

The Input/Output Complexity of Sorting and Related Problems

...

By: Alok Aggarwal and Jeffrey Scott Vitter (1988)

Presented by: Aitor Arrese-Igor

Overview

Paper provides bounds for Input and Outputs (I/O) between internal and external memory for the following algorithms

1. Sorting
2. Fast Fourier Transform
3. Permutation Networks
4. Permuting
5. Matrix Transposition

Introduction

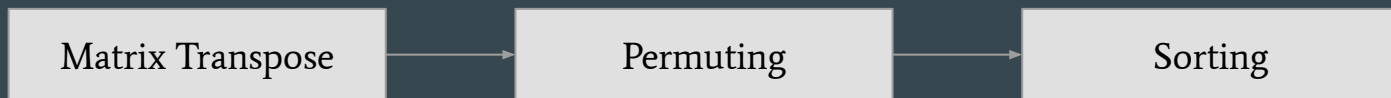
- Sorting accounts for roughly $\frac{1}{4}$ of all computer cycles, vastly due to external sorts
- External sorts is when a file is too large and must reside in secondary storage
 - Bottleneck in external sorting is the time for I/O between internal and secondary memory
 - Extremely large sorts are becoming increasingly common

Related Work: Floyd (1972)

Floyd studied a two level storage model with fast memory + slow storage

Bounded matrix transposition under **restricted params** at $O(N \log N / B)$

Merge Sort was established with a matching bound



This work left the open question of the results to all unrestricted M, B, P

Input and Outputs (I/O)

N = # records to Sort (File Size)

M = # records that can fit into internal memory (Memory Size)

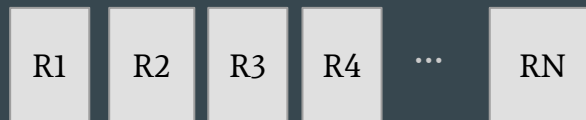
B = # records that can be transferred in a single block (Block Size)

P = # blocks that can be transferred concurrently

Internal Memory
Capacity = M records



Disk
Contains N records



Sorting

- Task of sorting can be broken into two steps
 - Determining the correct order
 - Moving the records into that order



Permuting

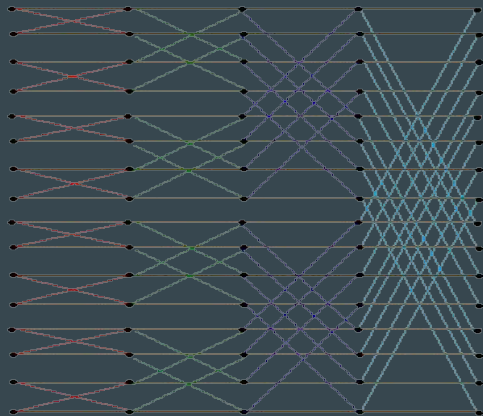
- Effectively sorting without needing to determine the ordering
 - Isolates the cost of I/O data movement
- Need to support all $N!$ possible orderings
 - Matrix Transposing is a subproblem as it is a structured permutation

Record	A	B	C	D
Destination	3	1	4	2



Fast Fourier Transform

- FFT becomes an I/O scheduling problem over a butterfly graph
- Every node depends on two nodes from the previous stage
 - Require both parents to be completed and in RAM to compute child
 - Challenge is to schedule data movements while respecting data dependencies



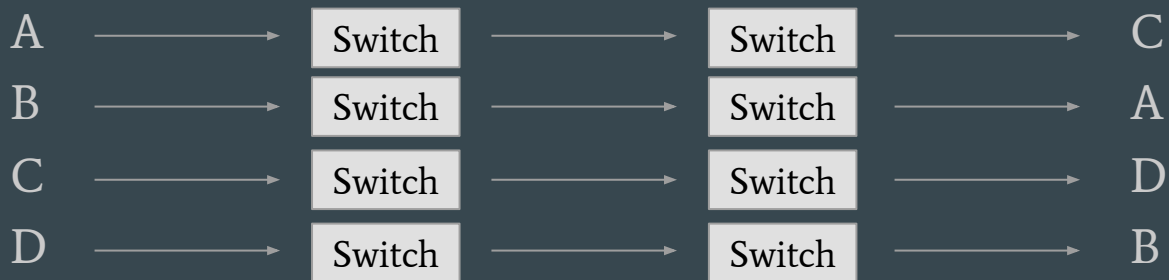
$\log_2 N$ depth

Permutation Network

- A fixed routing network that can realize any $N!$ permutation



- Computation proceeds stage by stage and can be processed through pebbling



How many I/Os are Necessary

THEOREM 3.1. *The average-case and worst-case number of I/Os required for sorting N records and for computing the N -input FFT digraph is*

$$\Theta\left(\frac{N}{PB} \frac{\log(1 + N/B)}{\log(1 + M/B)}\right). \quad (3.1)$$

$\frac{N}{PB}$ is the cost of moving through the file

$\frac{\log(1 + N/B)}{\log(1 + M/B)}$ Is the number of levels / passes

Simplifying the I/O Model

Lemma 4.1: Any computation can be transformed into one using only simple I/Os without using more I/Os

Proof: Work backwards and delete every I/O transfer that the final output does not directly depend on.

