The Input/Output Complexity of Sorting and Related Problems

Paper by: Alok Aggarwal and Jeffrey Scott Vitter

Presentation by: Bryan Chen

Outline

- 1. Motivation
- 2. Problem Statement(s)
- 3. Definitions/Setup
- 4. Analysis
- 5: Reflection
- 6: Discussion

Motivation

➔ Sorting is (really) expensive!
- Accounts for ~¼ of all computer cycles still
- Many of these are due to "external sorts" when file is large

Motivation

- ➔ Sorting is (really) expensive!
  - Accounts for ~¼ of all computer cycles still
  - Many of these are due to "external sorts" when file is large
- ➔ Bottleneck
  - I/O between internal memory and external storage (6.004)

Problem Statement(s)

- ➔ Model of extended memory (x):

Internal Memory (M) Secondary Storage (>=N)

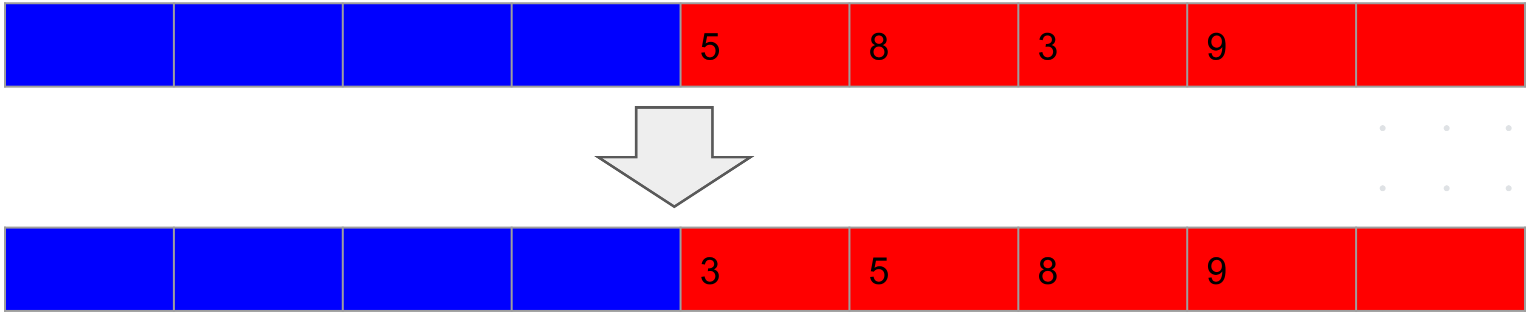
- ➔ 1-indexing: internal memory is x[1] to x[M], disk is x[M+1]...
- ➔ We want to find bounds on I/O for five similar problems

1. Sorting

➔ Problem: given records in disk, sort and replace in disk

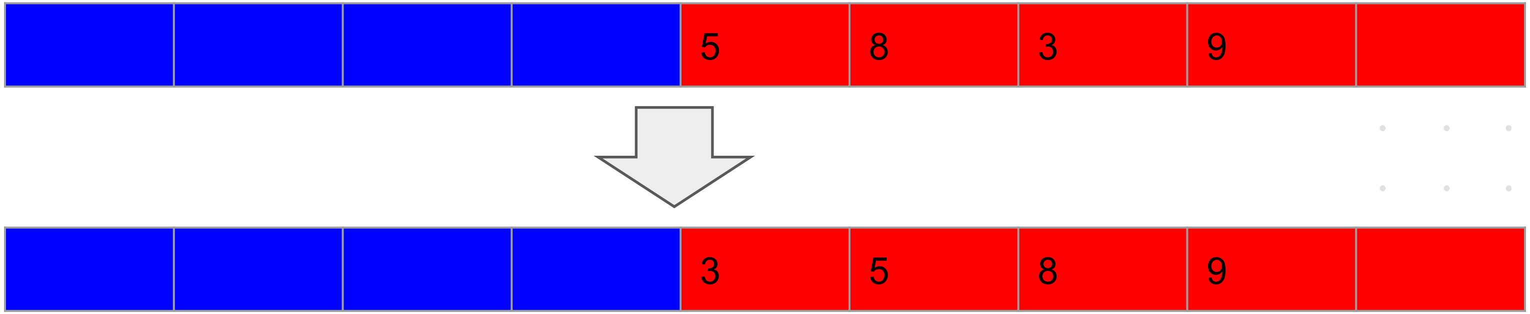
1. Sorting

➔ Problem: given records in disk, sort and replace in disk



1. Sorting

➔ Problem: given records in disk, sort and replace in disk



➔ Explicitly: given x[i] = nil for $1 \le i \le M$, x[M+i] = R_i for $1 \le i \le N$, sort them in nondecreasing order by their key values in x[M+1]...x[M+N]

