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

**YAHYA KEMAL KUYUMCU**

* **Selection Sort**

|  |  |
| --- | --- |
| **Time Analysis** | This algorithm operates in O(N^2) time complexity because it traverses the input data with two nested loops. The outer loop starts from index 0 and continues until index n-1, while the inner loop starts from the current outer index + 1 and iterates until it reaches the end of the array. We can express the time complexity as (n\*(n-1))/2, which evaluates to O(n^2). |
| **Space Analysis** | This algorithm does not create any subarrays or structures that would change based on the input's size. Therefore, we can conclude that it consumes constant space, denoted as O(1). |

* **Bubble Sort**

|  |  |
| --- | --- |
| **Time Analysis** | This algorithm is a linear search algorithm similar to selection sort. It operates in much the same way: by employing two nested loops to traverse the data. The outer loop iterates through the data n-1 times, while the inner loop compares two adjacent elements as it traverses the array. The use of these two loops categorizes this algorithm as quadratic time complexity, denoted by O(n^2). |
| **Space Analysis** | Because we dont create any subarrays or structures that changes size as input grows we can say that this algorithm takes constant space denoted by O(1). |

* **Quick Sort**

|  |  |
| --- | --- |
| **Time Analysis** | The algorithm consists of two main parts. Firstly, in the partitioning step, we determine the pivot of the array through O(n) comparisons. After locating the pivot's position, we divide the main array into two parts based on that index. Then, we recursively repeat the process for each of these subarrays. This dividing process takes log n time because we halve the array at each step. Consequently, due to these operations, we can assert that the QuickSort algorithm has a time complexity of O(n log n). |
| **Space Analysis** | The first part where we do the partitioning doesnt require any additional space depending on the input size because its operates on the main array just manipulating the indexes but there is variables which takes constant space so we can say that this part has a space complexity of O(1).  And the sorting part takes takes log n times because its halving the original array recursively, this recursion has log n space complexity on the method's stack. And combining these two parts we can say the algorithm has a space complexity of O(log n). |

* **Merge Sort**

|  |  |
| --- | --- |
| **Time Analysis** | The MergeSort algorithm operates in two main phases. Initially, it recursively divides the array into smaller halves, a process that consumes O(n log n) time. Each division involves splitting the array in half, which, when done recursively, results in log n calls, each taking O(n) time. Subsequently, in the merging phase, the sorted subarrays are merged back into the original array. This step also requires O(n) time per merge and recurses log n times. Consequently, the overall time complexity of MergeSort remains O(n log n). |
| **Space Analysis** | MergeSort's recursive calls necessitate the creation of temporary arrays, each with a size of n/2. With log n recursive calls, this results in a space complexity of n log n. Moreover, during the merging process, temporary arrays of size n/2 are utilized for each merge operation, recurring log n times. Consequently, the total space complexity for MergeSort is also O(n log n). |

**General Comparison of the Algorithms**

Choosing between these algorithms primarily depends on the size of the input data. For smaller datasets, opting for linear algorithms like selection sort or bubble sort is more efficient in terms of both space and time complexity. Linear algorithms are more efficient in terms of space complexity because they operate directly on the original array, whereas the other two algorithms utilize additional subarrays whose sizes vary based on the input data size.When comparing bubble sort and selection sort, they are nearly identical in efficiency considering both space and time complexity, as both algorithms traverse the data linearly.On the other hand, merge sort and quicksort are recursively implemented algorithms that use temporary arrays to sort the original array. This results in a less efficient space complexity compared to linear sort algorithms like bubble sort and selection sort. However, these algorithms are more efficient in terms of time complexity.In conclusion, for small datasets, linear algorithms like selection sort or bubble sort are preferable due to their efficiency in both space and time complexity. However, when dealing with larger datasets, I would opt for the quicksort algorithm. It outperforms linear algorithms in terms of time complexity and is more efficient in space complexity compared to merge sort. Therefore, for larger datasets, quicksort strikes a balance between time and space efficiency.