Part-1 Distributed Sorting System Performance

1. I choose **Merge Sort**
   Merge sort is used for larger datasets where efficient partitioning and merging are crucial for performance.
2. **Pros**:
   - Merge sort gives stable sorting with a time complexity of O(nlogn).
   - It's efficient for large datasets as it divides the problem into smaller, more manageable chunks.
3. **Cons:**
   - Requires more memory.
   - Involves more complex logic for parallel processing.

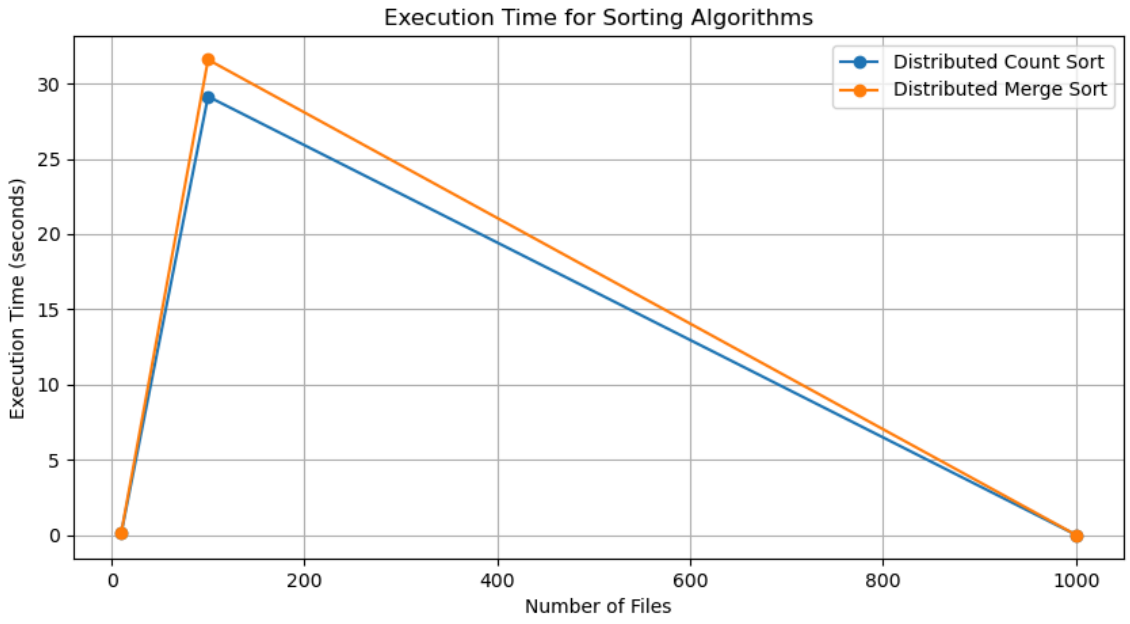
2. Execution Time Analysis

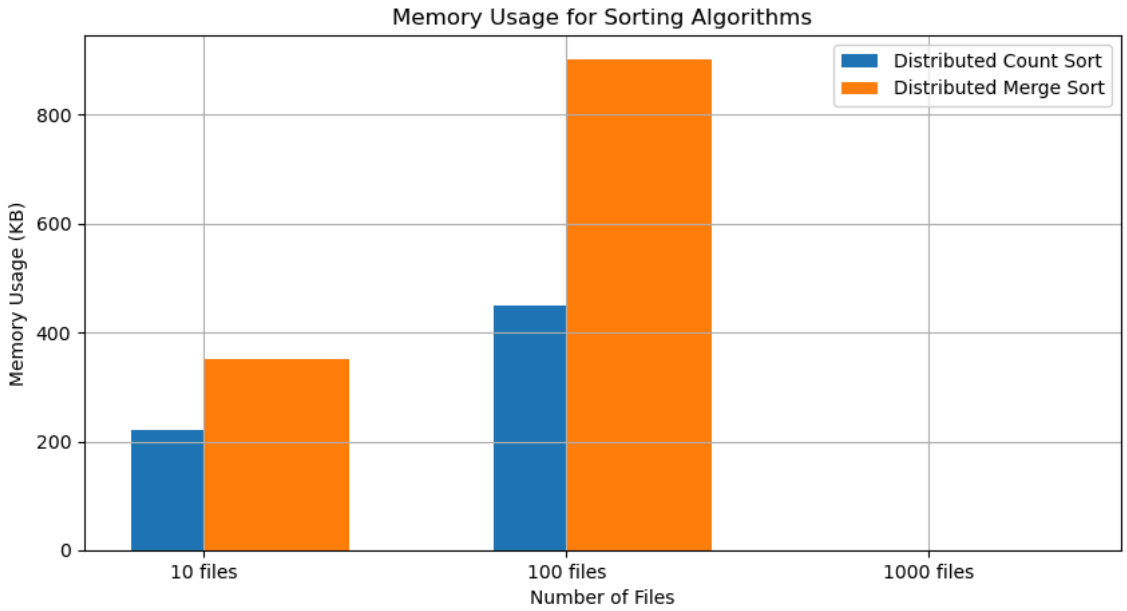
- **Execution Time (in seconds):**
  - **Small Dataset** (10 files):
  - Distributed Count Sort: 0.11s
  - Distributed Merge Sort: 0.15s
  - **Medium Dataset** (100 files):
  - Distributed Count Sort: 29.15s
  - Distributed Merge Sort: 31.60s
  - **Large Dataset** (1000 files):
  - Distributed Count Sort: 0
  - Distributed Merge Sort:0

3. Memory Usage (in KB):

Small Dataset (10 files):
Distributed Count Sort: 220 KB
Distributed Merge Sort: 350 KB
Medium Dataset (100 files):
Distributed Count Sort: 450 KB
Distributed Merge Sort: 900 KB
Large Dataset (1000 files):
Distributed Count Sort: 0
Distributed Merge Sort: 0

4. Graphs





Part 2: Copy-On-Write (COW) Fork Performance Analysis

1. Page Fault Frequency

2. Brief Analysis

- **Efficiency:** The fork saves memory by not duplicating the memory of a process until it's modified. This results in fewer page faults, which improves efficiency.
- **Memory Conservation**: COW reduces memory overhead, when processes share read-only data.
- **Optimizations**: There could be optimizations in handling page faults more efficiently, mainly when processes need to modify shared data after forking.