2. Fast Fourier Transform (FFT)

➔ Setup: N is a power of 2, everything else same as sorting

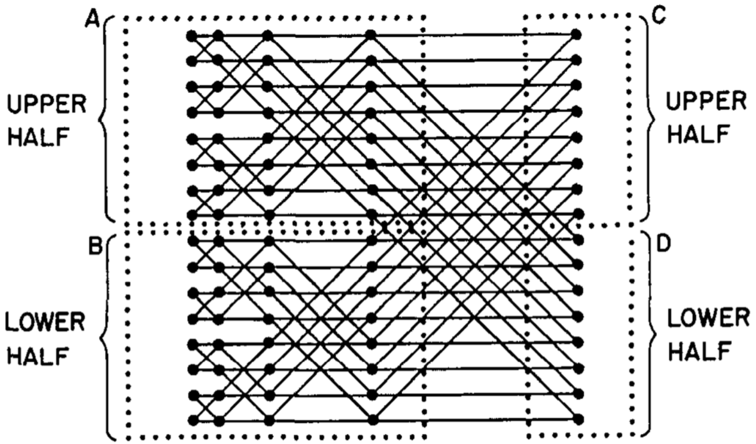
- $x[i] = nil$ for $1 \le i \le M$, $x[M+i] = R_i$ for $1 \le i \le N$

2. Fast Fourier Transform (FFT)

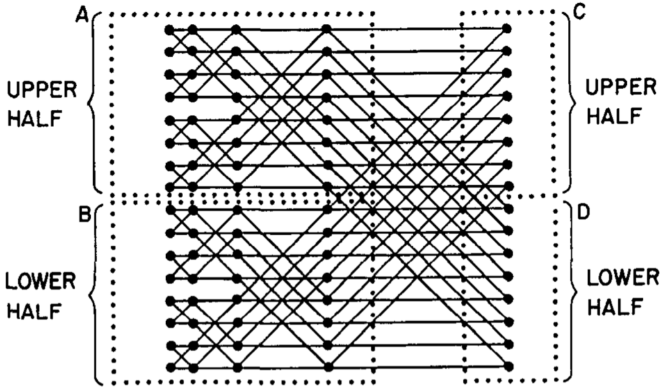
- ➔ Setup: N is a power of 2, everything else same as sorting
  - $x[i] = nil$ for $1 \le i \le M$, $x[M+i] = R_i$ for $1 \le i \le N$
- ➔ Goal: N output nodes from FFT directed graph (aka digraph) are pebbled and memory configuration stays the same

2. Fast Fourier Transform (FFT)

- ➔ Setup: N is a power of 2, everything else same as sorting
  - $x[i] = nil$ for $1 \le i \le M$, $x[M+i] = R_i$ for $1 \le i \le N$
- ➔ Goal: N output nodes from FFT directed graph (aka digraph) are pebbled and memory configuration stays the same

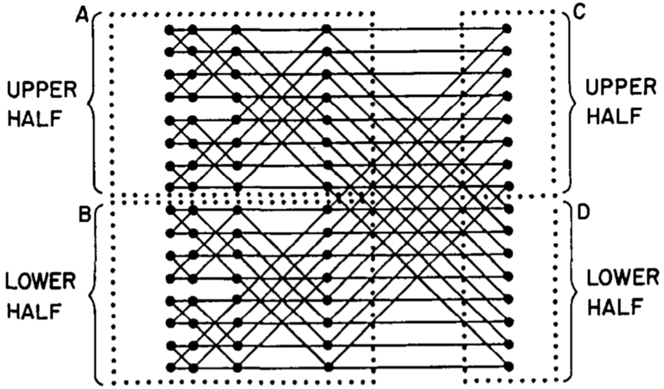


The FFT Digraph



- N rows
- (log N) + 1 columns
- Designate node at row i, column j as $n_{i,j}$ (0-indexed)
- Each $n_{i,j}$ is connected to $n_{i,j-1}$ and $n_{i \text{ XOR } (1<<j-1),j-1}$
- Can only pebble $n_{i,j}$ if predecessors are pebbled and their records are in internal memory

The FFT Digraph



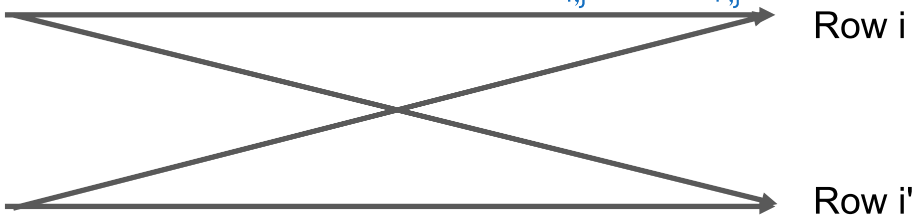
- ➔ N rows
- ➔ (log N) + 1 columns
- ➔ Designate node at row i, column j as $n_{i,j}$ (0-indexed)
- ➔ Each $n_{i,j}$ is connected to $n_{i,j-1}$ and $n_{i \text{ XOR } (1<<j-1),j-1}$
- ➔ Can only pebble $n_{i,j}$ if predecessors are pebbled and their records are in internal memory
- ➔ Way to delegate flow of records in/out of internal memory and disk

