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

Akif Safa Angi

1) Selection Sort

Time Analysis	Outer loop runs n -1 times(n is array length). Inner loop runs i+1 to n. So it runs (n-1) at first iteration and goes to 1 times. It's comparison counter and swap is always same in every case. So it may look slower than some sorts according to case. **It will run n*(n-1)/2 at total. Time complexity. O(n^2)** **Best Case = Average Case = Worst Case = O(n^2)**
Space Analysis	It uses int n to hold length of array. And uses a a fixed amount of extra space (a few variables like i, j, min). No additional data structures(like an array) are used. **It's space complexity is O(1)**

2) Bubble Sort

Time Analysis	**Best Case(when the array is already sorted):** The time complexity is O(n) because the outer loop will run only once, and no swaps will occur after the first pass. **Average Case**: For a random array, the average number of total swaps is (n^2)/4, thus the average case complexity is **O(n^2).** **Worst case:** outer loop runs n-1 times. Inner loop runs i+1 to n. So it runs (n-1) at first iteration and goes to 1 times. **It will run n*(n-1)/2 at total.** Comparison and swap counter may change depends on case. Especially in ordered array case it use less comparison than other sorts. **Best-case Time Complexity: O(n) (when the array is already sorted)** **Average and Worst-case Time Complexity: O(n^2)**
Space Analysis	It uses int n to hold length of array. And it uses bool swapped to check swapping. Also uses some fixed amount int like i and j in for loop. **Space Complexity: O(1)**

3) Quick Sort

Time Analysis	Quick Sort's time complexity ranges from **O(n log n)** in its best and average cases to **O(n^2)** in its worst case, depending on pivot selection. For comparison and swap counters it's not good at ordered array. **Best and Average Case Time Complexity: O(n logn)** **Worst Case Time Complexity: O(n^2)**
Space Analysis	Its space complexity is O(log n) due to its recursive partitioning, which requires stack space for function calls. **Space Complexity: O(logn)**

4) Merge Sort

Time Analysis	Merge Sort exhibits a consistent time complexity of O(n log n) regardless of the input data or its initial order. It achieves this efficiency by dividing the array into halves recursively until it reaches atomic units, then merging the sorted halves. But if we look to the comparison it usually use less than the others. Also doesn't do any swap. **Time Complexity: O(n log n)**
Space Analysis	Merge Sort requires additional space for the temporary arrays for merging process. The space complexity is **O(n)** because, in the worst-case scenario, it might need to temporarily store the entire array during merging. **Space Complexity: O(n)**

General Comparison of the Algorithms

In ordered array case according to time complexity best algorithm is bubble sort algorithm and worst algorithm is quick sort if we check comparison and swap counters. But in space comparison selection sort is better than the others. In space comparison worst case is merge sort.

In random case according to time complexity best algorithm is selection sort and bubble sort algorithm and worst algorithm is merge sort. If we check comparison and swap counters merge sort use less than the others. But in space comparison merge sort use more space than the others. In space comparison best case is selection sort.

In reverse ordered case according to time complexity best algorithm is bubble, selection and quick sort algorithm and worst algorithm is merge sort. If we check comparison and swap counters merge sort use less than the others. But in space comparison merge sort use more space than the others. In space comparison best case is selection sort.