Erdianti Wiga Putri A

13522053 – K01

**Rangkuman Algoritma dan Struktur Data**

**Chapter 4 – Speeding Up Your Code with Big O**

Big O Notation is a great tool for comparing competing algorithms, as it gives an objective way to measure them. Bubble Sort is a simple sorting algorithm that compares adjacent elements in an array and swaps them if they are in the wrong order. The process continues until the entire array is sorted, with each iteration known as a passthrough. The algorithm systematically moves through the array, comparing neighbouring elements and adjusting their positions as needed. The sorting process continues until a full passthrough is completed without any swaps, indicating that the array is fully sorted. Bubble Sort involves comparisons and swaps, with the total number of comparisons for an array of size N calculated using the formula (N - 1) + (N - 2) + (N - 3) ... + 1. In the worst-case scenario of a descendingly sorted array, the number of swaps equals the number of comparisons, resulting in a total step count twice that of the comparisons. This inefficiency becomes pronounced as the data size increases, showcasing a quadratic time complexity of O(N2), indicating a significant increase in steps as the data grows. This classification highlights Bubble Sort's relative inefficiency, which is graphically evident in the steep rise of steps as the data size expands. The hasDuplicateValue function utilizes nested loops, leading to a quadratic time complexity of O(N2). This approach is demonstrated through the tracking of comparisons, where an array of three elements incurs nine comparisons. The use of nested loops should signal potential inefficiencies, prompting consideration for alternative, more optimized solutions, especially when dealing with larger datasets to avoid performance issues. The updated hasDuplicateValue function utilizes a single loop and an array to track encountered numbers, effectively identifying duplicates without the need for nested iterations. By performing only N comparisons for N data elements, the algorithm demonstrates a linear time complexity of O(N). This optimization significantly improves the efficiency of the hasDuplicateValue function, making it a preferable choice, particularly when handling substantial data sets. Having a solid understanding of Big O Notation can allow us to identify slow code and select the faster of two competing algorithms.