CSE222 / BİL505

Data Structures and Algorithms

Homework #6 – Report

Ahmet Özdemir

1) Selection Sort

Time Analysis	Each time the first loop runs n-1 times, the number of times the inner loop runs is progressively reduced so time compelexity is: n.(n-1) / 2 --> O(n^2)
Space Analysis	In this algorithm, no additional array etc. is created. Only a few loop variables (i and j) and a variable called minIndex are used. So space complexity: O(1), in-place sorting

2) Bubble Sort

Time Analysis	O(n): If the array is already sorted, the algorithm will not perform any substitution on the first pass and will terminate directly. In this case, the loop is the length of the array, i.e. until it's gone. O(n^2): Since each element is compared with every other element, the time complexity due to two nested loops is expressed as n^2. In the worst case, if all elements are in reverse order, the maximum number of comparisons and substitutions takes place.

Space Analysis	In this algorithm, no additional array etc. is created. Only a few loop variables (i and j) and a variable called minIndex are used. So space complexity: O(1), in-place sorting

3) Quick Sort

Time Analysis	O(nlogn): Quick sort works recursively to sort the sub-arrays resulting from partitioning array elements. In the best and average case, the partition operation splits the array elements into two nearly equal parts, which creates logn depths at each level, with n operations at each depth. O(n^2): If the partition operation consistently selects the largest or smallest element as the pivot, recursive calls will result in subarrays with a single element added or removed. This reduces the efficiency of the algorithm and n^2 leads to its complexity.
Space Analysis	In-place Sorting: O(logn) - Requires stack space for recursive calls.

4) Merge Sort

Time Analysis	O(nlogn): Merge Sort continuously splits the array into sub-arrays by splitting it in half, and then merges these sub-arrays by sorting them. Since this operation operates on n elements at each level and the decomposition depth is logn, for all cases works in the nlogn complex.

Space Analysis	O(n): Merge Sort uses temporary arrays of the same size as the main array. This means that the need for extra space during the sort operation is directly proportional to the size of the array. The total space required for temporary arrays for an array of n elements is It is expressed as O(n).

General Comparison of the Algorithms

This comparison will provide an overview of the performance of each sorting algorithm based on their time and space complexities.

Time Complexity:

1. Bubble Sort:

- Best Case: O(n) (if the array is already sorted)

- Average and Worst Case: O(n^2)

- Iteratively scans the array and compares elements to swap, which can be quite slow for large arrays.

2. Selection Sort:

- All Cases: O(n^2)

- Finds and places the smallest element at the front in each iteration. The number of comparisons is proportional to the square of the size of the array, making it unsuitable for large arrays.

3. Merge Sort:

- All Cases: O(n log n)

- Sorts by dividing and merging the array, efficient even for large data sets because of its consistent performance across different scenarios.

4. Quick Sort:

- Best and Average Case: O(n log n)

- Worst Case: O(n^2)

- Quick and efficient in the best and average cases due to the partitioning strategy that balances the divides. However, performance degrades to quadratic time if the smallest or largest elements are consistently chosen as pivots.

Space Complexity:

1. Bubble Sort and Selection Sort:

- Both operate in-place with a space complexity of O(1).

2. Merge Sort:

- Requires additional space proportional to the array size, with a space complexity of O(n), due to the temporary arrays used for merging.

3. Quick Sort:

- Typically has a space complexity of O(log n) because of the recursive calls, which require stack space.

Overall, while Bubble Sort and Selection Sort are simpler and may be sufficient for small datasets, Merge Sort and Quick Sort are better suited for handling larger datasets efficiently. Merge Sort provides stability and consistent performance, whereas Quick Sort can offer superior performance with the right pivot selection strategy.