The Input/Output Complexity of Sorting and Related Problems

Alok Aggarwal and Jeffrey Scott Vitter
Communications of the ACM, 1988

Large sorts are an important issue

Example: Banks

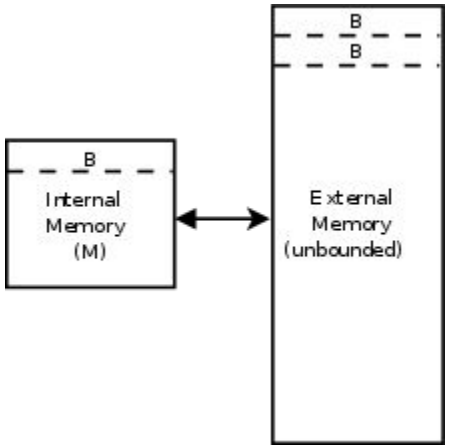
- Banks each night typically sort the checks of the current day into increasing order by account number
- Accounting files can be updated in a single linear pass through the sorted file
- Banks are required to complete this processing before opening for the next business day

Typical file sizes are expected to contain ten million records, totaling 10,000 megabytes, and current sorting methods would take most of one day to do the sorting (in 1988)

Today there are much bigger datasets that you might want to sort

Sorting bottleneck at I/O time

- Roughly one-fourth of all computer cycles perform sorting
- Resources are consumed by external sorts - file too large to fit in memory, must reside in secondary storage
- Takes time to transfer between memory and secondary storage



How to deal with large sorts? Two general approaches

Investigate alternate computer architectures such as parallel or distributed systems (done by Lindstrom and Vitter, 1985)

Examine the fundamental limits in terms of the number of I/OS for external sorting and related problems in current computing environments (current = 1988)

Parameters of the problem

N = # records to sort (*file size*)

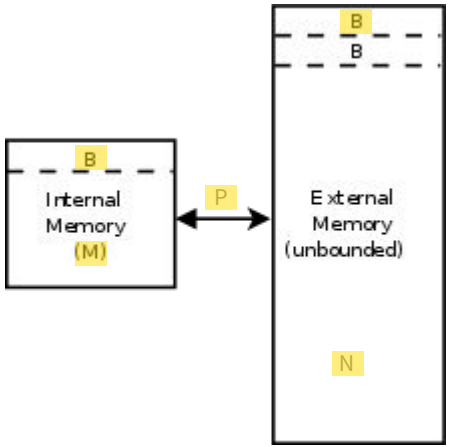
M = # records that can fit into internal memory (*memory size*)

B = # records that can be transferred in a single block (*block size*)

P = # blocks that can be transferred concurrently

$1 \leq B \leq M < N$

$1 \leq P \leq \text{floor}(M/B)$



I/O Transfers

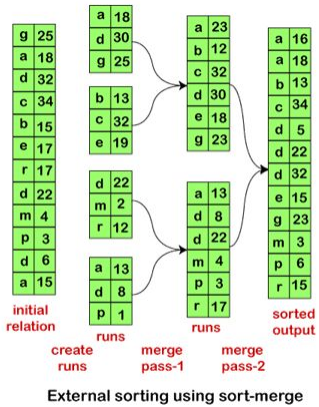
- Each block transfer is allowed to access any contiguous group of B records on the disk
- Parallelism:
  - Each block can transfer B info at once
  - P blocks can be transferred at a time (may include multiple I/O channels and read/write heads and an ability to access records in noncontiguous locations on disk in a single I/O)
- Extended memory - M locations in internal memory are x[1], x[2], . . . , x[M] and the locations on the disk are x[M + 1], x[M + 2], ...

Sorting and related problems

Sorting

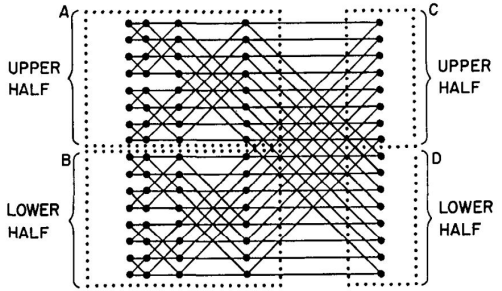
Begin: empty internal memory, N items stored in memory

End: empty internal memory, N sorted items stored in memory (sorted by key value)



External sorting using sort-merge

Sorting and related problems



FFT

Begin: empty internal memory, N items stored in memory (N is a power of 2)

End: N output nodes of the FFT directed graph (digraph) are “pebbled” and the memory configuration is exactly as it began

We can pebble node $n_{i,j}$ if its two predecessors have already been pebbled and if the records corresponding to those two predecessors both reside in internal memory.

