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

**SERDAR GENÇ**

1. **Selection Sort**

|  |  |
| --- | --- |
| **Time Analysis** | Selection Sort has a time complexity of O(n^2), where n is the number of elements in the array. It performs roughly (n^2-n)/2 comparisons and n-1 swaps in every scenerio, even if the array is already sorted. |
| **Space Analysis** | Selection Sort has a space complexity of O(1) because it sorts the array without requiring additional space. |

1. **Bubble Sort**

|  |  |
| --- | --- |
| **Time Analysis** | In the worst-case scenario, where the array is in reverse sorted order, Bubble Sort will require swapping adjacent elements in every pass, resulting in approximately (n^2-n)/2 swaps,  (n^2-n)/2 comparisons and time complexity of O(n^2). In the best-case scenario, where the array is already sorted, Bubble Sort will require no swaps, it will require n-1 comparisons and will have time complexity of O(N). |
| **Space Analysis** | Bubble Sort has a space complexity of O(1) because it sorts the array without requiring additional space. |

1. **Quick Sort**

|  |  |
| --- | --- |
| **Time Analysis** | The number of swaps and comparisons depend on the pivot selection strategy and the order of elements in the array. However, on average, Quick Sort performs O(n logn) comparisons and O(n logn) swaps and has time complexity of O(n logn) in best case where the median is always chosen as pivot. Because in this case the array is divided into two equal pieces at each step.The worst-case scenario occurs when the smallest or largest element is always chosen as the pivot. In this case, Quick Sort degrades to O(n^2) time complexity because each time, the size of the partition is reduced by only one element. In my code I take last element as pivot and worst-case occurs when i try to sort an array that is already sorted. |
| **Space Analysis** | Quick Sort has a space complexity of O(log n) for the recursive call stack due to the partitioning process. |

1. **Merge Sort**

|  |  |
| --- | --- |
| **Time Analysis** | Merge Sort divides the array into halves recursively until it reaches single-element arrays, and then merges them back together in sorted order. This process takes O(n logn) time in all cases. Merge Sort performs comparisons during the merging step. In every merge step, it compares elements from two sorted arrays to merge them. Again it performs O(n logn) comparisons in all cases. During merge sort no element swapping occurs because it doesn't rearrange elements within the original array. |
| **Space Analysis** | Merge Sort has a space complexity of O(n) because it requires additional space for the temporary arrays used during the merging process. |

**General Comparison of the Algorithms**

Selection Sort and Bubble Sort both have a time complexity of O(n^2) in the worst-case scenario, making them inefficient for large datasets. Quick Sort and Merge Sort have a time complexity of O(n log n) on average, making them much more efficient for larger datasets.

Bubble Sort performs relatively better in its best-case scenario when the array is already sorted, as it requires only O(n) comparisons and no swaps.

Quick Sort with poor pivot selection cause a worst-case time complexity of O(n^2) but it performs exceptionally well in its best-case scenario with a time complexity of O(n log n).

Merge Sort maintains its efficiency with a consistent time complexity of O(n log n) regardless of the input distribution, making it more reliable for general use.

Selection Sort and Bubble Sort have a space complexity of O(1), making them space-efficient. Merge Sort requires additional space for the temporary arrays used during the merging process, resulting in a space complexity of O(n). And Quick Sort, has a space complexity of O(log n) for the recursive call stack during partitioning.

Merge Sort is a stable sorting algorithm, this means that it preserves the order of elements if they are equal.