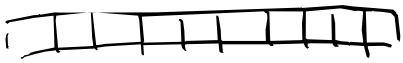
External memory algorithms

I/O model (external memory model).

main memory  of size M

external memory  , unbounded size,

partitioned into blocks of size B.

A block can be read or written at unit cost. (an I/O)

All computation can only be done in the main memory.

Goal: design algorithms with as few I/Os as possible.

$$\text{input size} \gg M \gg B \gg 1$$

Ex. scanning an array of N numbers takes $O(N/B)$ I/Os.

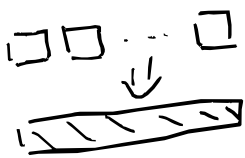
- think of input size $= n = N/B$ blocks. linear scan takes $O(n)$.

Sorting

Sort N number stored in $n = N/B$ blocks.

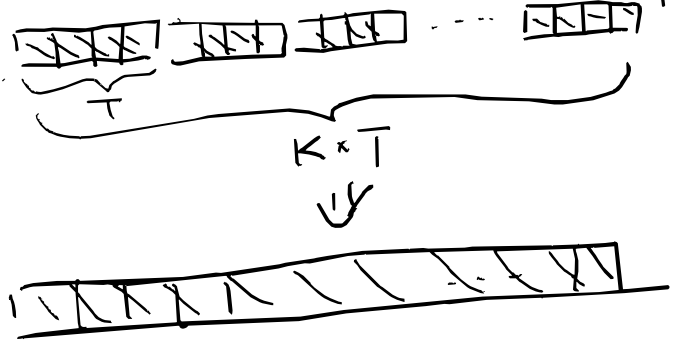
K-way merge sort ($K = M/4B$): $O(n \log_K n)$ I/Os.

Base case: read K blocks into memory, sort, write back



Merge K sorted arrays of T blocks, in $O(K \cdot T)$ I/Os. (*)
into a sorted array of KT blocks.

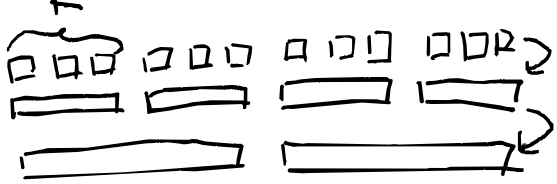
Key subroutine:



K-way merge:

- read the first block of each array into mem
- sort these $K \cdot B = M/4$ numbers, denote the sorted array by A
- initialize a block C (to empty) in memory to store the next block in the merged array
- repeatedly move the smallest number in A to C
  - if C has B numbers, write C to next output block in ext mem, empty C
  - if one of the K arrays has all its numbers moved to C from A, read its next block (if exists) add these numbers to A, maintain A in sorted order.

I/O cost



⋮

Iter i: (merge K sorted array of K^{i-1} blocks into K^i)
a total of $O(K^i \cdot \frac{n}{K^i}) = O(n)$ I/Os. $\times \frac{n}{K^i}$ times

A total of $\log_K n$ iterations

- $O(n \log_K n)$ I/Os.

External memory priority queues.

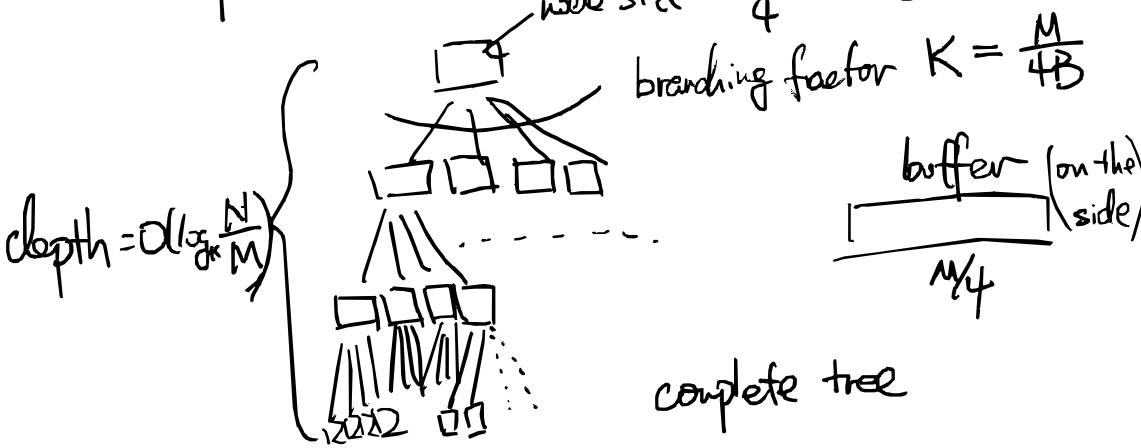
Maintain a set S:

- Ins(x): insert x
- ExtractMin(): delete and return min S

External mem heap : $O(\frac{1}{B} \cdot \log_{\frac{M}{B}} \frac{N}{M})$ amortized cost

$\ll 1$ when $B \gg \log N$

heap structure:



Each node stores between $\frac{1}{2} \cdot B$ and $K \cdot B$ numbers (except for the last leaf)
in sorted order!

It maintains the heap property

- the numbers stored in the parent node are all smaller than the numbers stored in the children nodes

The root node has the smallest $O(M)$ numbers

- the root node is stored in main mem
- all other nodes are stored in external mem

The heap is associated with a buffer of size $\frac{M}{4}$

- store "unprocessed" insertions,
- stored in mem

Ins(x):
  put x in the buffer.
  If the buffer is full:
    place the buffer in the last node
    restore the heap property bottom up

ExtractMin

- The min must be in the root node or in the buffer.
  find it and delete (in main mem)
- If the root node has $< \frac{K \cdot B}{2}$ numbers
  - "refill" by moving the smallest $\frac{K \cdot B}{2}$ numbers in its children up
  - recursively fix any children with $< \frac{K \cdot B}{2}$ numbers
  - for all leaves with $< \frac{K \cdot B}{2}$ numbers
    - merge the last leaf into it and restore heap property bottom up

I/O cost:

insert $O(M)$ numbers from buffer at once.

I/O cost per batch insertion:

- restoring the heap property between a parent & child
  - $O(K) = O(M/B)$ I/Os.
- $O(M/B)$ per node $\times$ $O(\log_{\frac{M}{B}} \frac{N}{M})$ levels
  $= O(\frac{M}{B} \cdot \log_{\frac{M}{B}} \frac{N}{M})$

amortized cost $O(\frac{1}{B} \cdot \log_{\frac{M}{B}} \frac{N}{M})$ I/O per ins

Extract Min:

$O(K)$ I/Os to refill a node

charge this cost to the $\Theta(K \cdot B)$ numbers moved up

- each number charged $O(\frac{1}{B})$ I/O

each number moves up $\leq O(\log_{\frac{M}{B}} \frac{N}{M})$ times.

$O(\frac{M}{B} \log_K \frac{N}{M})$ I/Os to merge the last leaf into a leaf

each last leaf is merged ≤ 2 times.

charge this cost to the numbers in the last leaf

cost $\leq O(\frac{1}{B} \cdot \log_{\frac{M}{B}} \frac{N}{M})$ per number.