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

BUKET GENÇER

1) Selection Sort

Time Analysis	Worst = O(n^2) The worst case occurs when the array is completely reverse ordered Each element requires searching through the remaining array to find the minimum or maximum, leading to approximately n*(n-1)/2 comparisons, thus O(n^2) complexity. Avarage = O(n^2) In the average case, the arrangement of the elements in the array is random Best = O(n^2) The best case is when the array is already sorted. , Even in the best case (sorted array), the algorithm still checks each element against all others not yet positioned correctly.
Space Analysis	Space Complexity = O(1) Only a few local variables are used, which requires a fixed amount of extra space so O(1)

2) Bubble Sort

Time Analysis	Worst = O(n^2) The worst case occurs when the array is completely reverse ordered. In this scenario, every element needs to be compared and swapped with every other element, requiring n*(n-1)/2 operations. Avarage = O(n^2) In the average case, the arrangement of the elements in the array is random Best = O(n) The best case is when the array is already sorted. In the best case, Bubble Sort makes a single pass through the array to check for any swaps needed. Detecting no swaps, it concludes efficiently in linear time.
Space Analysis	Space Complexity = O(1) Only a few local variables are used, which requires a fixed amount of extra space so O(1)

3) Quick Sort

Time Analysis	Worst = O(n^2) Occurs when the pivot selection is consistently the smallest or largest element of the sublist Average = O(n log n) The array is divided roughly in half with each recursive step, leading to log(n) levels of recursion. Best = O(n log n) Similar to the average case, but with the most optimal pivot selection at each step, perfectly dividing the array into equal parts.
Space Analysis	Space Complexity = O(log n) Quick Sort's space complexity stems from the stack space used for recursive calls during partitioning. Under ideal conditions, where partitions are evenly balanced, the recursive call stack grows to a maximum depth of log(n).

4) Merge Sort

Time Analysis	Worst = O(n log n) Average = O(n log n) Best = O(n log n) In the Merge Sort operation, we do a logarithmic number of divisions the merge occurs linearly (n). Merge sort always splits the array into two, and each merge operation merges two independently sorted subarrays, and this operation is independent of the initial order of the input. Therefore, the time complexity for the best, average and worst cases is always the same.
Space Analysis	Space Complexity = O(n) The maximum temporary array size can be the size of the entire main array. Although temporary arrays are created for each recursive call, these arrays are used only within the scope of that recursive call and are subsequently cleared from memory. However, at any given time period, the total temporary array size used does not exceed the size of the main array. Algorithm requires extra space equal to the size of the main array to complete.

General Comparison of the Algorithms

Time Complexity Comparison

Worst Case:

- **Quick Sort and Merge Sort (O(n log n))** - These algorithms typically perform well on large datasets. However, Quick Sort can degrade to O(n^2) if the pivot selection is poor.

- **Selection Sort and Bubble Sort (O(n^2))** - Both algorithms have a time complexity of O(n^2) in the worst case, which might be acceptable for small datasets but inefficient for larger ones.

Average Case:

- **Quick Sort and Merge Sort (O(n log n))** - Both are quite efficient in the average case and are preferred for large datasets.
- **Selection Sort and Bubble Sort (O(n^2))** - In the average scenario, these algorithms still operate with a time complexity of O(n^2), which is less efficient.

Best Case:

- **Bubble Sort (O(n))** - If the array is already sorted, Bubble Sort can operate in linear time, which is the fastest among the algorithms.
- **Quick Sort and Merge Sort (O(n log n))** - These algorithms perform at O(n log n) in the best case.
- **Selection Sort (O(n^2))** - Even in the best case, it requires a time complexity of O(n^2).

Space Complexity Comparison

- **Selection Sort and Bubble Sort (O(1))** - These algorithms can sort in-place, offering a constant space complexity. This means they do not require additional storage space.
- **Quick Sort (O(log n) - O(n))** - Ideally, Quick Sort has a space complexity of O(log n), but it can increase to O(n) in cases of unbalanced partitions.
- **Merge Sort (O(n))** - This algorithm needs extra space for its operations, which increases its memory usage but helps in faster sorting.