Intuitively, the FFT problem can be phrased as the problem of pumping the records into and out of internal memory in a way that permits the computation implied by the FFT digraph.

Sorting and related problems

Permutation Network

Begin: empty internal memory, N items stored in memory

End: N output nodes of the permutation network digraph are “pebbled” and the memory configuration is exactly as it began

(Like FFT, pull the correct info into memory necessary for each step of the permutation graph)

Sorting and related problems

Permuting

Begin: empty internal memory, N items stored in memory

End: empty internal memory, N permuted items stored in memory (permuted by key value)

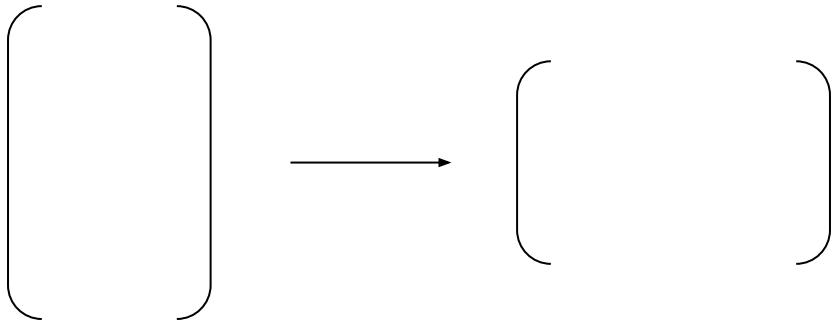
(Another version of sorting)

Sorting and related problems

Matrix Transposition

Begin: empty internal memory, pxq matrix stored in row major order on disk

End: empty internal memory, pxq matrix stored in column major order on disk



Theorems for I/O complexity

Permuting

Average / worst case number of I/Os required to permute N records is

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

Approach:

Bound the number of possible permutations that can be generated by *t* I/Os

If we take the value of *t* for which the bound reaches N! (all the permutations), we get a lower bound on the worst-case number of I/Os

Permuting

Average / worst case number of I/Os required to permute N records is

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

Approach: Automatically begin with 1 permutation

For each I/O:

- There can be at most N/B + t - 1 full tracks before the t^{th} output, and the records in the t^{th} output can go into one of at most N/B + t places relative to the full tracks
- Thus the t^{th} output changes the number of permutations generated by at most a multiplicative factor of N/B + t, which can be bounded trivially by N(1 + log N)

Permuting

Average / worst case number of I/Os required to permute N records is

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

Approach: Consider an input of B records from a specific track on disk

- If the B records were previously together in internal memory
  - B! possible orders of the B inputted records could already have been generated
  - Additional permutations at most a multiplicative factor of (M choose B), number of ways to intersperse B indistinguishable items within a group of size M
- If the B records were not output together previously
  - permutations generated is increased by an extra factor of B!, since the B records have not yet been permuted arbitrarily

Permuting

Average / worst case number of I/Os required to permute N records is

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

Approach: Consider an input of B records from a specific track on disk

- If the B records were previously together in internal memory

$$\left(\frac{N}{B} + t\right)B!\binom{M}{B} \leq N(1 + \log N)B!\binom{M}{B}$$

- If the B records were not output together previously

$$\left(\frac{N}{B} + t\right)\binom{M}{B} \leq N(1 + \log N)\binom{M}{B}.$$

Permuting

Average / worst case number of I/Os required to permute N records is

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

Approach: Determine the minimum value T such that the number of permutations generated is at least N!

$$(B!)^{N/B}\left(N(1 + \log N)\binom{M}{B}\right)^{T} \geq N!.$$

Permuting

Average / worst case number of I/Os required to permute N records is

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

Approach: Take logarithms, apply Stirling's formula, do some algebraic manipulation

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

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

Combine to get: $T = \Omega\left(\min\left\{N, \frac{N\log(N/B)}{B\log(M/B)}\right\}\right)$

Dividing by P gives final permutation bound

Permuting

Average / worst case number of I/Os required to permute N records is

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

The second term is the sorting bound

FFT / Permutation Networks

FFT $\Theta\left(\frac{N}{PB}\frac{\log(1+N/B)}{\log(1+M/B)}\right)$ **Permutation network** $\Omega\left(\frac{N}{PB}\frac{\log(1+N/B)}{\log(1+M/B)}\right)$