- Exactly one copy of every record exists, so we can track how many records move and rearrange

Permuting: Two Strategies

THEOREM 3.2. *The average-case and worst-case number of I/Os required to permute N records is*

$$\Theta\left(\min\left\{\frac{N}{P}, \frac{N}{PB} \frac{\log(1 + N/B)}{\log(1 + M/B)}\right\}\right). \quad (3.4)$$

1. When M and B are very small with respect to N , we can move the records naively and directly one at a time
2. Otherwise, the problem is just as hard as the task of sorting, which demonstrates that the action of routing the records is the dominating cost of sorting

Permuting Lower Bound

If T I/Os can generate fewer than $N!$ possible arrangements, then not all permutation can be produced.

$$(B!)^{N/B} * (\text{possibilities per I/O})^T \geq$$

- There are N/B first $\overset{N!}{\text{reads}}$ which can be fully permuted
- Possibilities per I/O depends on # of tracks to select B from and reorganizing in M
 - Explain on Chalk

$$T = \Omega\left(\min\left\{N, \frac{N \log(N/B)}{B \log(M/B)}\right\}\right).$$

FFT & Permute Networks Lower Bound

- A Permute Network can be constructed by stacking 3 FFTs
- I/Os cannot depend on the desired permutation as they are fixed
 - This eliminates the aspect of choosing a track from the computation

$$O\left(\frac{N}{PB} \frac{\log(1 + N/B)}{\log(1 + M/B)}\right).$$

Sorting Lower Bound

- Permuting is a special case of sorting, so its lower bound applies unless $B \log(M/B) = o(\log(N/B))$,
 - In this case Permuting lower bound only gives $\Omega(N/P)$
- Assume Adversary chooses the comparisons to maximize # orderings still possible

$$(B!)^{N/B} * (M \text{ choose } B)^T \geq N!$$

$$T = \Omega\left(\frac{N \log(N/B)}{B \log(M/B)}\right).$$

Matrix Transpose Lower Bound

THEOREM 3.3. *The number of I/Os required to transpose a $p \times q$ matrix stored in row-major order, is*

$$\Theta\left(\frac{N}{PB} \frac{\log \min\{M, 1 + \min\{p, q\}, 1 + N/B\}}{\log(1 + M/B)}\right). \quad (3.5)$$

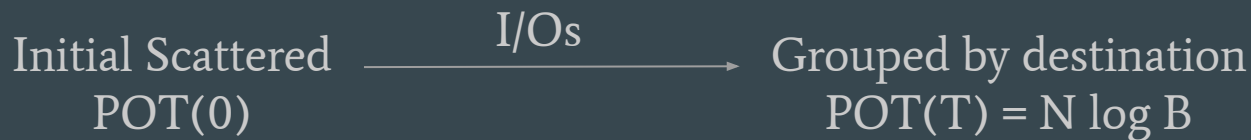
$$f(x) = \begin{cases} x \log x, & \text{if } x > 0; \\ 0, & \text{if } x = 0. \end{cases}$$

$$C_k(t) = \sum_{1 \leq i \leq N/B} f(x_{i,k})$$

$$\text{POT}(t) = C_M(t) + \sum_{k \geq 1} C_k(t).$$

- **Target group** : the B records that belong together in one final output track
- The potential rewards records from the same target group being concentrated together
- At the end there will be N/B tracks with each having a score of $B \log B$

Matrix Transpose: Bounding I/O



I/Os

- Output: cannot increase potential
- Input: Bring in at most B new records into memory, that contains at most $M - B$

$$\nabla \text{POT}(t) = O\left(B \log \frac{M}{B}\right).$$

$$T - \frac{N}{B} = \Omega\left(\frac{\text{POT}(T) - \text{POT}(0)}{B \log(M/B)}\right).$$

Merge Sort Upper Bound

1. Run Formation

- Load M records in Memory
- Sort them in RAM
- Write back as a sorted run
- Produce N/M sorted Runs

2. Merge Runs

- Keep one block from each run in memory
- Merge $M/B - 1$ runs simultaneously
- Repeat until only one sorted run remains

