Mathematical Derivation of the Average Runtime Complexity of Quicksort (Non-Random Pivot Version)

How Quick Sort works:

1. A single element is selected as the reference (pivot).
2. The array is partitioned and elements are rearranged into: Smaller than pivot, Equal to pivot and Greater than pivot.
3. Quick Sort is recursively applied on the partitions until sorting is complete.

Recurrence Relation for Expected Time Complexity

If the input array is randomly shuffled, the pivot generally splits the array into two nearly equal halves, meaning each subarray has an approximate size of n/2. The recurrence relation for the expected time complexity is:

T(n)=2T(n/2)+ O(n)

This recurrence represents two recursive calls (each of size n/2) plus the partitioning step, which takes O(n) time.

Solving the Recurrence Relation

Using the Master Theorem, the recurrence: **T(n) = n + 2T(n/2)**

Thus, the average runtime complexity of Quicksort is O(n log n), making it efficient in most cases.

Complexity Summary

- Best Case **O(n log n)**: Occurs when the pivot perfectly divides the array into two equal halves at every step.
- Worst Case **O(n^2)**: Happens when the pivot is always the smallest or largest element, leading to highly unbalanced partitions.
- Average Case **O(n log n)**: Typically maintains O(n log n) efficiency across randomized inputs, assuming an evenly distributed pivot choice.

Final Conclusion

Quicksort is a highly efficient comparison-based sorting algorithm. Although its worst-case complexity can reach O(n^2), choosing an optimal pivot or utilizing randomized Quicksort ensures an average runtime of O(nlogn).

Due to its efficiency, Quicksort is often favored for sorting large datasets, where it significantly outperforms simpler algorithms such as Bubble Sort and Insertion Sort.