# Sorting

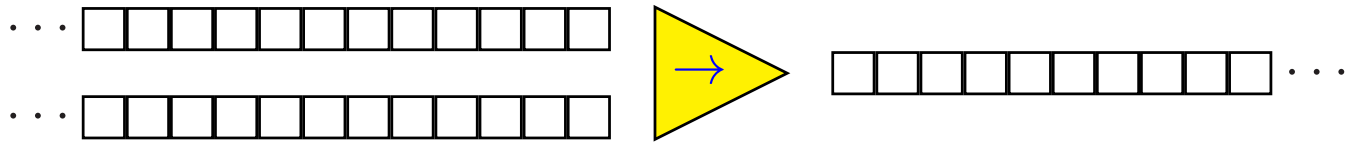
## Upper and Lower bounds

**[Aggarwal, Vitter, 88]**

# Part I: Upper Bound

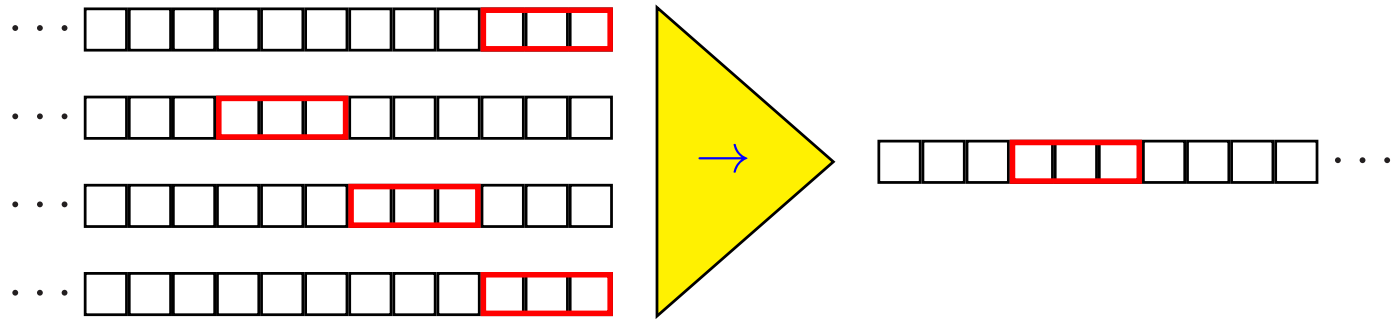
# Standard MergeSort

Merge of two sorted sequences $\sim$ sequential access



MergeSort: $O(N \log_2(N/M)/B)$ I/Os

# Multiway Merge



- For I/O-efficient $k$-way merge of sorted lists we need:

$$M \geq B(k+1) \Leftrightarrow M/B - 1 \geq k$$

- Number of I/Os: $2N/B$.

# Multiway MergeSort

- $N/M$ times sort M elements internally $\Rightarrow$ $N/M$ sorted *runs* of size $M$.
- Merge $k$ runs at a time, giving $(N/M)/k$ sorted runs of size $kM$.
- Merge $k$ runs at a time, giving $(N/M)/k^2$ sorted runs of size $k^2M$
- . . . repeat until only a single run remains.

At most $\log_k N/M$ phases, each using $2N/B$ I/Os. Largest $k$ is M/B-1.

$$O(N/B \log_{M/B}(N/M)) \text{ I/Os}$$

Note: we use $\log_a(b)$ as shorthand for $\max\{\log_a(b), 1\}$ (the above is not correct without this).

