CSE222 / BİL505
Data Structures and Algorithms
Homework #6 – Report

Cemal BOLAT

1) Selection Sort

Time Analysis	Selection Sort demonstrates consistent behavior across various types of inputs. It operates with a time complexity of O(n^2), where 'n' represents the number of elements in the array. Performance on Different Inputs: No matter how input is Selection Sort performs individual swaps and comparisons for each element. This means Selection sort has no optimization level.
Space Analysis	Selection Sort is an in-place sorting algorithm, so it does not require additional space and has O(1) space complexity.

2) Bubble Sort

Time Analysis	Bubble Sort demonstrates a time complexity of O(n^2) in both the worst and average cases, where 'n' represents the number of elements in the array. However, in the best case scenario where the array is already sorted, the time complexity reduces to O(n). Performance on Different Inputs: Sorted Array: The best case and the time complexity is O(n). Reversely Sorted Array: The worst and the time complexity is O(n^2). Randomized Array: The average case and the time complexity is O(n^2).
Space Analysis	Similar to Selection Sort, Bubble Sort operates in-place, meaning it does not require additional space and has O(1) space complexity.

3) Quick Sort

Time Analysis	Quick Sort may have different time complexity according to selection of pivot. If we peek poor pivot, time complexity can degrade to O(n^2) but average complexity is O(n logn). Examples of poor pivot selection include choosing the first or last element (as in my implementation) when the array is sorted, nearly sorted, or contains many duplicate elements.
Space Analysis	Space complexity in Quick Sort is related with Time Complexity on the scenario for given input. Average and Best case space complexity is O(logn). Worst case space complexity is O(n).

4) Merge Sort

Time Analysis	Among the four sorting algorithms discussed, Merge Sort stands out as the most efficient due to its consistent time complexity of O(n log n) for all cases.
Space Analysis	Merge Sort requires additional space proportional to the input size for storing temporary arrays during the merge process. Thus, its space complexity is O(n) each case on my implementation.

General Comparison of the Algorithms

Four of these algorithm has different advantages and disadvantages.

Selection Sort and Bubble Sort are simple and easy to implement but are less efficient compared to Quick Sort and Merge Sort, particularly for larger datasets.

Quick Sort and Merge Sort offer superior performance with a time complexity of O(n log n), making them well-suited for sorting large datasets efficiently.

While Quick Sort may suffer from worst-case scenarios with improper pivot selection, Merge Sort remains consistently efficient across all input scenarios, albeit with slightly higher space complexity.

Bubble Sort has best ‘best case senerio’ on nearly sorted datasets.

To clearify it here is the table of swap and comparison counter for nearly sorted and randomly sorted 100 elements.

Sort	Condition	Comparison	Swap
Selection Sort	Randomly sorted 100	4950	99
Selection Sort	Nearly sorted 100	4950	99
Bubble Sort	Randomly sorted 100	4719	2457
Bubble Sort	Nearly sorted 100	99	0
Quick Sort	Randomly sorted 100	775	450
Quick Sort	Nearly sorted 100	4590	5048
Merge Sort	Randomly sorted 100	545	0
Merge Sort	Nearly sorted 100	356	0