Complexity Analysis:

Sorting Algorithms:

Insertion Sort, Bubble Sort, Selection Sort: Worst-case time complexity: O(n^2) for each.

Quick Sort: Worst-case time complexity: O(n^2), but average-case is O(n log n).

Merge Sort: Worst-case time complexity: O(n log n).

Heap Sort: Worst-case time complexity: O(n log n).

Bubble Sort:

Average-case time complexity: O(n^2)

Best-case time complexity: O(n) (when the list is already sorted)

Selection Sort:

Average-case time complexity: O(n^2)

Best-case time complexity: O(n^2)

Insertion Sort:

Average-case time complexity: O(n^2)

Best-case time complexity: O(n) (when the list is almost sorted)

Merge Sort:

Average-case time complexity: O(n log n)

Best-case time complexity: O(n log n)

Quick Sort:

Average-case time complexity: O(n log n)

Best-case time complexity: O(n log n)

Heap Sort:

Average-case time complexity: O(n log n)

Best-case time complexity: O(n log n)

Searching Algorithms:

Linear Search: Worst-case time complexity: O(n).

Binary Search: Worst-case time complexity: O(log n)

Linear Search:

Average-case time complexity: O(n)

Best-case time complexity: O(1) (when the target element is the first element)

Binary Search:

Average-case time complexity: O(log n)

Best-case time complexity: O(1) (when the target is in the middle of the sorted list)

Overall Analysis:

The code allows users to choose different sorting and searching algorithms based on the input type (integer, double, or string). The Big O complexity for the entire code is dominated by the algorithm with the highest complexity. The worst-case complexities, the sorting and searching algorithms contribute O(n^2) and O(n log n) complexities.