3. Permutation Network

- ➔ Similar setup/description to FFT, but different graph shape
  - N rows, J+1 columns for some $J \geq \log N$
  - Allows for creation of permutations/different calculations
- ➔ Keeping same notation of $n_{i,j}$, $n_{i,j}$ is connected to $n_{i,j-1}$ and possibly $n_{i',j-1}$
  - If this is the case, then $n_{i',j}$ is also connected to $n_{i,j-1}$

3. Permutation Network

- ➔ Similar setup/description to FFT, but different graph shape
  - N rows, J+1 columns for some $J \geq \log N$
  - Allows for creation of permutations/different calculations
- ➔ Keeping same notation of $n_{i,j}$, $n_{i,j}$ is connected to $n_{i,j-1}$ and possibly $n_{i',j-1}$
  - If this is the case, then $n_{i',j}$ is also connected to $n_{i,j-1}$
  - This represents a switch between $n_{i,j}$ and $n_{i',j}$ from the preceding column:



The Permutation Digraph

- ➔ The prior constraints on the digraph are not sufficient to determine a permutation network
  - For each of the N! permutations on {1...N}, we must be able to set the switches so that each $n_{i,0}$ is mapped to some output node $n_{p\_i,j}$
    - This mapping represents the permutation
- ➔ Nodes can only be pebbled if predecessors have been pebbled and the corresponding records are in internal memory (as before)

4. Permuting

- Same setup as sorting, but the keys of the N records must form a permutation of {1…N}
- Not the same as permutation networks!

Permuting vs. Permutation Network

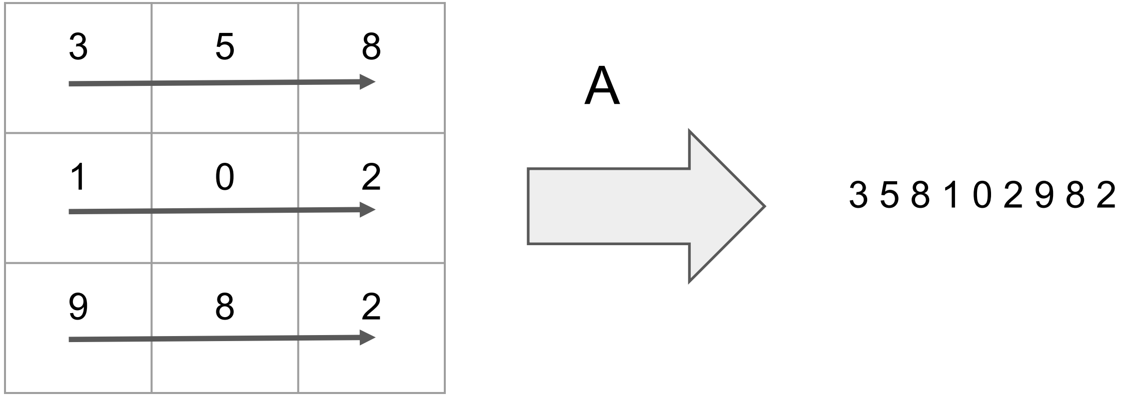
- ➔ A permutation network is generated by a sequence of I/Os to represent ways to generate N! permutations
- ➔ A permutation is a set of specific I/Os to end up with that permutation of records

Permuting vs. Permutation Network

- ➔ A permutation network is generated by a sequence of I/Os to represent ways to generate N! permutations
- ➔ A permutation is a set of specific I/Os to end up with that permutation of records
- ➔ In short:
  - Network = Same I/Os let you generate N! permutations
  - Permutation = I/Os depend on the specific permutation

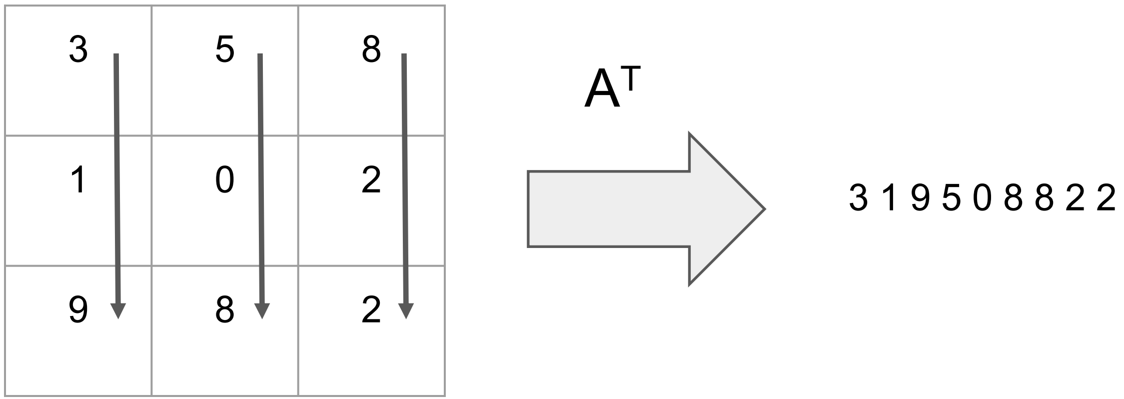
5. Matrix Transposition

- ➔ Setup: A *p* x *q* matrix (pq = N) of records is stored in row-major order on disk starting from x[M+1], internal storage empty
- ➔ Goal: Replace with column-major order on disk starting from x[M+1], with internal storage empty



