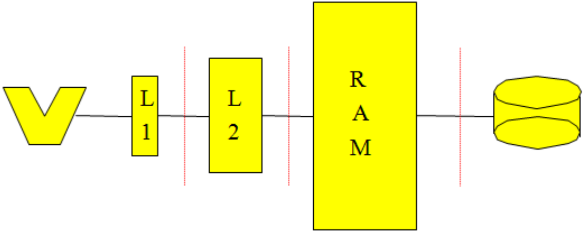
Massive Data Algorithmics

Lecture 2: Sorting

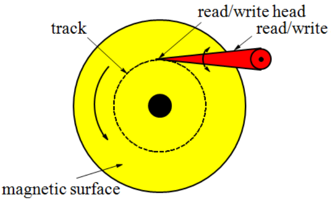
Hierarchical Memory



- Modern machines have complicated memory hierarchy
  - Levels get larger and slower further away from CPU
  - Data moved between levels using large blocks

Slow IO

- Disk access is 10^6 times slower than main memory access

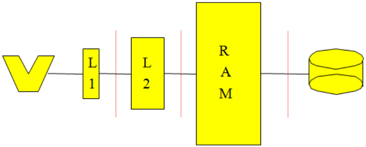


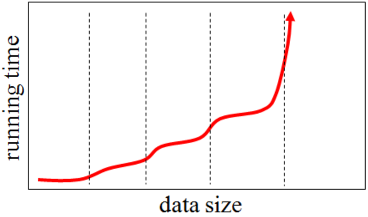
The difference in speed between modern CPU and disk technologies is analogous to the difference in speed in sharpening a pencil using a sharpener on ones desk or by taking an airplane to the other side of the world and using a sharpener on someone elses desk. (D. Comer)

- Disk systems try to amortize large access time transferring large contiguous blocks of data (8-16Kbytes)
- Important to store/access data to take advantage of blocks (locality)

Scalability Problems

- Most programs developed in RAM-model. Run on large datasets because OS moves blocks as needed
- Moderns OS utilizes sophisticated paging and prefetching strategies. But if program makes scattered accesses even good OS cannot take advantage of block access





External Memory Model(Cache-Aware Model)

$N = \#$ of items in the problem instance

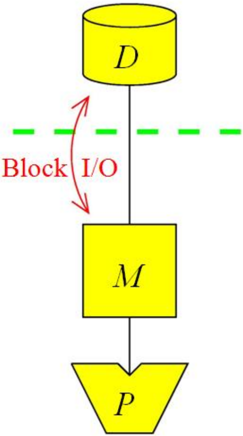
$B = \#$ of items per disk block

$M = \#$ of items that fit in main memory

$T = \#$ of items in output

I/O: Move block between memory and disk

We assume (for convenience) that $M > B^2$



Fundamental Bounds

	Internal	External
Scanning	N	N/B
Sorting	$N \log N$	$N/B \log_{M/B} N/B$
Permuting	N	$\min(N, N/B \log_{M/B} N/B)$
Searching	$\log N$	$\log_B N$

Note:

- Linear I/O: $O(N/B)$
- Permuting not linear
- Permuting and sorting bounds are equal in all practical cases
- B factor VERY important: $N/B < (N/B)\log_{M/B}(N/B) << N$

Scalability Problems: Block Access Matters

- Example: Reading an array from disk
  - Array size $N = 10$ elements
  - Disk block size $B = 2$ elements
  - Main memory size $M = 4$ elements (2 blocks)

1 5	2 6	3 8	9 4	7 10

Algorithm 1: N=10 I/Os

1 2	10 9	5 6	3 4	8 7

Algorithm 2: N/B=5 I/Os

- Difference between N and N/B large since block size is large
  - Example: $N = 256x106$, $B = 8000$, 1ms disk access time
    $\Rightarrow N$ I/Os take 256×103 sec $= 4266$ min $= 71$ hr
    $\Rightarrow N/B$ I/Os take $256/8$ sec $= 32$ sec

Queues and Stacks

- Queue
  -Maintain push and pop blocks in main memory

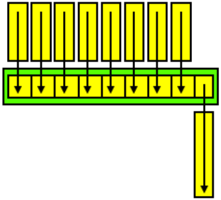
  $O(1/B)$ Push/Pop operations

- Stack
  -Maintain push/pop block in main memory

  $O(1/B)$ Push/Pop operations

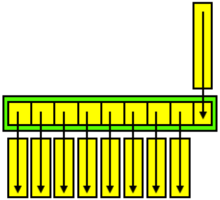
Sorting

- $< M/B$ sorted lists (queues) can be merged in $O(N/B)$ I/Os



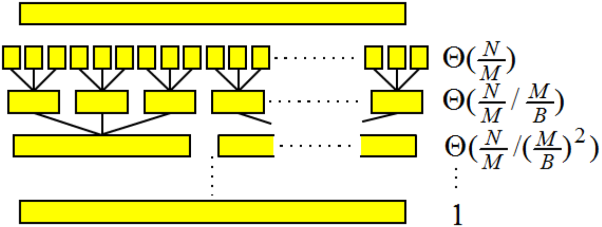
- M/B blocks in main memory

- Unsorted list (queue) can be distributed using $< M/B$ split elements in $O(N/B)$ I/Os