$M = 8, B = 2$

Unsorted File

[7 2 9 1] [8 6 4 9] [2 3 9 5]

Run Formation

[1 2 7 9] [4 6 8 9] [2 3 5 9]

Merge Runs

[1 2] [2] [] [7 9]

[4 6] [4 6] [4 6] [4 6]

[2 3] [2 3] [2 3] [2 3]

[] [1] [1 2] []

Merge Sort: I/O Complexity

Cost per pass = N/B

Initially have N/M runs, with each pass reducing by a factor of M/B

merge passes $\approx \log_{M/B}(N/M)$

$$O\left(\frac{N \log(N/B')}{B' \log(M/B')}\right).$$

P-Block lookahead is needed to reach final upper bound

- Places endmarkers in the tracks so the algorithm can forecast next P tracks to expire

Distribution Sort

1. Find approximate partitioning elements $b_1 < b_2 < \dots < b_S$ $S = \sqrt{M/B}$
2. Distribute data into the buckets
3. Recursively sort each bucket

I/O Cost

- Partitioning the entire file costs $O(N/(PB))$
- Each recursive level reduces problem size by roughly $\sqrt{M/B}$
- Stop at size M such that: $N/S^L < M$

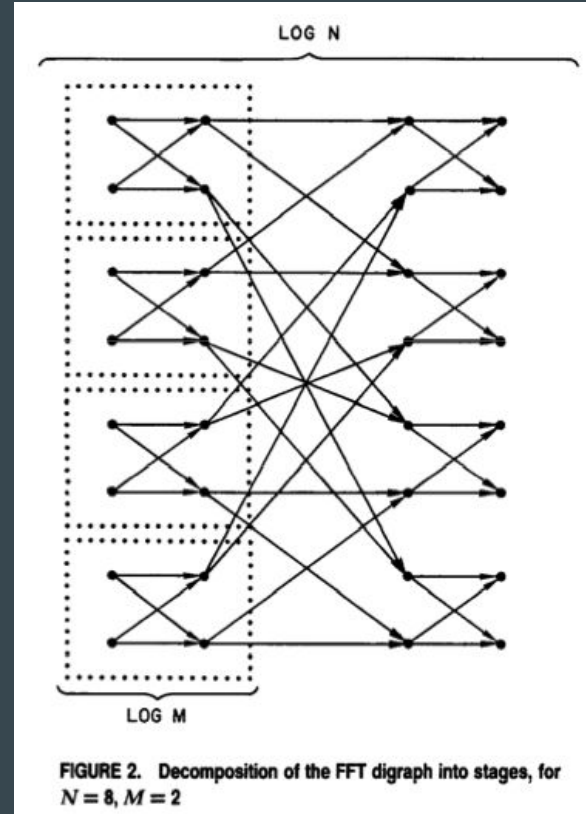
$$L = O(\log_S N/M)$$

$$L = O\left(\frac{N \log(N/B')}{B' \log(M/B')}\right)$$

FFT: Upper Bound

- FFT has $\log N$ levels
- Ram holds M records, so if we collect correct M we can process a size M FFT, meaning $\log M$
- This creates $\log N / \log M$ stages
- Transpose to accurately arrange for the next stage
 - Using simple merging procedure bounds the transposes at
- $O((N/(PB)) \log_{M/B} \min\{M, N/M\})$ I/Os.
- The I/O cost is all from transposing each stage

$$O\left(\frac{N \log N}{PB \log M} \left(\frac{\log \min\{M, N/B\}}{\log(M/B)}\right)\right),$$



Matrix Transposition: Upper Bound

The goal is to gradually bring records that belong in the same final block together

Algorithm Process:

- Identify target subgroups
- Load up to M records and combine pieces with same destination
- Repeat: each pass makes every target subgroup M/B times larger until it reaches B

I/O Cost

- Each pass scans the file once $O(N/(PB))$
- Each pass grows each group by M/B

Questions

1. “Can we remove our assumption that the records are indivisible and allow, for example, arbitrary bit manipulations and dissections of the records?”
 - a. Has been progress assuming major network-coding conjecture
2. The model treats internal computation as free to isolate I/O cost. How could we model both costs simultaneously?
3. How do these bounds and optimal algorithms hold in practice?
 - a. I expected experiments that measured CPU runtime vs I/O time

Opinions

- I thought proving I/Os cost in Sorting by also bounding similar problems like permutations was an interesting approach
- Paper ordering is particularly hard to follow
 - Solving 5 different problems at the same time and provides complex theorems before any analysis
- Extremely theory & proof heavy without experiments section or diagrams