5. Matrix Transposition

- ➔ Setup: A *p*x*q* matrix (pq = N) of records is stored in row-major order on disk starting from x[M+1], internal storage empty
- ➔ Goal: Replace with column-major order on disk starting from x[M+1], with internal storage empty



Definitions/Setup

- An I/O operation is *simple* if each record transfer involves being removed from disk (I)/internal memory (O) and placed into an empty spot in internal memory (I)/disk (O)
  - Full movement of data

Definitions/Setup

- An I/O operation is *simple* if each record transfer involves being removed from disk (I)/internal memory (O) and placed into an empty spot in internal memory (I)/disk (O)
  - Full movement of data
- Denote the k^{th} ($k \geq 1$) set of B continuous locations on disk as the *k^{th} track* (x[M+(k-1)B+1]…x[M+kB])
  - Since B is the block size, we consider transfers of exactly B records (a complete track)
    - Overflow is handled by simply filling the extra records with nil

Definitions/Setup

- For simplicity, we mostly consider $P=1$
  - Represents conventional disks
  - Can also shift bounds by a factor of P
- Assume WLOG that B, M, N are powers of 2 and $B < M < N$

Analysis (All)

- ➔ Observation: for FFT, permutation network, and matrix transposition, there is no input distribution
  - Average-case and worst-case models are the same
- ➔ For sorting and permuting, assume all N! inputs are equally likely
  - A bit more nuance must be used to separate the models

Analysis (All)

- ➔ Lemma 4.1: *for each computation that implements a permutation of the N records $R_1, R_2, \ldots, R_N$ (or that sorts or that transposes or that computes the FFT digraph or a permutation network), there is a corresponding computation strategy involving only simple I/Os such that the total number of I/Os is no greater*
  - Cancel transfer of a record if transfer is not needed for the final result
  - Resulting I/O strategy is simple
- ➔ From now on, assume all I/Os are simple

Analysis (Permuting)

- ➔ Lemma 4.1 allows us to take the following approach:
  - Bound number of possible permutations after t I/Os
  - At each I/O, we create more possible permutations between records
- ➔ Consider time t
  - At $t = 0$, we have 1 permutation (haven't done anything yet)
  - At most $N/B + t - 1$ full tracks before the t^{th} output
  - Thus, we have at most $N/B + t$ new places for a new full track to be
    - Amount of space between tracks doesn't matter when considering permutation
- ➔ Thus, at each timestep t we multiply the number of permutations by at most $N/B + t$
  - Can be bounded by $N(1 + \log N)$ for simplicity

Analysis (Permuting)

- ➔ Now, consider each block of B records from a given track
  - Could have B! possible orders based on rearrangement of internal memory
    - Implies an increase in number of possible permutations
- ➔ Apply mathematical computation and Stirling's approximation

Analysis (Permuting)

- ➔ Now, consider each block of B records from a given track
  - Could have B! possible orders based on rearrangement of internal memory
    - Implies an increase in number of possible permutations
- ➔ Apply mathematical computation and Stirling's approximation

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

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

Analysis (Permuting)

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

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

- ➔ It turns out that for small B and M, it's better to just do the naive solution of moving records once per block transfer

Analysis (FFT/Permutation Networks)

- Observation: by stacking three FFT digraphs, we can construct a permutation network
  - The output nodes of one FFT digraph should be the input nodes of the next
  - We need three FFT digraphs to capture all permutation possibilities
- This makes the two problems essentially equivalent in terms of I/O
  - Lower bound for permutation networks matches upper bound for FFT

Analysis (~~FFT/~~Permutation Networks)

- ➔ Observation: by stacking three FFT digraphs, we can construct a permutation network
  - The output nodes of one FFT digraph should be the input nodes of the next
  - We need three FFT digraphs to capture all permutation possibilities
- ➔ This makes the two problems essentially equivalent in terms of I/O
  - Lower bound for permutation networks matches upper bound for FFT
  - Consider permutation networks

Analysis (Permutation Networks)

- ➔ Observation: I/O sequence is fixed for a permutation network
- ➔ Can still apply similar analysis to permuting
  - Allows us to not have to deal with the fact that I/O depends on a desired permutation
  - Records transferred during I/O and track accessed are fixed for each I/O
    - Eliminates N/B + t term that we had to account for earlier
- ➔ In addition, each output can at most double the amount of generated permutations
  - Due to the swapping formulation of permutation network nodes' predecessors

Analysis (Permutation Networks)

- ➔ Apply Stirling's approximation again

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when $B\log(1+M/B) = o(\log(1+N/B))$. *The average-case and worst-case number of I/Os required for computing any N-input permutation network is*

$$\Omega\left(\frac{N}{PB}\frac{\log(1+N/B)}{\log(1+M/B)}\right); \tag{3.2}$$

Analysis (Permutation Networks)

- ➔ Apply Stirling's approximation again
- ➔ The Ω-bound is tight: permutation networks exist with necessary I/Os equal to

