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

EMRE YAVUZ

1) Selection Sort

Time Analysis	Time Complexity Analysis The Selection Sort illustrates a quadratic time complexity (O(n^2)) since it has to look through the unsorted part of the sequence many times and identify the smallest element regardless of its original position within the array. The fact that it has to keep finding this minimum by going through what has already been sorted makes it inefficient when it comes to sorting a large set.
Space Analysis	The in-place sorting algorithm has a space complexity of O(1). This signifies that it does not need any other storage apart from the memory already assigned for the original array, thus reflecting its optimal use of memory. It directly manipulates the array elements without needing extra space for auxiliary data structures, hence showcasing its efficiency in managing memory resources.

2) Bubble Sort

Time Analysis	The time analysis for Bubble Sort reveals that it has a worst-case and average time complexity of O(n^2), which renders it impractical for large lists. Nevertheless, in the event that the array is already sorted, Bubble Sort can attain a best-case time complexity of O(n) by adopting the approach of flagging to terminate the process early if no swaps are required throughout the traversal. This particular optimization empowers Bubble Sort with the capacity to deal effectively with either sorted or nearly sorted arrays.
Space Analysis	Bubble Sort operates by directly engaging the input array to be sorted without the need for any extra storage space, other than a constant amount of memory. Its space complexity is O(1), which underscores its effectiveness in minimizing memory consumption: it achieves this by working with array elements straightforwardly, without having to rely on supplementary data structures.

3) Quick Sort

Time Analysis	Time Analysis Quicksort is known for its efficiency, with average and best-case time complexity of O(n log n). However, in the worst case (O(n^2)), where a partition is persistently unbalanced due to poor pivot selection, such as when sorting a sorted array, performance can degrade significantly.

Space Analysis	Although quick sort is not complete (due to recursive stack space), its space complexity can be optimized to O(log n) through tail recursion.

4) Merge Sort

Time Analysis	Time analysis merge sort ensures a consistent time complexity of O(n log n) in all scenarios, showing strong performance and reliability even on nearly sorted or reverse sorted data.
Space Analysis	The main disadvantage is space usage, which requires additional memory proportional to the array size (O(n)) and is less space efficient than the in-place sorting algorithm.

General Comparison of the Algorithms

General comparison of algorithms

In a broad view of the algorithms, Quick Sort and Merge Sort are usually preferred over Selection or Bubble Sort due to their use of a logarithmic factor that decreases the number of comparisons as the dataset increases.

Despite its inefficiency, in cases with small or nearly sorted data, Bubble Sort can be quite effective if optimized. On the other hand, Selection Sort is generally not a good choice unless specific memory limitations are in place.

Quick Sort is often the choice for bigger data sets, mainly because of its great performance in average case which makes it very useful especially in real life situations. However, Merge Sort is more effective where data stability is important— like when you need to keep equal elements' order or if you want a steady sorting time regardless of what the input conditions look like. This makes Merge Sort a wise selection for applications that value predictability and reliability since these two are not compromised under any circumstances.