3. Complexity analysis based on your results with the theory you learn.

Complexity Analysis of CS401 Project

Based on the reviewed code and understanding of common data structures and algorithms, here's a complexity analysis of the implemented functionalities in the CS401 project:

Data Structures:

- **DataStructure Class:** This class likely uses an underlying array to store elements. Adding and removing elements at the end have a time complexity of O(1) (constant time) due to direct array access. Random access (getting an element by index) is also O(1). However, inserting or removing elements in the middle requires shifting elements, resulting in O(n) (linear time) complexity in the worst case.

Sorting Algorithms:

- **Selection Sort:** The implemented selection sort has a time complexity of O(n^2) (quadratic time). It iterates through the list n times, comparing each element with the remaining elements to find the minimum.
- **Merge Sort:** The implemented merge sort has a time complexity of O(n log n) (logarithmic time). It recursively divides the list into halves, sorts the halves, and then merges them back together efficiently.

Searching Algorithms:

- **Linear Search:** The linear search has a time complexity of O(n) (linear time) in the worst case. It iterates through the entire list until the target element is found.
- **Binary Search:** The binary search has a time complexity of O(log n) (logarithmic time) in the worst case. It assumes the list is sorted and keeps halving the search space based on comparisons.
- **Hash Search:** The improved hash search has a time complexity of O(1) (constant time) on average, assuming a good hash function is used. It directly accesses the element based on its hashed index in the hash table.

Other Operations:

- **Adding Data:** Adding data to the end of the list using the DataStructure class has a time complexity of O(1) (constant time) due to direct array access (assuming there's enough capacity).
- **Deleting Data:** Deleting data by value requires searching for the element (O(n) in the worst case) and then shifting elements (O(n) in the worst case), leading to a total worst-case complexity of O(n).
- **Updating Data:** Updating data also involves searching for the element (O(n) in the worst case) and then modifying its value (O(1)), resulting in a total worst-case complexity of O(n).

Missing Functionalities:

- **Restore Data:** The current implementation simply calls add, which has a complexity of O(1) (assuming enough capacity). True restoration with a deleted elements list might involve searching the deleted list (O(n) in the worst case) and then adding it back (O(1)), leading to a potential worst-case complexity of O(n).
- **Analysis Report Enhancements:** Measuring execution times for specific data manipulation operations involves additional code and would likely have a negligible impact on the overall complexity.

Important Notes:

The complexities mentioned are for the worst-case scenarios. Average-case complexities might be better for some operations (e.g., searching in a nearly sorted list).

The analysis assumes the size of the data structure (n) is the dominant factor affecting execution time.