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

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when $B\log(1+M/B) = o(\log(1+N/B))$. *The average-case and worst-case number of I/Os required for computing any N-input permutation network is*

$$\Omega\left(\frac{N}{PB}\frac{\log(1+N/B)}{\log(1+M/B)}\right); \tag{3.2}$$

Analysis (Sorting)

- ➔ Can immediately derive bounds based on permuting
  - Sorting is just one special case of permuting
- ➔ But we can do better!
  - Sorting is different since we know which permutation to generate based on what belongs where

Analysis (Sorting)

- ➔ Can immediately derive bounds based on permuting
  - Sorting is just one special case of permuting
- ➔ But we can do better!
  - Sorting is different since we know which permutation to generate based on what belongs where
- ➔ Adversarial argument for lower-bound:
  - In the worst case, all our simple I/Os compare against all the records in internal memory
  - At each step, when importing B new records, we will need to compare against at most M-B records

Analysis (Sorting)

- ➔ After similar mathematical computation to permutation networks, we can arrive at the following (familiar) bound for worst-case:

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*

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

Analysis (Sorting)

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*

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

- ➔ We wish to show this bound also holds in the average-case scenario
- ➔ Consider the comparison tree with N! leaves, representing all the N! orderings possible
  - Each node represents an input operation

Analysis (Sorting)

- ➔ Each node has a degree of at most M choose B, except for nodes corresponding to track input
  - These have degree at most B!(M choose B)
  - There can be at most N/B of these
- ➔ Taking the sum of leaf depths divided by N! (averaging), minimizing over all possible computation trees, and normalizing P yields the same bounds as Theorem 3.1

Analysis (Matrix Transposition)

- Approach using a potential function POT(t) over time t
- WLOG, assume p and q (matrix dimensions) are powers of 2
- Let the i^{th} target group ($1 \le i \le N/B$) be the set of records we want in the i^{th} track in the end

Analysis (Matrix Transposition)

- Let f(x) be a continuous function:

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

- Define togetherness to the k^{th} track at time t:

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

  - $x_{i,k}$ records contained by k^{th} track belonging to i^{th} target group
- Also define togetherness of internal memory

$$C_M(t) = \sum_{1 \leq i \leq N/B} f(y_i)$$

  - y_i is number of records belonging to i^{th} target group in memory

Analysis (Matrix Transposition)

➔ Define the potential function POT(t) to be the sum of togetherness ratings of internal memory and all tracks

- At the end of the algorithm, internal memory is empty and each track has all the records it should have

Analysis (Matrix Transposition)

- ➔ Define the potential function POT(t) to be the sum of togetherness ratings of internal memory and all tracks
  - At the end of the algorithm, internal memory is empty and each track has all the records it should have
  - Internal memory contributes 0 to potential, each block contributes Blog B
    - Thus, at termination I/O number T, POT(T) = Nlog B

Analysis (Matrix Transposition)

- ➔ Define the potential function POT(t) to be the sum of togetherness ratings of internal memory and all tracks
  - At the end of the algorithm, internal memory is empty and each track has all the records it should have
  - Internal memory contributes 0 to potential, each block contributes Blog B
    - Thus, at termination I/O number T, POT(T) = Nlog B
- ➔ Casework at t=0 based on size of B vs. p and q yields

$$\mathrm{POT}(0) = \begin{cases} 0, & \text{if } B < \min\{p,q\}; \\ N \log \dfrac{B}{\min\{p,q\}}, & \text{if } \min\{p,q\} \le B \le \max\{p,q\}; \\ N \log \dfrac{B^2}{N}, & \text{f } \max\{p,q\} < B; \end{cases} \qquad (4.15)$$

Analysis (Matrix Transposition)

- ➔ Properties of the potential function
  - Does not increase when a block is output from internal memory to disk
  - Increases when a track is input from disk to internal memory
- ➔ Can show inputs cause potential function to increase by O(B log M/B)
  - Monitoring how togetherness changes in independent components and applying a convexity argument

Analysis (Matrix Transposition)

➔ We can use the potential function to then prove:

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

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

Analysis (Conclusion)

➔ Of the theorems we've seen, all the non-trivial bounds include N/PB

- N/PB is the number of I/O operations necessary to scan N records once

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when B log(1 + M/B) = o(log(1 + N/B)). The average-case and worst-case number of I/Os required for computing any N-input permutation network is

$$\Omega\left(\frac{N}{PB}\frac{\log(1+N/B)}{\log(1+M/B)}\right); \tag{3.2}$$

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

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

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

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

Analysis (Conclusion)

- ➔ Of the theorems we've seen, all the non-trivial bounds include N/PB
  - N/PB is the number of I/O operations necessary to scan N records once
- ➔ Coefficients of N/PB represent number of "passes" through file
  - A "linear-time" algorithm in terms of passes would use O(N/PB) I/Os
  - Thus, the terms below intuitively indicate degree of nonlinearity

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when $B\log(1+M/B) = o(\log(1+N/B))$. The average-case and worst-case number of I/Os required for computing any N-input permutation network is

$$\Omega\left(\frac{N}{PB}\frac{\log(1+N/B)}{\log(1+M/B)}\right); \quad (3.2)$$

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

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

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

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

Analysis (Optimal Algorithms)

