Mini Project – Text File Conversion

Compare the performance (time taken) of C, C++, Java, R, and Python programs for

1. Convert 200MB, 400 MB, 600 MB, 800 MB, and 1000MB text files to **upper case**.

**Sample Result**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **File Size** | **Time Taken (sec)** | | | | |
| C | C++ | Java | R | Python |
| 200MB | 12 | 15 | 18 | 20 | 25 |
| 400MB | 20 | 25 | 30 | 35 | 40 |
| 600MB | 34 | 36 | 40 | 45 | 53 |
| 800MB | 45 | 50 | 55 | 60 | 75 |
| 1000MB | 55 | 60 | 70 | 80 | 100 |

**Performance Insights**

1. **C**:
   * Fastest overall.
   * Close to hardware with minimal overhead.
   * Uses low-level memory operations (e.g., fread, fwrite).
2. **C++**:
   * Slightly slower than C due to object-oriented abstractions.
   * With efficient use of streams and buffers, still performs well.
3. **Java**:
   * Performs moderately well.
   * Slightly slower due to JVM overhead.
   * Garbage collection and I/O buffer management impact performance.
4. **R**:
   * Not optimized for heavy text processing.
   * Higher memory usage and slower I/O make it less suitable for such tasks.
5. **Python**:
   * Easiest to implement, but slowest.
   * Interpreted language with higher-level abstractions.
   * Performance can improve slightly with tools like NumPy or multi-threading, but not on par with compiled languages.

**🧠 Conclusion**

* For large-scale text transformations:
  + Use **C/C++** for **maximum performance**.
  + **Java** offers a balance between performance and portability.
  + **Python** and **R** are best suited for **smaller files** or **rapid prototyping**, not raw speed.