# Multiway MergeSort

Note that

$$1 + \log_{M/B}(x) = \log_{M/B}(M/B) + \log_{M/B}(x) = \log_{M/B}(x \cdot M/B)$$

Therefore

$$O(N/B \log_{M/B}(N/M)) = O(N/B \log_{M/B}(N/B))$$

Defining

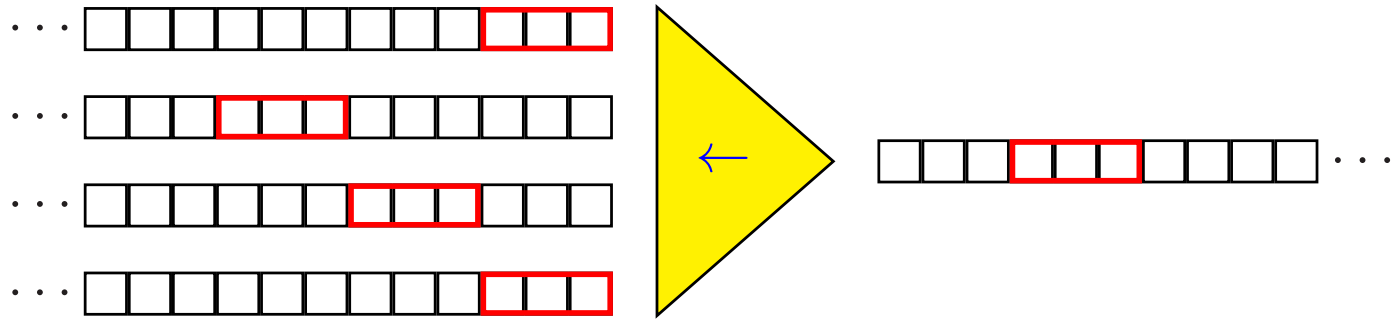
$\boxed{\text{n} = \text{N/B}}$ and $\boxed{\text{m} = \text{M/B}}$

we get

$$\boxed{\text{Multiway MergeSort:} \quad O(n \log_m(n))}$$

# Multiway QuickSort (DistributionSort)

Multiway splitting according to $k$ splitting elements:



- For I/O-efficient $k$-way distribution of sorted lists we need:

$$M \geq B(k+1) \Leftrightarrow M/B - 1 \geq k$$

- Number of I/Os: $2N/B$.
- We would also like to choose the $k$ elements elements such that $k$ is sufficiently large and the split is even (all subsequences are sufficiently reduced in size).

# Finding Partitioning Elements

Lemma: We can in O(N/B) I/Os choose $\sqrt{M/B}$ partitioning elements such that each subsequence is of size at most $N/\Theta(\sqrt{M/B})$.

For proof of lemma, see handout.

Since $\log_{\sqrt{y}}(x) = \log_2(x)/\log_2(y^{1/2}) = 2\log_y(x)$, it is easy to see that

$$\log_{\Theta(\sqrt{y})}(x) = \Theta(\log_y(x))$$

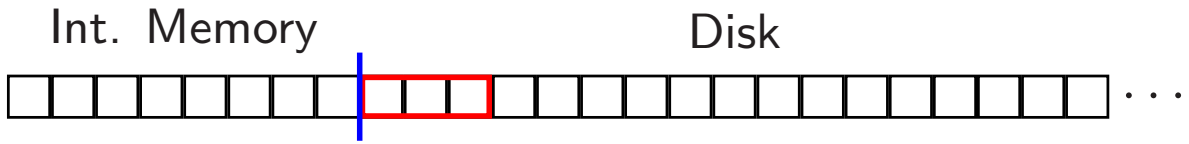
for all $y$ and $x$.

Hence, an analysis somewhat similar to that for multiway mergesort gives that an I/O-optimal sorting algorithm based on distribution is possible.

# Part II: Lower Bound

# The Model

View memory as single array of cells, each holding one element. First $M$ cells are the internal memory.




*Comparison-based* version of the I/O-model. The only allowed operations are:

- Comparison of elements in internal memory.
- Moving, copying, destroying elements in internal memory.
- Read/Write: transfer $B$ contiguous elements between disk and internal memory. Source cells are copied, target cells are overwritten.

Assume $M \geq 2B$. Wlog I/Os are assumed block-aligned (since a non-block-aligned I/O may be simulated using $\Theta(1)$ block-aligned I/Os).