- ➔ Some promising theoretical constraints are presented
  - Unclear how exactly they constrain explicit algorithms
- ➔ Leads to discussion of "optimal algorithms" which are able to achieve bounds in Theorems 3.1-3.3
- ➔ WLOG, again assume $B < M < N$ are all powers of 2
- ➔ Suffices to just consider worst-case complexity: average-case result follows immediately

Analysis (Optimal Algorithms: Sorting)

➔ Two sorting algorithms examined: merge and distribution

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when $B\log(1+M/B) = o(\log(1+N/B))$. *The average-case and worst-case number of I/Os required for computing any N-input permutation network is*

$$\Omega\left(\frac{N}{PB}\frac{\log(1+N/B)}{\log(1+M/B)}\right); \quad (3.2)$$

Analysis (Optimal Algorithms: Sorting)

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when $B\log(1 + M/B) = o(\log(1 + N/B))$. *The average-case and worst-case number of I/Os required for computing any N-input permutation network is*

$$\Omega\left(\frac{N}{PB}\frac{\log(1 + N/B)}{\log(1 + M/B)}\right); \quad (3.2)$$

➔ Merge Sort

- Divide and conquer algorithm
- Given an array of unsorted records:
  - Split them into two halves
  - Sort each half independently (e.g. with mergesort)
  - Combine them iteratively

7	6	3	1	8	4

Analysis (Optimal Algorithms: Sorting)

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when $B\log(1+M/B) = o(\log(1+N/B))$. *The average-case and worst-case number of I/Os required for computing any N-input permutation network is*

$$\Omega\left(\frac{N}{PB}\frac{\log(1+N/B)}{\log(1+M/B)}\right); \quad (3.2)$$

- ➔ Merge Sort
  - Divide and conquer algorithm
  - Given an array of unsorted records:
    - Split them into two halves
    - Sort each half independently (e.g. with mergesort)
    - Combine them iteratively

7	6	3	1	8	4

Analysis (Optimal Algorithms: Sorting)

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when $B\log(1 + M/B) = o(\log(1 + N/B))$. *The average-case and worst-case number of I/Os required for computing any N-input permutation network is*

$$\Omega\left(\frac{N}{PB}\frac{\log(1 + N/B)}{\log(1 + M/B)}\right); \tag{3.2}$$

➔ Merge Sort

- Divide and conquer algorithm
- Given an array of unsorted records:
  - Split them into two halves
  - Sort each half independently (e.g. with mergesort)
  - Combine them iteratively

3	6	7		1	4	8

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when $B\log(1+M/B) = o(\log(1+N/B))$. *The average-case and worst-case number of I/Os required for computing any N-input permutation network is*

$$\Omega\left(\frac{N}{PB}\frac{\log(1+N/B)}{\log(1+M/B)}\right); \quad (3.2)$$

Analysis (Optimal Algorithms: Sorting)

➔ Merge Sort

- Divide and conquer algorithm
- Given an array of unsorted records:
  - Split them into two halves
  - Sort each half independently (e.g. with mergesort)
  - Combine them iteratively

3	6	7

1	4	8

1					

Analysis (Optimal Algorithms: Sorting)

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when $B\log(1+M/B) = o(\log(1+N/B))$. *The average-case and worst-case number of I/Os required for computing any N-input permutation network is*

$$\Omega\left(\frac{N}{PB}\frac{\log(1+N/B)}{\log(1+M/B)}\right); \qquad (3.2)$$

➔ Merge Sort

- Divide and conquer algorithm
- Given an array of unsorted records:
  - Split them into two halves
  - Sort each half independently (e.g. with mergesort)
  - Combine them iteratively

3	6	7

1	4	8

1	3				

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when $B \log(1 + M/B) = o(\log(1 + N/B))$. *The average-case and worst-case number of I/Os required for computing any N-input permutation network is*

$$\Omega\left(\frac{N}{PB}\frac{\log(1 + N/B)}{\log(1 + M/B)}\right); \quad (3.2)$$

Analysis (Optimal Algorithms: Sorting)

➔ Merge Sort

- Divide and conquer algorithm
- Given an array of unsorted records:
  - Split them into two halves
  - Sort each half independently (e.g. with mergesort)
  - Combine them iteratively

3	6	7

1	4	8

1	3	4	6	7	8

Analysis (Optimal Algorithms: Sorting)

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when $B\log(1+M/B) = o(\log(1+N/B))$. The average-case and worst-case number of I/Os required for computing any N-input permutation network is

$$\Omega\left(\frac{N}{PB}\frac{\log(1+N/B)}{\log(1+M/B)}\right); \quad (3.2)$$

- ➔ Merge Sort
  - Divide and conquer algorithm
  - Given an array of unsorted records:
    - Split them into two halves
    - Sort each half independently (e.g. with mergesort)
    - Combine them iteratively

1	3	4	6	7	8

Analysis (Optimal Algorithms: Sorting)

- ➔ Merge Sort
- ➔ I/O Complexity
  - In this formulation, we sort our tracks into runs and then combine them
  - At each merging step, one block being merged resides in internal memory
    - We don't know which block to fetch next

