1 Algorithms for external storage

Problem 1. *External sorting* can handle massive amounts of data, especially when the data being sorted do not fit into the main memory (RAM) and must reside in the slower external memory (hard drive).

\[ CPU \longleftrightarrow RAM \longleftrightarrow Disk \]

Assumption: CPU and RAM operations are almost free, but Disk operation is expensive.

Goal: minimize disk accesses.

Strategy: sort-merge (In sorting phase, chunks of data small enough to fit in main memory are read, sorted and written out to a temporary file; In merge phase, the sorted subfiles are combined into a single larger file).

2 Disk Access Model

Assumptions:

- Problem size : N
- RAM size: M
- Disk Block size: B
- All reads/writes are in blocks
- All blocks take same time

2.1 Sorting in External Memory

Since there are \( N \) items on disk and memory can only store \( M \) items, there are \( \frac{N}{M} \) lists. Let us illustrate the process with a 2-merge sort process in Figure 2.1:

- Read M items from disk and sort by conventional method (such as quick sort or merge sort)
Figure 2.1: N items in disk

- Write the sorted data back to disk
- Repeat above steps until all $\frac{N}{M}$ lists are sorted
- Merge two near sublists and write the final result on disk

Operation Analysis:
- In each level, the major operation is reading and writing blocks. Thus, it is $\frac{N}{M} \log \frac{N}{M}$ I/Os.
- Based on the 2-merge sort property, there are log $\frac{N}{M}$ levels given $\frac{N}{M}$ initial elements.
- Total I/Os is $O \left( \frac{N}{M} \log \frac{N}{M} \right)$.

Instead of 2-merge sort, a more common way is to merge elements from all $\frac{N}{M}$ lists together, which forms the $\frac{N}{M}$-merge sort. In this case:
- In each level, the major operation is still $\frac{N}{M} \log \frac{N}{M}$ I/Os.
- There are log $\frac{N}{M}$ $\frac{N}{M}$ levels given $\frac{N}{M}$ initial elements.
- Total I/Os is $O \left( \frac{N}{M} \log \frac{N}{M} \right)$.

2.2 $\Omega(n \log n)$ Lower Bound for In-memory Sorting

Assume, there are $n$ items in memory:
- There are $n!$ possible initial orderings
- If algorithms never performs more than $k$ comparisons, then it can sort at most $2^k$ input arrangements

So we need $2^k \geq n!$, then $k \geq \log_2 n! \approx n \log_2 n$. Thus, the lower bound for in-memory operation is $\Omega(n \log n)$. 
2.3 lower bounds for external memory sorting

Assume there are \( N \) elements in disk:

- There are \( N! \) possible input permutations
- Give algorithm for free; initial sort of \( M \)-sizes chunks \( \frac{N!}{(M!)^M} \)

Upon loading new block into memory, \( \binom{M}{B} \) outcomes from all comparisons.

\[
\log \left( \binom{M}{B} \right) \frac{N!}{(M!)^M}
\]

\[
= \log \left( \binom{M}{B} \right) \frac{N!}{(M!)^M} = \log \left( \binom{M}{B} \right)
\]

\[
\approx N \log N - \frac{N}{M} M \log M = N \log \frac{N}{M}
\]

\[
\geq \frac{N \log \frac{N}{M}}{\log \frac{M}{B}} \approx \frac{N \log \frac{N}{M}}{B \log M - B \log B}
\]

\[
= \frac{N \log \frac{N}{M}}{B \log \frac{M}{B}} = \frac{N}{B} \log \frac{M}{M} = \Omega \left( \frac{N}{B} \log \frac{M}{M} \right)
\]