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

ABDULLAH TURKMEN

1) Selection Sort

Time Analysis	The time complexity of selection sort is O(n^2), making it one of the less efficient sorting algorithms. This is because selection sort involves iterating through the array multiple times, with each iteration involving a linear search to find the minimum or maximum element. As the size of the array (n) increases, the number of comparisons and swaps grows quadratically, resulting in a significant increase in execution time. When we change element of arr, comparison count is not change (Comparison Counter: 27, Swap Counter: 7)
Space Analysis	Selection sort has a space complexity of O(1), as it only requires a small amount of extra memory to store temporary swaps.

2) Bubble Sort

Time Analysis	The time complexity of bubble sort is O(n^2) in the worst and average cases, making it one of the less efficient sorting algorithms. This is because bubble sort involves repeatedly iterating through the array, comparing adjacent elements and swapping them if they are in the wrong order. As the size of the array (n) increases, the number of comparisons and swaps grows quadratically, resulting in a significant increase in execution time. However, in the best case scenario, when the array is already sorted, bubble sort has a time complexity of O(n), as it only requires a single pass through the array to confirm that it is sorted. (Comparison Counter: 7, Swap Counter: 0)
Space Analysis	Bubble sort has a space complexity of O(1), as it only requires a small amount of extra memory to store temporary swaps.

3) Quick Sort

Time Analysis	The time complexity of quicksort is O(n log n) on average, making it one of the most efficient sorting algorithms. However, in the worst case, when the pivot is chosen poorly, the time complexity can degrade to O(n^2). This occurs when the partitioning step divides the array into very unbalanced subarrays, leading to a recursive call with a large subarray.

	Despite this, quicksort's average-case performance is excellent, and it is often used in practice due to its high speed and low overhead. İf pivot is top right ->(Comparison Counter: 14 Swap Counter: 9) if pivot is top – left -> Comparison Counter: 15 Swap Counter: 8
Space Analysis	Quicksort has a space complexity of O(log n) due to the recursive call stack, although this can be reduced to O(1) with an in-place implementation.

4) Merge Sort

Time Analysis	The time complexity of merge sort is O(n log n) in all cases (best, average, and worst), making it one of the most efficient and reliable sorting algorithms. This is because merge sort divides the array into two halves recursively, sorts each half, and then merges the two sorted halves into a single sorted array. The logarithmic factor comes from the recursive division of the array, while the linear factor comes from the merging step. (Comparison Counter: 15 Swap Counter: 0) it has no swap.
Space Analysis	Merge sort has a space complexity of O(n), as it requires a temporary array to store the merged result.

General Comparison of the Algorithms

In general, the sorting algorithms discussed - selection sort, bubble sort, quicksort, and merge sort - exhibit distinct trade-offs between efficiency, simplicity, and stability. Selection sort and bubble sort, with their O(n^2) time complexities, are simple to implement but inefficient for large datasets. Quicksort, with its average-case O(n log n) time complexity, is a popular choice for its high speed and low overhead, but its worst-case performance can be poor. Merge sort, also with an O(n log n) time complexity, offers excellent stability and efficiency, but requires extra memory for the merging step. Ultimately, the choice of algorithm depends on the specific requirements of the application, including the size and nature of the data, available memory, and performance constraints. While each algorithm has its strengths and weaknesses, understanding their complexities and trade-offs is essential for selecting the most suitable algorithm for a given problem.