Approach:

Wu and Feng (1981) show that we can construct a permutation network by stacking together three FFT digraphs, so that the output nodes of one FFT are the input nodes for the next

Thus the FFT and permutation network problems are essentially equivalent

The lower bound for permutation networks matches the upper bound for FFT

FFT / Permutation Networks

FFT $\Theta\left(\frac{N}{PB}\frac{\log(1+N/B)}{\log(1+M/B)}\right)$ **Permutation network** $\Omega\left(\frac{N}{PB}\frac{\log(1+N/B)}{\log(1+M/B)}\right)$

Approach: Graph allows us to use optimal I/O strategy (unlike regular permutation)

Regardless of the permutation, the records that are transferred during an I/O and the track accessed during the I/O are fixed for each I/O

Eliminates the (N/B + t) term from permutation computation

$$\cancel{\left(\frac{N}{B}+t\right)}B!\binom{M}{B} \le N(1+\log N)B!\binom{M}{B}$$

$$\cancel{\left(\frac{N}{B}+t\right)}\binom{M}{B} \le N(1+\log N)\binom{M}{B}.$$

Then perform some more approximations etc.

Sorting

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

Permuting is a special case of sorting, so the lower bound for permuting also applies to sorting

Sorting

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

Note: this bound is the same as the second term in the normal permutation bound

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

Permutation always minimized by second term except when M and B $\ll$ N

Then permuting is as hard as more general problem of sorting

Dominant component of sorting in this case, in terms of the number of I/Os, is the routing of the records, not the determination of their order

Sorting

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

Note: this bound is the same as the second term in the normal permutation bound

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

When M and B $\ll$ N (so first term is smaller), the optimum algorithm for permuting is to move the records in the naive manner, one record per block transfer

This is precisely the case where advance knowledge of the output permutation makes the problem of permuting easier than sorting.

Matrix Transposition

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

When B is large, matrix transposition is as hard as general sorting

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

For smaller B, the special structure of the transposition permutation makes transposing easier

The three cases come from whether block size B is less than matrix dimensions, greater than one matrix dimension but not the other, or larger than both matrix dimensions

Optimal Algorithms

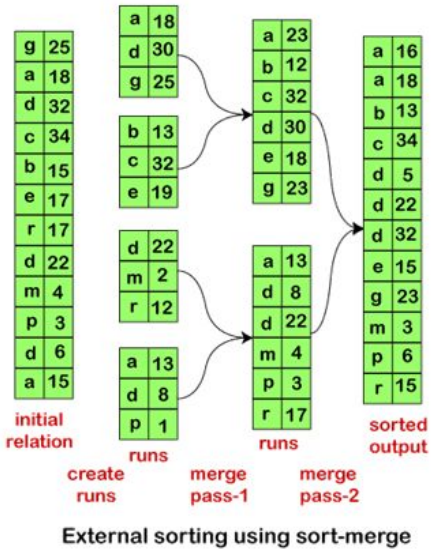
Merge Sort

- Run formation phase
  - Size M groups of blocks sorted
  - Output to consecutive positions
- Merge phase
  - M/B^{-1} runs are merged into one longer run

Resulting number of I/Os is

$$\frac{2N}{B}\left(1+\left\lceil \log_{M/B^{-1}}\frac{N}{M}\right\rceil\right)=O\left(\frac{N\ \log(N/B)}{B\ \log(M/B)}\right)$$

(not optimal)



Merge Sort

Use ‘endmarkers’ (key values of the last record in the each of the next P - 1 tracks of the run) to get more optimal behavior

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

Dividing by P’ = PB/B’ and with some algebraic manipulation, we get the desired upper bound

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

Distribution Sort

Find S approximate partitioning elements $b_1, b_2, \ldots, b_s$ that break up the file into roughly equal-sized “buckets”

Can compute the approximate partitioning elements using O(N/(PB)) I/Os

With O(M/(PB)) additional I/Os we can input M records from disk into internal memory and partition them into the S bucket ranges

Records in each ‘bucket’ range can be stored on disk in contiguous groups of B records each with a total of O(M/(PB) + S/P) = O(M/(PB)) I/Os

Procedure is repeated for another N/M - 1 stages, in order to partition all N records into buckets