Merge Sort

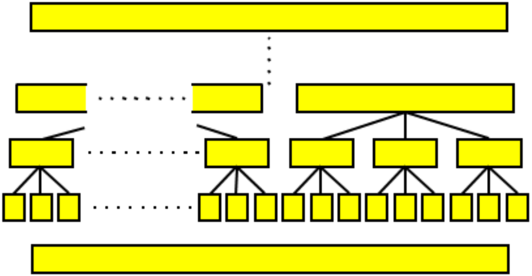
- Create N/M memory sized sorted lists
- Repeatedly merge lists together $T(M/B)$ at a time



- $\Rightarrow O(\log_{M/B} N/M)$ phases using $O(N/B)$ I/Os each $\Rightarrow$ $O(N/B \log_{M/B} N/B)$ I/Os

Distribution Sort (Multiway Quicksort)

- Compute $\Theta(M/B)$ splitting elements
- Distribute unsorted list into $\Theta(M/B)$ unsorted lists of equal size
- Recursively split lists until fit in memory



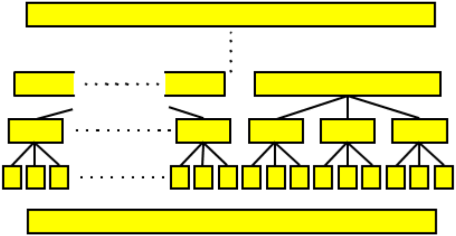
- $\Rightarrow O(\log_{M/B} N/M)$ phases $\Rightarrow O(N/B \log_{M/B} N/B)$ I/Os if splitting elements computed in $O(N/B)$ I/Os

Computing Splitting Elements

- In internal memory (deterministic) quicksort split element (median) found using linear time selection
- Selection algorithm: Finding ith element in sorted order
  1) Select median of every group of 5 elements
  2) Recursively select median of $\sim N/5$ selected elements
  3) Distribute elements into two lists using computed median
  4) Recursively select in one of two lists
- Analysis:
  - Step 1 and 3 performed in $O(N/B)$ I/Os.
  - Step 4 recursion on at most $\sim (7/10)N$ elements
  - $T(N) = O(N/B) + T(N/5) + T(7N/10) = O(N/B)$ I/Os

Distribution Sort (Multiway Quicksort)

- Distribution sort

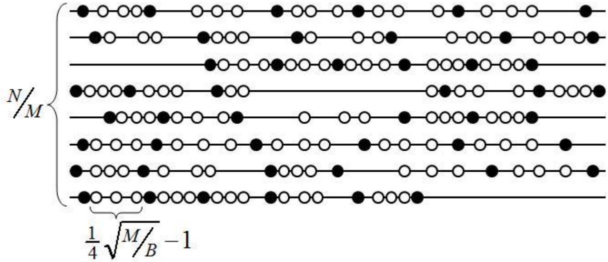


- Computing splitting elements:
  - $\Theta(M/B)$ times linear I/O selection $\Rightarrow O(NM/B^2)$ I/O algorithm
  - But can use selection algorithm to compute $\sqrt{M/B}$ splitting elements in $O(N/B)$ I/Os, partitioning into lists of size $< 3/2(N/\sqrt{M/B})$
  - $\Rightarrow O(\log_{\sqrt{M/B}} N/M) = O(\log_{M/B} N/M)$ phases $\Rightarrow O(N/B \log_{M/B} N/B)$

Computing Splitting Elements

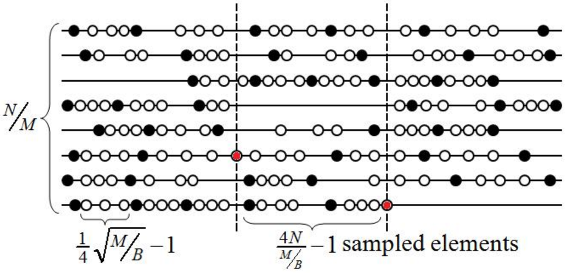
1) Sample $4N/\sqrt{M/B}$ elements:
   - Create N/M memory sized sorted lists
   - Pick every $1/4\sqrt{M/B}$ th element from each sorted list

2) Choose $\sqrt{M/B}$ split elements from sample:
   - Use selection algorithm $\sqrt{M/B}$ times to find every $4N/(M/B)$ th element



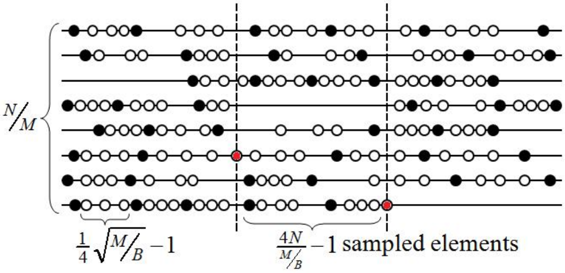
Computing Splitting Elements

1) Sample $4N/\sqrt{M/B}$ elements:
   - Create N/M memory sized sorted lists
   - Pick every $1/4\sqrt{M/B}$ th element from each sorted list
2) Choose $\sqrt{M/B}$ split elements from sample:
   - Use selection algorithm $\sqrt{M/B}$ times to find every $4N/(M/B)$ th element



