sorted_data(size) that generates an array of size random integers between 1 and 100 and sorts the array using insertion sort.
- **Dataset**: For this phase, use the following or any random integers dataset:

```
small_data = [34, 7, 23, 32, 5, 62, 29, 12, 40, 8]
```

- **Expected Output**: After sorting the data with insertion sort, the result should be sorted as shown in the example below:

```
[5, 7, 8, 12, 23, 29, 32, 34, 40, 62]
```

- **Learning Outcome**: Understand and implement insertion sort, and prepare a sorted dataset for efficient searching in the subsequent phases.

Phase 2: Implement Binary Search on Sorted Data

- **Task**: Using the sorted array from Phase 1, implement a binary search function binary_search(sorted_array, target). This function should return the index of the target if found or None if the target is not present.
- **Sample Targets**:
  - Target: 29 (Expected index: 5)
  - Target: 100 (Expected result: None, as it is not in the array)

- **Expected Output**: The function should return the index of the target value or indicate that the target is not in the array.
- ***Learning Outcome***: Apply binary search to locate elements in a sorted array, reinforcing understanding of efficient search algorithms.

Phase 3: Recursive Merge Sort for Large Data

- **Task**: Extend generate_sorted_data(size) to handle larger datasets, such as arrays with 1,000 elements. Sort this larger dataset using merge sort.
- **Dataset**: Use the following subset as part of a larger array

```
large_data = [55, 22, 89, 34, 67, 90, 15, 72, 39, 44] + [random.randint(1, 100) for _ in range(990)]
```

- **Expected Output**: After sorting with merge sort, the first 10 elements of the sorted data should look similar to (sample output):

```
[15, 22, 34, 39, 44, 55, 67, 72, 89, 90]
```

- ***Learning Outcome***: Understand merge sort's efficiency for larger datasets, and compare its effectiveness with simpler sorting algorithms.

Phase 4: Search Performance Comparison

- **Task:** Compare the search times of linear search and binary search on the sorted array from Phase 3. Use the time module to measure the elapsed time for each search method.
    - Implement timing functionality using time.perf_counter().
- **Sample Target:** Use 72 as the target for both search methods.
- **Expected Output:** Display the elapsed time for each search method, illustrating the efficiency of binary search over linear search on large datasets.
- *Learning Outcome*: Analyze and understand the benefits of binary search on large, sorted datasets compared to linear search.

Additional Question for Reflection:

How does the choice of sorting algorithm impact the performance of searching algorithms? Discuss why binary search is faster than linear search on large, sorted data and how efficient sorting methods enhance overall performance.

Rubrics:

Please note that the all the grading mentioned in the Rubrics table is subject to practical demonstration (presentation) of your submitted code solution

Criteria	Points	Description
docStrings	10	Appropriate internal-documentation (i.e. comments & docstring) for each function you implemented in this work
Phase 1: Insertion Sort Implementation	15	Correctly generates and sorts the data using insertion sort, providing the expected output.
Phase 2: Binary Search Implementation	15	Successfully searches for target values using binary search on the sorted dataset from Phase 1.
Phase 3: Merge Sort for Large Data	20	Implements merge sort on a larger dataset, with results matching expected output format.
Phase 4: Search Performance Analysis	20	Accurately measures and compares the time taken by linear and binary searches, with clear results.
Reflection Question Response	10	Provides a thoughtful and accurate discussion on sorting and searching efficiency.
Merging to Github classroom/ Github repositories	10	All students need to upload the code on their github repositories or github classroom

Instructions:

1. **Please note that Academic Integrity Violation is strictly prohibited, any student found guilty will be dealt by the investigation committee for the violation.**
2. Upload **ALL the file (including the provided text files)** to the MyCourses Assignment box as (group_number_activity02.zip) (only one team-member needs to submit to MyCourses).
3. You are not allowed to use any concepts not explained in class this term. Otherwise, -10% will be applied.
4. For late submission, 10% of the work's assigned mark will be deducted, for every calendar day without exception, for a maximum of 2 days, after which it will be graded as zero.