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

Mehmet Can Olçay - 235008003006

1) Selection Sort

Time Analysis	For the all cases of selection sort it's time complexity is $O(n^2)$ because all of cases it goes through two for loop and there is no break . Also it's obvious in the screenshots , it has always same comparison counter time.It's comparision is unneceserray even its sorted array it checks 28 times.
	`Initial Array: 4 2 6 5 1 8 7 3`
	`Selection Sort => Comparison Counter: 28 Swap Counter: 7 Sorted Array: 1 2 3 4 5 6 7 8`
	`Initial Array: 1 2 3 4 5 6 7 8`
	`Selection Sort => Comparison Counter: 28 Swap Counter: 7 Sorted Array: 1 2 3 4 5 6 7 8`
	`Initial Array: 8 7 6 5 4 3 2 1`
	`Selection Sort => Comparison Counter: 28 Swap Counter: 7 Sorted Array: 1 2 3 4 5 6 7 8`

Space Analysis	Its space complexity is O(1) because its not use stack , it directly complete process without extra space.

2) Bubble Sort

Time Analysis	Its time complexity is O(n) for the best case because there is swap variable to check condition which is if any change did not happen inner for loop then break the loop.But other cases its time complexity is $O(n^2)$ because all of cases it goes through two for loop.
	$O(n^2)$ because all of cases it goes through two for loop.Worst Case.
	`Initial Array: 8 7 6 5 4 3 2 1`
	`Bubble Sort => Comparison Counter: 28 Swap Counter: 28 Sorted Array: 1 2 3 4 5 6 7 8`
	$O(n^2)$ because all of cases it goes through two for loop.Average case.Average and worst case comparision time nearly same but swap time is better in average case.
	`Initial Array: 4 2 6 5 1 8 7 3`
	`Bubble Sort => Comparison Counter: 27 Swap Counter: 12 Sorted Array: 1 2 3 4 5 6 7 8`
	O(n) because inner for loop breaks the loop.Best case.
	`Initial Array: 1 2 3 4 5 6 7 8`
	`Bubble Sort => Comparison Counter: 7 Swap Counter: 0 Sorted Array: 1 2 3 4 5 6 7 8`

Space Analysis	Its space complexity is O(1) because its not use stack , it directly complete process without extra space.

3) Quick Sort

Time Analysis	In the best-case scenario, Quick Sort has a time complexity of O(n log n). This occurs when the pivot chosen splits the array into two equal parts. Each partition step takes O(n) time, and there are approximately log n recursive calls. Best case is the select median as pivot and it divide two equal array .
	`Initial Array: 4 2 6 5 1 8 7 3`
	`Quick Sort => Comparison Counter: 14 Swap Counter: 10 Sorted Array: 1 2 3 4 5 6 7 8`
	In the worst-case scenario, Quick Sort has a time complexity of $O(n^2)$. This happens when the pivot chosen is either the smallest or largest element in the array, leading to unbalanced partitions even this scenario is the sortest one.It has too many swap and comparision times even sorted array.
	`Initial Array: 1 2 3 4 5 6 7 8`
	`Quick Sort => Comparison Counter: 28 Swap Counter: 35 Sorted Array: 1 2 3 4 5 6 7 8`
	One of other worst-case scenario even its swap counter is 19 because still there is 28 times comparision happens and it indicates that how pivot choice is important here because pivot is the last element according to my implemantation.
	`Initial Array: 8 7 6 5 4 3 2 1`
	`Quick Sort => Comparison Counter: 28 Swap Counter: 19 Sorted Array: 1 2 3 4 5 6 7 8`

Space Analysis	Quick Sort typically has a space complexity of O(log n) for the recursive call stack. This is because the recursion depth is at most log n, where n is the number of elements in the array.In the worst-case scenario, where Quick Sort encounters highly unbalanced partitions, it can consume O(n) space due to the depth of the recursion stack.

4) Merge Sort

Time Analysis	Merge Sort has a consistent time complexity of O(n log n) in all cases: best-case, worst-case, and average-case scenarios. It recursively divides the array into halves until each subarray has only one element, and then merges the subarrays in sorted order.Scenarios is not the matter.Thus sorting is stable. the comparison count in merge sort can vary depending on the input data and the algorithm's execution, but it typically has a time complexity of O(n log n). Comparision time nearly same the each other.
	`Initial Array: 4 2 6 5 1 8 7 3`
	`Merge Sort => Comparison Counter: 15 Swap Counter: 0 Sorted Array: 1 2 3 4 5 6 7 8`
	`Initial Array: 1 2 3 4 5 6 7 8`
	`Merge Sort => Comparison Counter: 12 Swap Counter: 0 Sorted Array: 1 2 3 4 5 6 7 8`
	`Initial Array: 8 7 6 5 4 3 2 1`
	`Merge Sort => Comparison Counter: 12 Swap Counter: 0 Sorted Array: 1 2 3 4 5 6 7 8`

Space Analysis	Space complexity is the O(n), Additional space is required for the temporary array used during merging.

General Comparison of the Algorithms

In summary, each sorting algorithm has its advantages and disadvantages. Selection sort and bubble sort are simple but inefficient for large datasets. Quick sort is fast on average but can have poor worst-case performance. Merge sort is stable and efficient, if we have enough memory , i prefer to use merge sort between these algorithims because its the both stable and fastest . But the choice of algorithm depends on factors such as the size of the dataset, stability requirements, and performance considerations.If we dont have enough memory we have to use slow algorithms and here my choice will be bubble sort is better than selection sort.

If there is enough space my decision will be :

Merge Sort > Quick Sort > Bubble Sort > Selection Sort

If there is no enough space my decision will be :

Buble Sort > Selection Sort > Quick Sort > Merge Sort