

# CSE222 HOMEWORK 7

# REPORT

Süleyman Koramaz

1901042615

# tıme complexıty analysıs

## merge sort

### worst-case

worst-case time complexity of the merge sort algorithm is O(n log n). The reason of this that algorithm repeatedly divides the input array into two halves until each half contains only one element. This process takes O(log n) time. The merge operation takes O(n) time because it iterates over each element in the input array once.

### best-case

Merge sort always divides the input array into two halves and recursively sorts them. Even if the input array is already sorted or partially sorted, the algorithm will still perform the same number of comparisons and merge operations to sort the array. So best-case time complexity of the merge sort algorithm is also O(n log n).

### avarage-case

Merge sort always divides the input array into two halves and recursively sorts them. Even if the input array is already sorted or partially sorted, the algorithm will still perform the same number of comparisons and merge operations to sort the array. So time average-case complexity of the merge sort algorithm is also O(n log n).

## selectıon sort

### worst-case

worst-case time complexity of the selection sort algorithm is O(n^2). The reason of this that it iterates all array which has a linear time complexity of O(n). It also compares each element with the current minimum and updates the minimum value accordingly which has also a linear time complexity of O(n).

### best-case

Selection sort always iterates all array and compares each element even if the input array is already sorted or partially sorted. So best-case time complexity of the selection sort algorithm is also O(n^2)

### avarage-case

Selection sort always iterates all array and compares each element even if the input array is already sorted or partially sorted. So avarage-case time complexity of the selection sort algorithm is also O(n^2)

## ınsertıon sort

### worst-case

The worst-case time complexity of the insertion sort algorithm is O(n^2). The reason of this that it iterates all array which has a linear time complexity of O(n). For each element in the unsorted portion, insertion sort needs to compare it with each element in the sorted portion until it finds the correct position. Which has also a linear time complexity of O(n).

### best-case

best-case time complexity of the insertion sort algorithm is O(n). The reason of this that it iterates all array which has a linear time complexity of O(n). But array is already sorted so it just iterates array.

### avarage-case

average-case time complexity of the insertion sort algorithm is O(n^2). The reason of this that the input array is randomly shuffled or has a random distribution of elements. While insertion sort may perform more efficiently on partially sorted or nearly sorted arrays, on average, the algorithm still requires a quadratic number of comparisons and shifting operations.

## bubble sort

### worst-case

worst-case time complexity of the bubble sort algorithm is O(n^2). The reason of this that in each pass of bubble sort, it compares adjacent elements and swaps them if they are in the wrong order. The number of comparisons in the worst case is roughly (n-1) + (n-2) + ... + 1, which sums up to n\*(n-1)/2. This results in a quadratic time complexity of O(n^2) for comparisons.

### best-case

best-case time complexity of the bubble sort algorithm is O(n). The reason of this that the number of comparisons required in the best case is n-1, resulting in a linear time complexity of O(n) for comparisons.

### avarage-case

average-case time complexity of the bubble sort algorithm is also O(n^2) . In the average-case scenario, the input array is randomly shuffled or has a random distribution of elements. While bubble sort may perform better on partially sorted arrays, on average, the algorithm still requires a quadratic number of comparisons and swaps.

## quıck sort

### worst-case

The worst-case time complexity of the quicksort algorithm is O(n^2). The reason of this that quick sort is a divide-and-conquer algorithm that selects a pivot element and partitions the array around the pivot. In the worst-case scenario, the choice of the pivot results in highly unbalanced partitions, causing the algorithm to make suboptimal splits. When quicksort encounters this unbalanced partitioning, the algorithm ends up performing inefficiently, resulting in a time complexity of O(n^2).

### best-case

The best-case time complexity of the quicksort algorithm is O(n log n). The reason of this that The best-case time complexity of the quicksort algorithm is O(n log n). In the best-case scenario, the choice of the pivot element consistently divides the array into two roughly equal subarrays at each recursive step In the best-case scenario, the recursion depth is logarithmic, as each partition reduces the problem size by a factor of two.

### avarage-case

Through mathematical analysis and probabilistic reasoning, it has been determined that the average-case time complexity of quicksort is O(n log n) the same as the best-case time complexity under the assumption of random input distributions.

# orıgınal maps

A screenshot of a computer screen

Description automatically generated with medium confidence

A screenshot of a computer screen

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

# SORTED MAPS

## worst-case

A screenshot of a computer program

Description automatically generated with medium confidence

# SORTED MAPS

## best-case

A screenshot of a computer program

Description automatically generated with medium confidence

# SORTED MAPS

## avarage-case

A screenshot of a computer program

Description automatically generated with medium confidence

# RUNNING TIME RESULTS

A screenshot of a computer screen

Description automatically generated with low confidence

# Comparison of the sorting algorithms

## worst-case

Merge sort algorithm is faster than the others in worst-case. Merge sort has time complexity of O(n.logn), other algorithms have time complexity of O(n^2).

## best-case

Insertion and Bubble sort algorithms are faster than the others in best-case. Insertion and Bubble sort have time complexity of O(n), Merge and Quick sort algorithms have time complexity of O(n.logn), Selection has time complexity of O(n^2).

## avarage-case

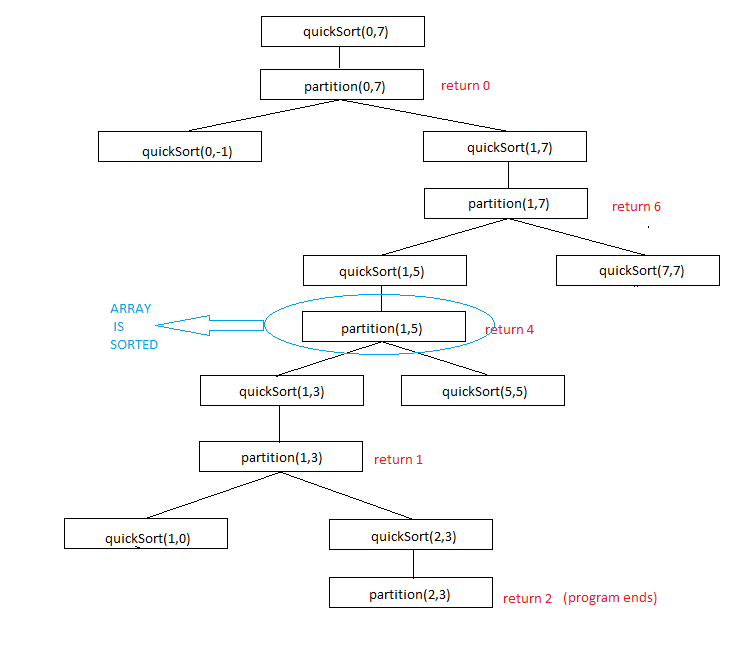
Merge and Quick sort algorithms are faster than the others in average-case. Merge and Quick sort algorithms have time complexity of O(n.logn), other algorithms have O(n^2).

# analyze keepıng the input ordering

Quick algorithm doesn’t keep the input ordering, the other algorithms keep the input ordering.

## Reason of why quick doesn’t keep ınput orderıng

Because program is working recursively. And array is sorted before the ending.



(“Buzzing bees buzz.”)

First of all, in first call of partition(0,7), The element that should be at the bottom of the row goes to the top.

This figure is the steps of the recursive function. Array is sorted in partition(1,5) step but program continues recursively. Even array is sorted if program call partition function and makes 1 change in any condition at the end of the function. These changes effect the order of the elements which have same value.

To sum, The element that should be at the bottom of the row will be top of the order and the rest will be mixed.