|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Avg CPU Used (≈ 100 − %idle) | Avg Disk %util | Notes |
| Bubble Sort | ≈ 1.9 % | ≈ 0.4 % | |  | | --- | |  |   CPU light, minimal disk I/O (in-memory) |
| Insertion Sort | |  | | --- | |  |   ≈ 1.5 % | |  | | --- | |  |   ≈ 0.5 % | Similar pattern, slightly higher peaks |
| Merge Sort | ≈ 3.0 % | |  | | --- | |  |   ≈ 0.6 % | More CPU work due to recursion splits |
| Quick Sort | ≈ 2.0 % | ≈ 0.3 % | Balanced – fastest and efficient overall |

Summary Table

**Analysis :**

During the performance testing, I observed that all four sorting algorithms used very low CPU and disk resources since the data was processed entirely in memory. Among them, Merge Sort showed slightly higher CPU activity due to its recursive nature, while Quick sort performed efficiently with balanced CPU usage. Overall, the results confirm that the “memory database” design minimizes disk operations and keeps system utilization low.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Algorithm** | |  | **Real Time (s)** | | --- | --- | | **User Time (s)** | |  | **Sys Time (s)** | | --- | --- | | | **Notes** | | --- | |
| |  | | --- | | **Bubble Sort** | | 0.146 | 0.064 | 0.064 | Many swaps; higher real time on larger data |
| |  | | --- | | **Insertion Sort** | | 0.094 | 0.044 | 0.038 | Good on nearly-sorted data; sequential inserts |
| |  | | --- | | **Merge Sort** | | 0.072 | 0.040 | 0.026 | Divide-and-conquer; stable and efficient |
| **Quick Sort** | 0.128 | 0.062 | 0.062 | Usually fastest due to partitioning |

Sorting Performance (Real, User, Sys Time)

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **CPU Usage (%)** | **Time Complexity** | **Notes** |
| |  | | --- | | **Bubble Sort** | | ~99.6% | O(n^2) | Very high CPU usage; slow on large input |
| |  | | --- | | **Insertion Sort** | | ~97% | O(n^2) | Efficient for small data; sequential inserts |
| |  | | --- | | **Merge Sort** | | ~99% | O(n log n) | Stable; good balance between speed & CPU |
| **Quick Sort** | ~98.8% | O(n log n) | Fastest overall; best CPU efficiency |

CPU Performance Table (from TOP Command)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Time(Real Runtime)** | **Top(CPU %)** | **Vmstat(Avg CPU us+sy)** | **Iostat(Disk util%)** | **Notes** |
| |  | | --- | | **Bubble Sort** | | ~0.046 s (fastest runtime in test) | ~99.6 % | ~90–95 % | ~1–3 % | O(n²); very high CPU usage due to frequent swaps |
| |  | | --- | | **Insertion Sort** | | ~0.048 s | ~97 % | ~85–90 % | ~1–2 % | O(n²); performs better on nearly-sorted data |
| |  | | --- | | **Merge Sort** | | ~0.058 s | ~99 % | ~92–94 % | ~1–3 % | O(n log n); stable and efficient divide-and-conquer sort |
| **Quick Sort** | ~0.050 s | ~98.8 % | ~90–92 % | ~1–2 % | O(n log n); fastest overall due to partitioning |

Master Results Table