# The Sorting Problem

- At start, input elements

  $$x_1, x_2, x_3, x_4, x_5, x_6, x_7, \ldots, x_N$$

  reside in the first $N$ cells outside internal memory.

- When algorithm stops, it should tell which of the $N!$ possible permutations

  $$x_7, x_2, x_{113}, x_N, x_{46}, x_1, \ldots, x_6$$

  of the input will make it sorted.

- We only consider inputs where all elements are different (enough for a lower bound). For these, exactly one permutation makes the input sorted.

# Adversaries

*Adversary*: An algorithm giving answers to comparisons performed by a sorting algorithm.

Answers must be consistent: there should always exist at least one permutation

$$x_7, x_2, x_{113}, x_N, x_{46}, x_1, \ldots, x_6$$

such that all answers given are true if this permutation makes the input sorted (ie., there should exist at least one possible input justifying the answers of the adversary).

Intuition of lower bound is that new comparisons can only be made by bringing new elements together in internal memory. This requires I/Os.

The goal of an adversary is to give as little new order information as possible for each new I/O. We need to quantify order information.

# Quantifying Order Information

Represent the answers of adversary by a directed graph $G = (V, E)$:

- $V = \{x_1, x_2, x_3, \ldots, x_N\}$
- $(x_i, x_j) \in E$ iff adversary was asked to compare $x_i$ and $x_j$, and answered $x_i < x_j$.

A permutation

$$x_7, x_2, x_{113}, x_N, x_{46}, x_1, \ldots, x_6$$

is called *compatible* with the graph if all edges go from left to right wben nodes are laid out linearly according to this permutation.

(In DM507 such a permutation of the nodes is called a topological sort of the graph, and it is proved that one exists iff the graph is acyclic)

The more compatible permutations remain, the less order information has been given by the adversary (ie., the more inputs are still possible).

# Order Information Dynamics

Given a sorting algorithm, an adversary algorithm, and a simultaneous run of the two, we let $G_t$ be the graph after $t$ I/Os have taken place, and let $S_t$ be the set of permutations compatible with $G_t$.

We have:

- Adversary must maintain $|S_t| \geq 1$ ($\Leftrightarrow G_t$ acyclic) for consistency.
- $|S_0| = N!$ (initial graph $G_0$ has no edges, so all permutations are compatible).
- $|S_t|$ is a decreasing function of $t$ ($G_t$ only gets more edges).
- A correct sorting algorithm cannot stop before $|S_t| = 1$ (if $|S_t| > 1$, adversary can still choose between several possible inputs, hence prove algorithms answer wrong).

# Adversary Definition

At each Read, the contents of internal memory changes, allowing new comparisons.

Adversary will settle answers to *all* new comparisons made possible, and add the corresponding edges to $G_t$. Hence, edges in $G_t$ always form a superset of those implied by the actual comparisons requested by the algorithm.

Adversary will settle these answers by deciding on *one total order of the elements currently in internal memory*, among all such orders compatible with previously settled answers (edges in $G_t$), ie., among all such orders that keep $G_t$ acyclic when adding all the edges implied by the order.

For the $t$th I/O, let $X_t$ denote the number of such orders. It remains to describe which of these possible orders the adversary chooses.

# Adversary Definition

Each choice of such order (of elements in internal memory) induces a different $G_t$, hence a different $S_t$ (recall, this is a subset of all permutations). For the family of possible $S_t$'s, the following holds:

- They are contained in $S_{t-1}$ (as edges only get added to graph).
- They cover all of $S_{t-1}$ (as any member (a permutation of all input elements) of $S_{t-1}$ determines a specific order of the elements currently in internal memory, and will be compatible with the $G_t$ induced by that choice of order (hence will be in that $S_t$)).
- None of them overlap each other (as any permutation of the input elements determines a specific order of the elements currently in internal memory, and can only be compatible with the $G_t$ induced by that choice of order (hence can only be in that $S_t$)) – any other order must have at least one of the added edges reversed.

