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

ALPER TAVŞANOĞLU

1) Selection Sort

Time Analysis	In Selection Sort, we select the smallest element of the array by comparing each number and when we find a smaller element, we change the smallest element with what we found. For n element array, outer loop runs n-1 times. For each iteration of the outer loop, the inner loop runs for n times. So, time complexity of the selection sort will be O(n^2).
Space Analysis	Since we only swap some numbers in the array in place and we do not use any additional data structures the space complexity for this algorithm is O(1).

2) Bubble Sort

Time Analysis	In Bubble Sort, we compare two elements and swaps them until they are in the right order. Normally Bubble Sort's time complexity O(n^2) with nested itreation. But with my Optimized Bubble Sort Algorithm implementation i added swap_check variable to see array is already sorted or not. If the array already sorted, we will not swap elements, so we just need only one itreation for array with n element. So time complexity will be O(n), if array already sorted (best case). On other hand, algorithm will stop when sorted. For worst case situation it will be O(n^2).
Space Analysis	Since we only swap some numbers in the array in place and we do not use any additional data structures the space complexity for this algorithm is O(1).

3) Quick Sort

Time Analysis	In Quick Sort, we are using divide-and-conquer algorithm. It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The sub-arrays are then sorted recursively. In the most balanced case, each time we perform a partition we divide the list into two nearly equal pieces. This means each recursive call processes a list of half the size. The result is that the algorithm uses only O(n log n) time for best case. This may occur if the pivot chosen at the each step divides the array into roughly equal halves. On other hand, worst case for quick sort algorithm time compexity will be O(n^2). This may occur if the pivot happens to be the smallest or largest element in the list. So the time complexity of QuickSort depends on the pivot selection.

Space Analysis	The space complexity of my QuickSort code varies based on how balanced the partitioning is during the recursive steps, with a more balanced partition resulting in a more efficient use of space. The maximum depth of the recursive call stack will depend on the height of the recursion tree. The worst case, the recursion depth becomes O(n) when the smallest or largest element is always chosen as the pivot. The best case, the recursion tree will be balanced, leading to a recursion depth of O(log n).

4) Merge Sort

Time Analysis	In Merge Sort, Best Case Time Complexity is O(n log n), Worst Case Time Complexity is O(n log n), Average Time Complexity is O(n log n). The time complexity of MergeSort is O(n log n) in all the 3 cases (worst, average and best) as the MergeSort always divides the array into two halves and takes linear time to merge two halves.
Space Analysis	Merge sort has a space complexity of O(n).This is because, unlike the other sorting algorithms, Merge Sort requires additional space proportional to the array size for the temporary arrays used during the merge process (int[] left_Arr and int[] right_Arr in my code).

General Comparison of the Algorithms

The choice of sorting algorithm depends on the specific requirements and constraints of the environment in which it is used, including considerations of data size, order, speed requirements, and space availability.

Best use case for BubbleSort when the data is nearly sorted as it minimizes unnecessary operations. Worst use case, large datasets and datasets in random or reverse order.

Best use case for SelectionSort when we need minimal number of swaps. Worst use case, large datasets like BubbleSort.

Best use case for QuickSort when large datasets where average performance is important. Worst use case, when the pivot is consistently the smallest or largest element.

Best use case for QuickSort Sorting linked lists and when stable sorting is required. Worst use case, when if memory is a constraint.

Other examples;

If Space is a constraint, best choice is BubbleSort or SelectionSort, both of these sorts have O(1) space complexity. Other hand, if space is not a concern, Merge and Quick sorts.

For nearly sorted or small datasets, best choice is BubbleSort, especially if the dataset is small or nearly sorted, as it minimizes unnecessary operations.