Report

Team Number: 25
Name 1: Shloka Bhatt (ID: 1002150636)
Name 2: Maharshi Shah (ID: 1002158021)

Contributions: Insertion Sort, and Report done by Shloka Bhatt
Random Data generator.py, Merge Sort, Quick Sort done by Maharshi Shah
README.md file done by both

- **References:**
  - timeit module - https://note.nkmk.me/en/python-timeit-measure/#:~:text=source%3A%20timeit_module.py-,timeit.,the%20number%20of%20executions%20increases.
  - File Handling in Python: https://pythonnumericalmethods.berkeley.edu/notebooks/chapter11.01-TXT-Files.html
  - List Comprehensions in Python: https://docs.python.org/3/tutorial/datastructures.html#list-comprehensions

- **Time Complexity of Insertion Sort Algorithm:**
  - T(n) = O(n) for best case and $O(n^2)$ for worst case and average case.

- **Time Complexity of Merge Sort Algorithm:**
  - T(n) = O(n*logn) for all cases.

- **Time Complexity of Quick Sort Algorithm:**
  - T(n) = O(n*logn) for best and average case and $O(n^2)$ for worst case.

- **Experimental Results:**

  1. Upon executing the initial script, insertion_sort.py, the following outcomes were observed:
     - The initial script, dealing with a dataset comprising 20 sets of random data, demonstrated efficient performance by completing the sorting process in a mere 0.000047 seconds.
     - The second script, encompassing a dataset of 100 sets of random data, accomplished the sorting operation within a duration of 0.000712 seconds.
     - The third script, handling a dataset consisting of 2000 sets of random data, completed the sorting process in 0.110340 seconds.
     - The fourth script, dealing with a dataset comprising 6000 sets of random data, required 1.486085 seconds to execute the sorting process.

  2. While executing the merge_sort.py, the following outcomes were observed:
     - The initial script, dealing with a dataset comprising 20 sets of random data, demonstrated efficient performance by completing the sorting process in a mere 0.000039 seconds.

- The second script, encompassing a dataset of 100 sets of random data, accomplished the sorting operation within a duration of 0.000150 seconds.
- The third script, handling a dataset consisting of 2000 sets of random data, completed the sorting process in 0.004120 seconds.
- The fourth script, dealing with a dataset comprising 6000 sets of random data, required 0.020312 seconds to execute the sorting process.

3. While executing the quick_sort.py, the following outcomes were observed:
    - The initial script, dealing with a dataset comprising 20 sets of random data, demonstrated efficient performance by completing the sorting process in a mere 0.000047 seconds.
    - The second script, encompassing a dataset of 100 sets of random data, accomplished the sorting operation within a duration of 0.000139 seconds.
    - The third script, handling a dataset consisting of 2000 sets of random data, completed the sorting process in 0.007658 seconds.
    - The fourth script, dealing with a dataset comprising 6000 sets of random data, required 0.023124 seconds to execute the sorting process.

- **Differences between theoretical and experimental results:**

The observed execution time for Insertion Sort with n=20 was 0.000047 seconds, indicating a faster performance than the worst-case time complexity of O(n^2). Real-world factors, like a small dataset or partially ordered data, can lead to performance variations, deviating from theoretical expectations.

Similarly, Merge Sort's theoretical time complexity is O(n*log n), yet the actual execution time for n=20 was 0.000039 seconds, suggesting a faster performance in this specific case.

For Quick Sort with n=6000, the theoretical time complexity is O(n*log n), but the measured time of 0.023124 seconds reflects real-world variations. Constant factors, hardware specifics, and implementation details contribute to deviations from theoretical expectations.

In essence, theoretical time complexity offers a broad understanding of scaling with input size, but real-world execution times can be influenced by various factors, as seen in the experimental results.

- **Comparison of Insertion Sort, Merge Sort, Quick Sort:**

Insertion Sort:

- Anticipated Traits:
    - Time Complexity:
        - Worst Case: $O(n^2)$
        - Best Case: O(n)
        - Average Case: $O(n^2)$
    - Applicability:
        - Effective for modest datasets or partially ordered data.
        - Diminishes in efficiency as the dataset size expands.

Merge Sort:

- Anticipated Traits:
  - Time Complexity:
    - Worst Case: O(n*log n)
    - Best Case: O(n*log n)
    - Average Case: O(n*log n)
  - Applicability:
    - Effective for substantial datasets.
    - Maintains consistent performance across varied input scenarios.

Quick Sort:

- Anticipated Traits:
  - Time Complexity:
    - Worst Case: $O(n^2)$ (rare, with poor pivot choices)
    - Best Case: O(n*log n)
    - Average Case: O(n*log n)
  - Applicability:
    - Efficient for large datasets.
    - Typically swifter than other \(O(n \log n)\) algorithms in practical scenarios.

Analysis:

1. Small Datasets (20 and 100 rows):
   - Insertion Sort may exhibit competitive performance due to its efficacy with smaller datasets.
   - Merge Sort and Quick Sort might showcase similar performance in optimal and average scenarios.

2. Medium Dataset (2000 rows):
   - Merge Sort and Quick Sort are anticipated to surpass Insertion Sort significantly.
   - The effectiveness of Merge Sort and Quick Sort becomes apparent with more extensive datasets.

3. Large Dataset (6000 rows):
   - Merge Sort and Quick Sort are expected to substantially outpace Insertion Sort.
   - Quick Sort, in particular, is likely to be faster owing to its lower constant factors.

Anomalies:

1. Insertion Sort Anomalies:
   - Best Case Anomaly:
     In cases where the dataset is already partially ordered or exhibits a specific structure, Insertion Sort might deliver surprisingly efficient performance, potentially surpassing theoretical best-case expectations.

2. Quick Sort Anomalies:
   - Worst Case Anomaly:

Although Quick Sort is generally efficient, a worst-case scenario (quadratic time complexity) could arise with unfavorable pivot choices. However, such instances are rare in practice.

Honour of Code:

I pledge, on my honour, ~~code~~ to uphold UT Arlington's tradition of academic integrity, a tradition that values hardwork and honest effort in the pursuit of academic excellence.

I promise that I will submit only work that I personally create or that I contribute to group collaborations, and I will appropriately reference any work from other sources.

I will follow the highest standards of integrity & uphold the spirit of the Honour code.

I will not participate in any form of cheating / sharing the questions / solutions.

09/30/2023 09/30/2023