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when $B\log(1 + M/B) = o(\log(1 + N/B))$. *The average-case and worst-case number of I/Os required for computing any N-input permutation network is*

$$\Omega\left(\frac{N}{PB}\frac{\log(1 + N/B)}{\log(1 + M/B)}\right); \qquad (3.2)$$

Analysis (Optimal Algorithms: Sorting)

- ➔ Merge Sort
- ➔ I/O Complexity
  - In this formulation, we sort our tracks into runs and then combine them
  - At each merging step, one block being merged resides in internal memory
    - We don't know which block to fetch next
  - In each track, we can place P-1 markers telling us the key values of the next P-1 tracks of the run
    - Using this forecasting information, we can achieve the bounds in Theorem 3.1

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when $B\log(1+M/B) = o(\log(1+N/B))$. *The average-case and worst-case number of I/Os required for computing any N-input permutation network is*

$$\Omega\left(\frac{N}{PB}\frac{\log(1+N/B)}{\log(1+M/B)}\right); \qquad (3.2)$$

Analysis (Optimal Algorithms: Sorting)

➔ Distribution Sort

- Assume M/B is a perfect square, and let S = √(M/B)
- With O(N/PB) I/Os we can find S partitioning elements to bucket the records
    - Can repeatedly use a linear rank-finding algorithm via median of medians (e.g. 6.046)
- Recursively sort the buckets

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when $B\log(1 + M/B) = o(\log(1 + N/B))$. *The average-case and worst-case number of I/Os required for computing any N-input permutation network is*

$$\Omega\left(\frac{N}{PB}\frac{\log(1 + N/B)}{\log(1 + M/B)}\right); \qquad (3.2)$$

Analysis (Optimal Algorithms: Sorting)

➔ Distribution Sort

➔ I/O Complexity

- Define T(n) to be I/Os needed to sort n records

$$T(N) = \sum_{1 \le i \le S+1} T(N_i) + O\left(\frac{N}{PB}\right).$$

➔ By solving this, we get the bounds in Theorem 3.1

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when $B \log(1 + M/B) = o(\log(1 + N/B))$. *The average-case and worst-case number of I/Os required for computing any N-input permutation network is*

$$\Omega\left(\frac{N}{PB}\frac{\log(1 + N/B)}{\log(1 + M/B)}\right); \quad (3.2)$$

Analysis (Optimal Algorithms: Permuting)

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

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

- ➔ Special case of the sorting algorithm
  - Can just re-use it here to get the same bounds as sorting
  - Additional naive case when B and M are small for the N/P min term
    - If B log(M/B) = o(log N/B)
- ➔ Either of the two above prior sorting algorithms will work

Analysis (Optimal Algorithms: Matrix Transposition)

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

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

- ➔ WLOG, assume p and q are powers of 2
- ➔ In each track of B records, partition records into *target sub-groups* based on end location
  - Merge these target subgroups over course of algorithm
    - Records in same target subgroup should stay together
  - If x is the initial size of target-subgroups, then:

$$x = \begin{cases} 1, & \text{if } B < \min\{p, q\}; \\ \dfrac{B}{\min\{p, q\}}, & \text{if } \min\{p, q\} \le B \le \max\{p, q\}; \\ \dfrac{B^2}{N}, & \text{if } \max\{p, q\} < B. \end{cases} \qquad (5.6)$$

Analysis (Optimal Algorithms: Matrix Transposition)

➔ I/O Complexity

- Each pass (O(N/PB) I/Os) multiplies size of target subgroup by M/B
- Calculating amount of passes (and thus number of I/Os) results in Theorem 3.3

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

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

Analysis (Optimal Algorithms: FFT/Permutation Network)

➔ Again, three FFT digraphs form a permutation network

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when $B\log(1+M/B) = o(\log(1+N/B))$. *The average-case and worst-case number of I/Os required for computing any N-input permutation network is*

$$\Omega\left(\frac{N}{PB}\frac{\log(1+N/B)}{\log(1+M/B)}\right); \quad (3.2)$$

Analysis (Optimal Algorithms: FFT~~/Permutation Network)~~

➔ Again, three FFT digraphs form a permutation network

- Only consider FFT

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when $B\log(1+M/B) = o(\log(1+N/B))$. *The average-case and worst-case number of I/Os required for computing any N-input permutation network is*

$$\Omega\left(\frac{N}{PB}\frac{\log(1+N/B)}{\log(1+M/B)}\right); \tag{3.2}$$

Analysis (Optimal Algorithms: FFT)

- ➔ For simplicity, assume log M divides log N
- ➔ Decompose FFT graph into (log N)/(log M) equal size stages by column