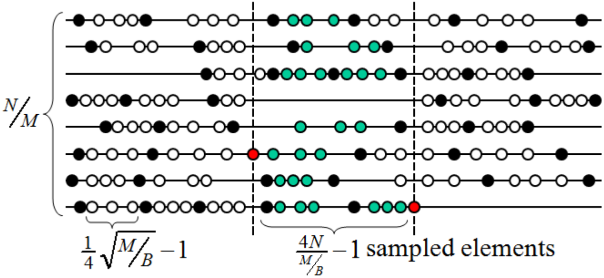
Computing Splitting Elements

1) Sample $4N/\sqrt{M/B}$ elements:
   - Create N/M memory sized sorted lists
   - Pick every $1/4\sqrt{M/B}$ th element from each sorted list
2) Choose $\sqrt{M/B}$ split elements from sample:
   - Use selection algorithm $\sqrt{M/B}$ times to find every $4N/(M/B)$ th element



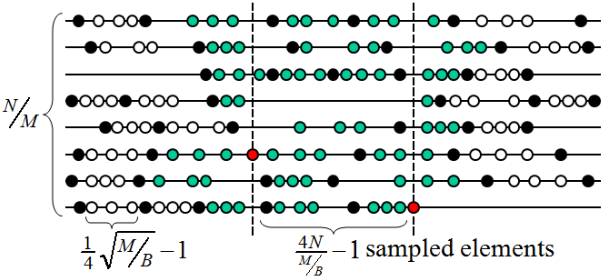
Computing Splitting Elements

- Elements in range R defined by consecutive split elements
  - Sampled elements in R: $4N/(M/B)-1$
  - Between sampled elements in R: $(4N/(M/B)-1)(1/4\sqrt{M/B}-1)$
  -
  -



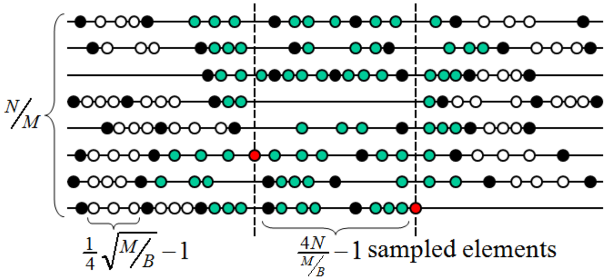
Computing Splitting Elements

- Elements in range R defined by consecutive split elements
  - Sampled elements in R: $4N/(M/B)-1$
  - Between sampled elements in R: $(4N/(M/B)-1)(1/4\sqrt{M/B}-1)$
  - Between sampled element in R and outside R: $2(N/M)(1/4\sqrt{M/B}-1)$
  -



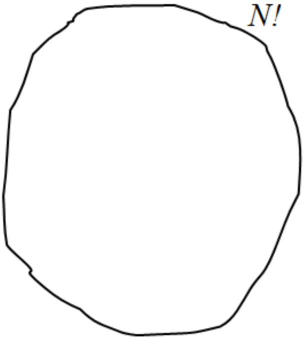
Computing Splitting Elements

- Elements in range R defined by consecutive split elements
  - Sampled elements in R: $4N/(M/B)-1$
  - Between sampled elements in R: $(4N/(M/B)-1)(1/4\sqrt{M/B}-1)$
  - Between sampled element in R and outside R: $2(N/M)(1/4\sqrt{M/B}-1)$
  - $4N/(M/B)+N/\sqrt{M/B}-4N/(M/B)+N/(2B\sqrt{M/B})<(3/2)N/\sqrt{M/B}$



Sorting lower bound

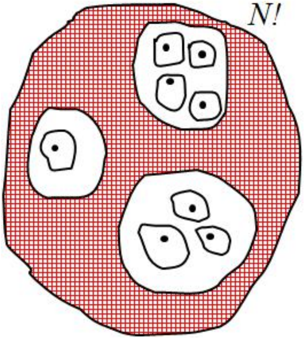
- Sorting N elements takes $\Omega(N/B \log_{M/B} N/B)$ I/Os in comparison model
- Proof:
  - Initially N elements stored in N/B first blocks on disk
  - Initially all N! possible orderings consistent with out knowledge
  - After t I/Os?



Sorting lower bound

- Consider one input assuming:
  - S consistent orderings before input
  - Compute total order of elements in memory
  - Adversary choose worst outcome of comparisons done

- possible orderings of $M-B$ old and B new elements in memory
- Adversary can choose outcome such that still consistent orderings
- Only get $B!$ term N/B times
- consistent orderings after t I/Os



References

- **Input/Output Complexity of Sorting and Related Problems**
  A. Aggarwal and J.S. Vitter. CACM 31(9), 1998
- **External partition element finding**
  Lecture notes by L. Arge and M. G. Lagoudakis.