**TOPIC:**

**STUDENT NAME:**

**UNIVERSITY NAME:**

**COURSE NAME:**

**INSTRUCTOR NAME:**

**DATE OF SUBMISSION:**

**ANSWER 1:**

**Quick Sort:**

**Problem Solved**

Quick Sort is used for sorting an abstract list, it is an in-place sorting algorithm and follows divide and conquer approach and sort in non-increasing or non-decreasing manner. This places an array into two sub-arrays by selecting an element, known as pivot, then sorts these two sub-arrays.

**Key Steps**

Partitioning: Choose an element and put the elements so that all elements with value less than the selected element are to the left of the selected element, while all elements with value greater than the selected element are to the right of the selected element (Heuberger & Krenn, 2020).

Recursion: On the left and right sub-arrays continue the partitioning step all the way down until the basis case of a single-element or an empty array is hit.

Combination: As it was mentioned the partitioning stage rearranges the elements around the pivot do not require an additional combination step.

**Time Complexity Analysis**

Best Case: O(nlogn):

Worst Case: O(n^2)

Average Case: Ω (nlogn)

**Recurrence Relation**

For an array of size n and number of elements is k:

T(n) = T(k)+T(n-k-1)+ Θ(n)

Substitution method

Recursion tree method

The total work is:

Master Method

The recurrence fits the form with a=2,b=2 and f(n)=n

Therefore,

Since,

Therefore,

**Practical Implications**

The strength of Quick Sort is experienced particularly when used for sorting very large data sets because it is an efficient sorting algorithm whose overall complexity is rated O(nlogn) and does not use extra memory space to sort its data. In particular, its performance depends on balanced partitions, and therefore it is ineffective when used for pre-sorted or nearly sorted data unless a random pivot is chosen.

Merge Sort:

**Problem Solved**

Merge Sort is an efficient divide and rule sort that is also stabilizing and sorting elements in non-increasing or non-decreasing order. originals wire an array into two states, each state if recursively sorted and ultimately merging the twos (Godichon-Baggioni et al., 2023).

**Key Steps**

Divide: Divide the array in two equal parts, then divide each half and so on till the subarrays will contain only one element.

Conquer: The merge operation is to merge two subarrays and sort the array.

Combine: At the end of each merge process join the sorted subarrays together to form the fully sorted array.

Time Complexity Analysis

Best Case: O(nlogn)

Worst case: Θ(nlogn)

Average case: Ω(n log n)

**Recurrence relation**

Solution Using the Substitution Method

Solution Using the Recursion-Tree Method

At level 0 (the top level), the work done is Θ(n)

At level 1, the work done by the two recursive calls is Θ(n/2)+ Θ(n/2)= Θ(n)

Therefore,

Solution Using the Master Method

The recurrence T(n)=2T(n/2)+Θ(n) fits the form T(n)=aT(n/2)+f(n) with a=2,b=2 and f(n)=n

Therefore,

n^loga=n^log2=n

Since,

f(n)=n=n^loga

Therefore,

T(n)=Θ(nlogn)

**Practical Implications**

Merge Sort is optimized when there is a need to sort large-sized data sets. It has time complexity at O (n log n) and uses memory for merge operations, that is to be said why it might not be advisable for constrained memory environments. Merge sort is often in preference over scenarios where such predictable performances are required with respect to stable sorting over linked list sorting.

**Performance Matrices for both the algorithms:**

The output of the chosen performance metrics illustrates different efficiency profiles of Quick Sort and Merge Sort in terms of the time and memory consumption on various types of input data. Quick Sort showed the best result for a random array with the execution time of 0.004552 seconds and a memory consumption of 48.45 KB. It exhibited slightly higher times, most notably on sorted data (0.007111 seconds), but kept memory usage relatively tight across all datasets. As for Merge Sort, execution time stayed closer; 0.005686 seconds for sorted data, 0.003673 seconds for reverse sorted data, and 0.004232 for random data. Memory used for Merge Sort was also constant being 23.67 to 23.86 KB that is also justified by the fact that its time complexity of O(nlogn) does not depend on the type of input.

**Observed Discrepancies:**

As the theoretical analysis predicts, Quick Sort can run in O(n^2) when given sorted or reverse-sorted input but the observed time does not indicate a significant slowdown may be due to the small size of the data set used. As it will be seen in practice, Quick Sort also fluctuates the memory more due to recursive partitioning and the use of the stack for recursive calls, especially on lesser random data. On the other hand, Merge Sort can sustain stable measures due to its deterministic approach of splitting and merging which leads to better memory and execution estimates proportional to its O(n logn ) efficiency. The findings suggest that theoretical worst-case adversities established here are realistic for large-scale applications. The disparity observed here can be explained by the size of the data set, input structure, and overhead assumed by Python in terms of recursion and memory allocation peculiar to each algorithm.

**REFERENCES:**

Godichon-Baggioni, A., Werge, N., & Wintenberger, O. (2023). Non-asymptotic analysis of stochastic approximation algorithms for streaming data. *ESAIM: Probability and Statistics*, *27*, 482-514. <https://www.esaim-ps.org/articles/ps/pdf/2023/01/ps220013.pdf>

Heuberger, C., & Krenn, D. (2020). Asymptotic analysis of regular sequences. *Algorithmica*, *82*(3), 429-508. <https://link.springer.com/article/10.1007/s00453-019-00631-3>