**TASK 1: Introduction to Search Algorithms**

**Objective:** Understand linear and binary search algorithms and implement them.

**Assignment:**

1. **Theory:** Write a brief explanation of the difference between linear search and binary search in terms of algorithmic approach and efficiency.
2. **Implementation:** Write a Python program to implement linear search and another program to implement binary search.
3. **Analysis:** For a given list of n elements, compare the worst-case performance of linear search and binary search. Explain why binary search requires the list to be sorted beforehand.
4. **Extra Challenge:** Implement a recursive version of binary search and compare its performance with the iterative version in terms of readability and efficiency.

**TASK 2: Dive into Sorting Algorithms - Part 1**

**Objective:** Explore and implement bubble sort and insertion sort.

**Assignment:**

1. **Theory:** Explain how bubble sort and insertion sort algorithms work, including their sorting mechanism.
2. **Implementation:** Write Python programs to implement bubble sort and insertion sort.
3. **Visualization:** Create a simple animation (can use online tools or Python libraries) to visualize the sorting process of both algorithms on a small list of integers.
4. **Analysis:** Discuss the time complexity of bubble sort and insertion sort in the best, average, and worst cases. Explain the scenarios where one might prefer insertion sort over bubble sort.

**TASK 3: Dive into Sorting Algorithms - Part 2**

**Objective:** Understand merge sort and *quick sort (not covered in lecture – do research on your own!)* through implementation and analysis.

**Assignment:**

1. **Theory:** Describe the divide-and-conquer strategy used in merge sort and the partitioning logic in quick sort.
2. **Implementation:** Write Python programs to implement merge sort and quick sort.
3. **Comparison:** Compare merge sort and quick sort in terms of their time complexity in the best, average, and worst cases. Discuss the space complexity of merge sort.
4. **Extra Challenge:** Modify the quick sort implementation to choose the pivot in three different ways: always the first element, always the last element, and the median of three. Discuss how the pivot selection strategy affects the algorithm's performance.

**TASK 4: Advanced Sorting Algorithms and Complexities**

**Objective:** Study shell sort and heap sort (*(not covered in lecture – do research on your own!)*, focusing on their algorithmic design and complexities.

**Assignment:**

1. **Theory:** Explain the concepts behind shell sort's gap sequence and heap sort's binary heap data structure.
2. **Implementation:** Implement shell sort and heap sort in Python.
3. **Analysis:** Analyze the time complexity of shell sort and heap sort, noting the impact of the gap sequence on shell sort's performance.
4. **Extra Challenge:** For heap sort, demonstrate how the algorithm can be used to sort in-place without needing additional memory for another heap data structure.

**TASK 5: Search and Sort Algorithm Applications**

**Objective:** Apply knowledge of search and sorting algorithms to solve real-world problems.

**Assignment:**

1. **Problem Solving:** Choose a real-world dataset (e.g., student grades, stock prices, etc.). Apply appropriate sorting algorithms to organize the data, and then use search algorithms to answer specific questions about the data (e.g., finding the median, searching for specific entries).
2. **Efficiency Analysis:** Justify the choice of algorithms for the dataset based on the data's characteristics and the complexity of the algorithms.
3. **Discussion:** Write a short essay on the importance of algorithm selection in software development, considering factors like data size, time complexity, and the nature of the data.
4. **Extra Challenge:** Implement a hybrid sorting algorithm that combines two sorting techniques and demonstrates its effectiveness on your chosen dataset compared to a single algorithm approach.