# Adversary Definition

In other words: the family of possible $S_t$'s forms a *partition* of $S_{t-1}$.

In particular, their sizes sum to the size of $S_{t-1}$.

If we assume $|S_t| < |S_{t-1}|/X_t$ for all the possible $S_t$'s, we get a contradiction via

$$|S_{t-1}| = \text{sum of sizes} < X_t(|S_{t-1}|/X_t) = |S_{t-1}|$$

Hence, there exist at least one possible $S_t$ such that

$$|S_t| \geq |S_{t-1}|/X_t$$

The adversay after I/O number $t$ chooses the order of elements in internal memory giving that $S_t$.

# Upper Bounds on $X_t$

Any of the orders of the new contents of internal memorý can be constructed by first choosing $B$ locations among the $M$ possible ones (in the sorted order of the elements in internal memory), and then choosing a distribution into these locations of the $B$ elements of the block read.

This is because the order of the $M - B$ elements residing in internal memory before the I/O is already known (their order was settled by the adversay after the previous Read).

If the block read was previously written by the algorithm, the order of its $B$ elements has been settled earlier (as they were together in internal memory), and there is only one possible distribution of them over the $B$ chosen order-locations. If the block is untouched, there are $B!$ possible distributions of them (since we have block-aligned I/Os, a block is either completely untouched or completely touched).

# Upper Bounds on $X_t$

| Type of I/O | Read untouched block | Read touched block | Write |
|---|---|---|---|
| $X_t$ | $\binom{M}{B}B!$ | $\binom{M}{B}$ | 1 |

Note: at most $N/B$ I/0s on untouched blocks.

From $|S_0| = N!$ and $|S_t| \geq |S_{t-1}|/X_t$ we get

$$|S_t| \geq \frac{N!}{\binom{M}{B}^t (B!)^{N/B}}$$

Sorting algorithm cannot stop before $|S_t| = 1$. Thus,

$$1 \geq \frac{N!}{\binom{M}{B}^t (B!)^{N/B}}$$

for any correct algorithm making $t$ I/Os.

# Lower Bound Computation

$$1 \geq \frac{N!}{\binom{M}{B}^t (B!)^{N/B}}$$

$$t \log \binom{M}{B} + (N/B) \log(B!) \geq \log(N!)$$

$$3tB \log(M/B) + N \log B \geq N(\log N - \log e)$$

$$3t \geq \frac{N(\log N - \log e - \log B)}{B \log(M/B)}$$

$$t = \Omega(N/B \log_{M/B}(N/B))$$

Lemma was used:

a) $\log(x!) \geq x(\log x - \log e)$

b) $\log(x!) \leq x \log x$

c) $\log \binom{x}{y} \leq 3y \log(x/y)$ when $x \geq 2y$

# Proof of Lemma

Lemma:

a) $\log(x!) \geq x(\log x - \log e)$

b) $\log(x!) \leq x \log x$

c) $\log \binom{x}{y} \leq 3y \log(x/y)$ when $x \geq 2y$

Stirlings formula: $x! = \sqrt{2\pi x} \cdot (x/e)^x \cdot (1 + O(1/12x))$

Proof (using Stirling):

a) $\log(x!) = \log(\sqrt{2\pi x}) + x(\log x - \log e) + o(1)$

b) $\log(x!) \leq \log(x^x) = x \log x$

c) $\log \binom{x}{y} \leq \log(\frac{x^y}{(y/e)^y}) = y(\log(x/y) + \log(e))$

$\leq 3y \log(x/y)$ when $x \geq 2y$

# The I/O-Complexity of Sorting

Defining

$\mathsf{n} = \mathsf{N}/\mathsf{B}$

$\mathsf{m} = \mathsf{M}/\mathsf{B}$

$N/B \log_{M/B}(N/B) = \mathrm{sort}(N)$

we have proven

**I/O cost of sorting:**

$$\Theta(N/B \log_{M/B}(N/B)) = \Theta(n \log_m(n)) = \Theta(\mathrm{sort}(N))$$