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

AYKUT SERT

1) Selection Sort

Time Analysis	Selection sort does n-1 comparisons in each inner loop iteration, making n * (n-1)/2 comparisons in total. It is displaced at most once per iteration, resulting in at most n displacements in total. As a result, the time complexity of the selection sort is expressed as O(n^2) and can perform O(n^2) comparisons and O(n) substitutions in the best, worst and average cases.
Space Analysis	The space complexity of selection sort is expressed as O(1) because the algorithm uses fixed extra memory. Selection sort simply uses it to sort the elements within the array and does not require any additional data structure or memory space. A fixed number of extra variables are used during the process, and as the array size increases, the need for extra memory does not change but remains constant. Therefore, the space complexity of the selection sort is considered to be O(1).

2) Bubble Sort

Time Analysis	Comparison Counter: In the worst case, (n-1) comparisons are made for each element, so the total number of comparisons is (n-1) + (n-2) + ... + 1 = n * (n-1) / 2 It is possible. In this case, O(n^2) comparisons are made. Swap Counter: In the best case scenario there may be no swapping at all, in which case swap_counter is 0. In the worst case (when the array is in reverse order), swap_counter is approximately n * (n-1) / 2, since all elements are swapped with each other sequentially. Swapped: Checked at each loop iteration. In the best case (if the array is already sorted), no changes are made and the loop loops once. In the worst case (when the array is in reverse order), the loop loops n times as one or more changes are made at each iteration. According to this analysis, the time complexity of the bubble sort algorithm is O(n^2). In the worst case it can do O(n^2) comparisons and swaps. However, in the best case (if the array is already sorted), the number of comparisons may be O(n), but the number of substitutions will still be O(n^2).
Space Analysis	Bubble sort uses only one variable (temp) to temporarily store the elements of the array. Therefore, the space complexity of your bubble sort is expressed as O(1). Bubble sort does not require an additional data structure or memory space to sort an array. It just takes the existing array and performs the sorting by swapping within that array. Therefore, the space requirement of your bubble sort does not increase directly proportional to the array size, it remains constant. This makes it a very efficient sorting algorithm in terms of memory usage.

3) Quick Sort

Time Analysis	Comparison Counter: Quick sort compares each element of the array with a pivot element during each partition operation. A comparison is made on average for each element. This means that in the average case there will be O(n log n) comparisons. Swap Counter: Quick sort, elements may need to be swapped during each partition operation. However, each element is displaced only once. Therefore, the total number of displacements will be O(n log n) on average. As a result, we can say that the average time complexity of quick sort is O(n log n). In the best case (if the array is already sorted) or in the worst case (if the array is reverse-ordered), the time complexity will still be O(n log n).
Space Analysis	The space complexity of quick sort is expressed as O(n) in the worst case and O(log n) in the average case. In the worst case, the recursive depth is equal to the size of the array and a call stack is created at each level, resulting in O(n) extra memory consumption. However, in the average case, the recursive depth is logarithmic, and in this case the call stacks are logarithmic in number, resulting in O(log n) extra memory consumption.

4) Merge Sort

Time Analysis	Comparison Counter: Merge sort increments the comparison counter when performing comparisons between two subarrays. Since each subarray is already sorted, there is no comparison when combining two subarrays. Therefore, the total number of comparisons is the comparisons required to merge the subsequences. This means that there will be n/2 comparisons for each subarray, resulting in O(n log n) comparisons in total. Swap Counter: Merge sort does not perform swapping. Therefore, swap_counter is not used. As a result, the time complexity of merge sort is known as O(n log n). This algorithm performs splitting and merging of subarrays at each level. Each division operation takes logarithmic time, while each concatenation operation takes linear time to combine n-element arrays. Therefore, the total time complexity is O(n log n).
Space Analysis	Merge sort creates temporary auxiliary arrays to merge each subarray. These auxiliary sequences occur at each level and within each subsequence. Additionally, another temporary array is used to store all elements of the original array during merge operations at each level. In the worst case, your merge sort may consume O(n) extra memory on an array with n elements.

General Comparison of the Algorithms

- Bubble Sort, Selection Sort, Quick Sort and Merge Sort are four basic algorithms that are frequently used among sorting algorithms and have different performance

characteristics.Bubble Sort is a simple yet effective algorithm whose time complexity is usually expressed as O(n^2). In the worst case, each element is sorted by comparing it to the others, which can be slow on large data sets.

- Selection Sort is a simple algorithm, similar to bubble sort, and its time complexity is usually specified as O(n^2). However, unlike bubble sort, selection sort selects the minimum or maximum element to place an element in the correct position at each step, so it can be faster in some cases.
- Quick Sort is an algorithm based on divide and conquer strategy and usually has O(n log n) time complexity. It runs fast in the average case, but can have O(n^2) time complexity in the worst case. Although it is a fast algorithm, it can be memory intensive due to extra memory usage.
- Merge Sort is an algorithm also based on the divide and conquer strategy and usually has O(n log n) time complexity. Merge sort always has constant time complexity and is a stable sorting algorithm. It can be memory intensive due to extra memory usage, but typically uses O(n) extra memory.