Distribution Sort

The buckets are totally ordered with respect to one another

The remainder of the algorithm consists of recursively sorting the buckets one-by-one and appending the results to disk

I/Os needed to input the contents of the ith bucket into internal memory during the recursive sorting is bounded by $G_i/P = O(N_i/(PB))$

T(n) is the number of I/Os used to sort n records $$T(N) = \sum_{1 \le i \le S+1} T(N_i) + O\left(\frac{N}{PB}\right)$$

Using the facts that $N_i = O(N/S) = O(N/m)$ and $T(M) = O(M/(PB))$, we get the desired upper bound

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

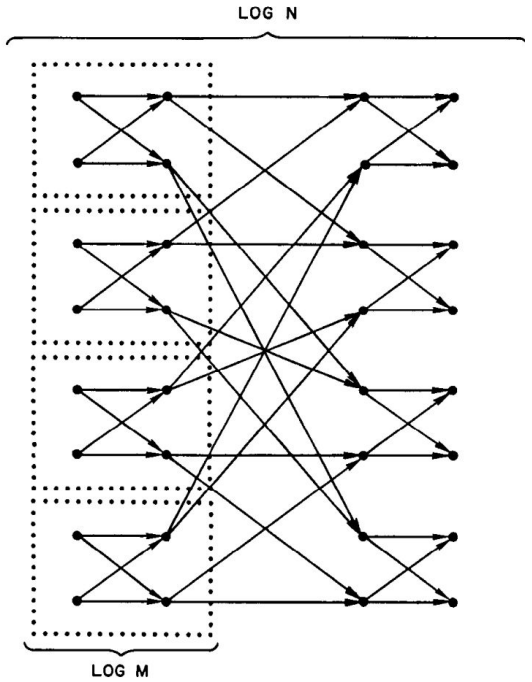
FFT and Permutation Network Algorithm

FFT digraph can be decomposed into (log N)/log M stages ->

Stage k corresponds to the pebbling of columns (k - 1)log M + 1, (k - 1)log M + 2, . . . , k log M in the FFT digraph

M nodes in column (k - 1)log M that share common ancestors in column k log M are processed together in a phase

Corresponding M records are brought into internal memory via a transposition permutation, and then the next log M columns can be pebbled



FFT and Permutation Network Algorithm

I/O requirement for each stage is thus due to the transpositions needed to rearrange the records into the proper groups of size M

Transpositions requires a total of $O((N/(PB))\log_{M/B}\min(M, N/M))$ I/Os

There are (log N)/log M stages, making the total number of I/Os

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

Can be shown by some algebraic manipulation to equal the upper bound

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

Matrix Transposition

Matrix transposition is a special case of permuting

Uses merging procedure

In each pass, which takes 2N/PB I/Os, the size of each target subgroup increases by a multiplicative factor of M/B

Total passes $\left\lceil \log_{M/B} \frac{B}{x} \right\rceil$.

Algebra and addition of conditions depending on block and matrix size yield

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

Conclusions

Summary

Derived matching upper and lower bounds, up to a constant factor, for the average-case and worst case number of I/Os needed to perform sorting-related tasks

Sorting, FFT, permutation networks, permuting, and matrix transposition

Related Work

Beigel and Gill: problem of determining how many applications of a black box capable of sorting k records are necessary to sort N records -> corresponds to sorting problem for the special case P = M = k and B = 1

Kwan and Baer: alternative disk model, in which P = 1 and the disk is decomposed into contiguous cylinders, each composed of several tracks -> considers locality of information accessed on disk (these authors think this is overly pessimistic)

Final open question

Can we remove our assumption that the records are indivisible and allow, for example, arbitrary bit manipulations and dissections of the records?

Lower bound should still hold in this more general model, since it is unlikely that these operations are of any great help, but no proof is known

Assessment of paper

Strengths

- Rigorous proofs
- Showed interesting connections between different sorting problems
- Found actual algorithms that perform at these bounds

Weaknesses

- Assumptions made such as full block transfer, indivisibility of keys

Further work

- Test experimentally
- Test in combination with parallelization techniques
- Special cases such as integer sorting (allowing arbitrary manipulations of the integers (hashing, XOR tricks, etc.)) (Farhadi et al 2021)
    - Integer sorting can be much faster than standard comparison sorting in RAM only computations
    - Possible to design external memory version that also outperforms comparison sorting