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

DENİZ ALTINIŞIK

210104004064

1) Selection Sort

Time Analysis	O(n^2) in all cases, regardless of the input. This is because it iterates over the array n times and, in each iteration, performs a comparison for each remaining element. Hence, its time complexity remains the same for sorted, reversely sorted, and randomized arrays.
Space Analysis	Selection Sort has an auxiliary space complexity of O(1) as it only requires a constant amount of extra space for variables like loop indices and temporary storage during swapping. Thus, its space complexity remains constant regardless of the input size.

2) Bubble Sort

Time Analysis	O(n^2) in the worst case for all input types. It iterates over the array n times, comparing adjacent elements and swapping them if they are in the wrong order. This process continues until the array is sorted. Hence, its time complexity remains quadratic irrespective of the input.
Space Analysis	Bubble Sort's space complexity is O(1), requiring only a constant amount of additional space for variables. It does not depend on the input size.

3) Quick Sort

Time Analysis	O(n log n) as average case complexity, but worst-case time complexity is O(n^2) which occurs when the pivot selection consistently leads to highly unbalanced partitions, such as when the smallest or largest element is choosen as pivot. Quick Sort's performance can vary significantly based on the input. For sorted or reversely sorted arrays, if the pivot selection is not optimized, Quick Sort can degrade to its worst-case time complexity. On the other hand, for randomized arrays, Quick Sort typically performs closer to its average-case time complexit.
Space Analysis	O(log n) for the recursive function calls on the stack during the sorting process. However, in the worst case, it can degrade to O(n) due to the unbalanced partitioning. Despite this, Quick Sort's space usage is generally efficient compared to algorithms like Merge Sort, which require additional space proportional to the input size.

4) Merge Sort

Time Analysis	O(n log n) for all cases, including sorted, reversely sorted, and randomized arrays. It divides the input array into halves recursively until individual elements are reached and then merges them in sorted order. This divide-and-conquer approach ensures a logarithmic time complexity.
Space Analysis	Merge Sort has an additional space complexity of O(n) due to the need for auxiliary arrays during the merging process. Each recursive call creates temporary arrays for merging, leading to a linear increase in space usage with the input size.

General Comparison of the Algorithms

The choice of sorting algorithm depends on various factors, including the size and nature of the input data, as well as considerations regarding time and space efficiency.

For general-purpose sorting with relatively small datasets, Quick Sort or Merge Sort may be preferred due to their average-case time complexity of O(n log n). However, for scenarios where space efficiency is critical, algorithms like Selection Sort or Bubble Sort, with their minimal space requirements, might be more suitable.