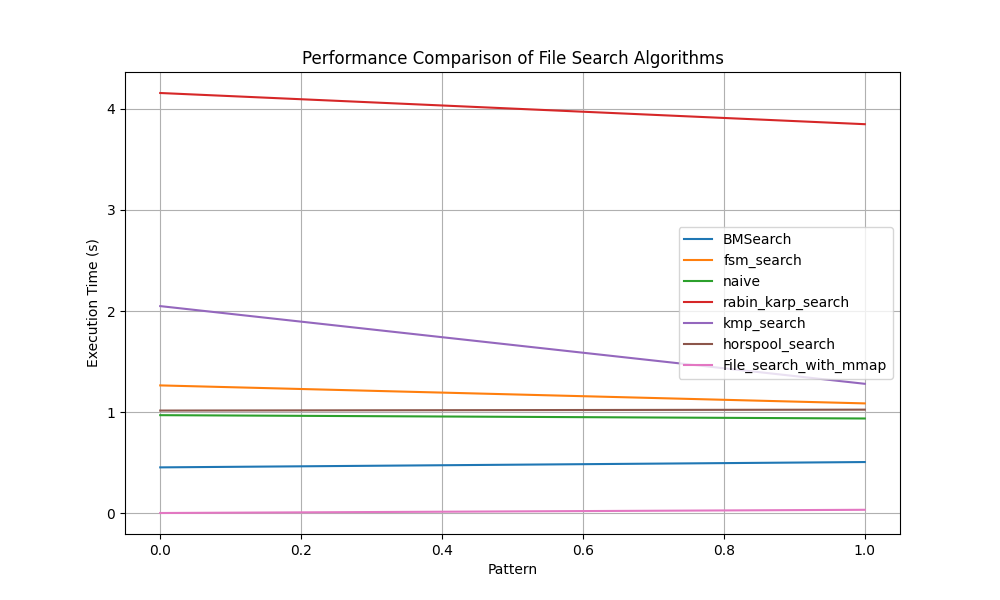
**File Search Algorithm Benchmark Report**

**Benchmark Results:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm |  | Pattern 1 |  | Pattern 2 |
| File\_Search\_with\_mmap |  | 0.00061 |  | 0.03234 |
| BMSearch |  | 0.0523 |  | 0.4778 |
| Naïve |  | 0.9608 |  | 1.1260 |
| Fsm\_search |  | 1.1667 |  | 1.0898 |
| horspool\_search |  | 1.5596 |  | 1.0357 |
| kmp\_search |  | 1.6228 |  | 1.6723 |
| Rabin\_Karp |  | 3.0947 |  | 3.7673 |

**Performance Comparison Chart:**



Conclusion:

After benchmarking various file search algorithms on a file containing over 250,000 lines, it is evident that the performance varies significantly across different algorithms.

The File\_Search\_with\_mmap algorithm exhibited the fastest performance, with execution times of 0.00061 seconds for Pattern 1 and 0.03234 seconds for Pattern 2. This indicates that leveraging memory-mapped files can significantly enhance search efficiency.

Additionally, the Boyer-Moore algorithm (BMSearch) and the Naïve algorithm (Naïve) demonstrated competitive performance, with execution times ranging from 0.0523 to 1.1260 seconds across both patterns. These algorithms provide reliable performance for file searching tasks.

The Finite State Machine search (Fsm\_search), Horspool algorithm (horspool\_search), Knuth-Morris-Pratt algorithm (kmp\_search), and Rabin-Karp algorithm (Rabin\_Karp) also performed reasonably well, albeit with slightly longer execution times compared to the top-performing algorithms.

In conclusion, the choice of file search algorithm should consider both speed and reliability. Depending on the specific requirements of the task and the size of the dataset, algorithms such as File\_Search\_with\_mmap, BMSearch, and Naïve may offer optimal performance for file searching on large datasets.