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when $B\log(1+M/B) = o(\log(1+N/B))$. *The average-case and worst-case number of I/Os required for computing any N-input permutation network is*

$$\Omega\left(\frac{N}{PB}\frac{\log(1+N/B)}{\log(1+M/B)}\right); \tag{3.2}$$

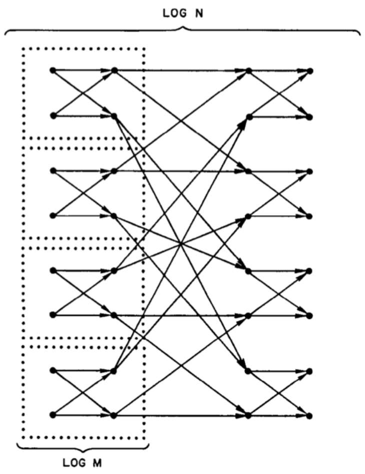


FIGURE 2. Decomposition of the FFT digraph into stages, for $N = 8$, $M = 2$

Analysis (Optimal Algorithms: FFT)

- ➔ For simplicity, assume log M divides log N
- ➔ Decompose FFT graph into (log N)/(log M) equal size stages by column
  - Stage k (1 ≤ k ≤ (log N)/(log M)) corresponds to pebbling of columns (k-1)log M to klog M in the digraph
  - To compute pebbling on these log M columns, we must take M columns as input via a transposition permutation

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when B log(1 + M/B) = o(log(1 + N/B)). The average-case and worst-case number of I/Os required for computing any N-input permutation network is

$$\Omega\left(\frac{N}{PB}\frac{\log(1+N/B)}{\log(1+M/B)}\right); \quad (3.2)$$

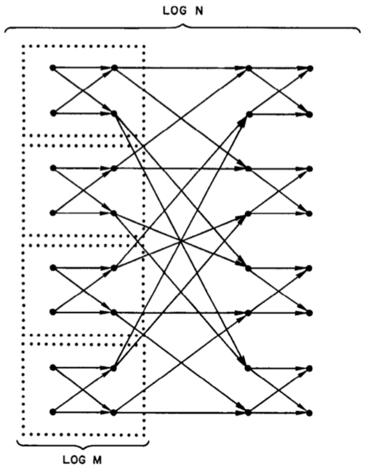


FIGURE 2. Decomposition of the FFT digraph into stages, for $N = 8$, $M = 2$

Analysis (Optimal Algorithms: FFT)

- ➔ For simplicity, assume log M divides log N
- ➔ Decompose FFT graph into (log N)/(log M) equal size stages by column
  - Stage k (1 ≤ k ≤ (log N)/(log M)) corresponds to pebbling of columns (k-1)log M to klog M in the digraph
  - To compute pebbling on these log M columns, we must take M columns as input via a transposition permutation
- ➔ By combining prior results and some mathematical reorganization, get Theorem 3.1

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*

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

For the sorting lower bound, the comparison model is used, but only for the case when M and B are extremely small with respect to N, namely, when B log(1 + M/B) = o(log(1 + N/B)). The average-case and worst-case number of I/Os required for computing any N-input permutation network is

$$\Omega\left(\frac{N}{PB}\frac{\log(1+N/B)}{\log(1+M/B)}\right); \tag{3.2}$$

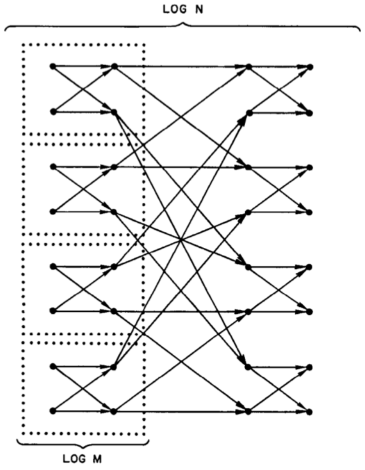


FIGURE 2. Decomposition of the FFT digraph into stages, for $N = 8$, $M = 2$

Reflection (Strengths)

- ➔ Introduction is solid with interesting premise
  - Presents a common problem with a relatable issue
- ➔ Good amount of content in the paper
  - Theoretical analysis of I/O
  - Several similar useful problems presented
  - Corresponding algorithms for each of the five problems shown

Reflection (Weaknesses)

- ➔ Organization style somewhat lackluster
  - Lots of context switching between the five problems in each section
    - Not consistent: theorems presented in different order than they are proved
  - Somewhat hurts readability
  - Intimidating theorems presented upright instead of leading into them
- ➔ Use of jargon is sometimes unexplained/excessive
  - e.g. "pebbling" is not explained
- ➔ Math is quite involved, sometimes lacking explanation
  - Some more diagrams could also be helpful as a visual break
    - Problem formulation, algorithms, etc.

Reflection (Future Work)

- ➔ Assumption that records are indivisible simplifies much of the theoretical computation
  - Being able to operate on individual bits could be useful, but hard to examine
  - Would help gain insight on information transfer
- ➔ Analysis on other memory paradigms to represent different kinds of systems
- ➔ Implementing real memory checks to examine performance in practice

Discussion Questions

- → Did you find the paper's relatively unique flow confusing? Or did it help to relate the problems together? When was it helpful/harmful?
- → Why can we assume WLOG that $B < M < N$? Or that they are powers of 2?
- → What are some applications for the graph formulation of FFT the authors provide?
- → Do you think the paper has aged well in the past ~30 years with all the technological progress that has been made?