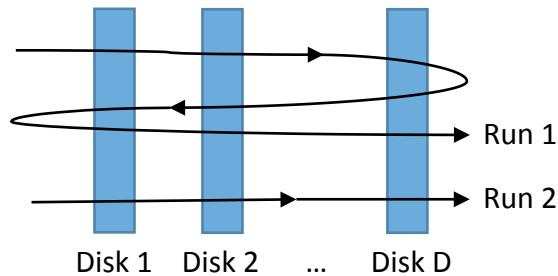


SORTING ON PARALLEL DISK MODELS (PDM)

DISK STRIPED MERGE SORT (DSM)



1 – Form runs of length M each.

2 – Merge the $\frac{N}{M}$ runs using R -way merge where $R = \frac{M}{DB}$. Note that since we normally assume that $M = \Theta(DB)$, R will be $O(1)$.

We'll keep BD keys from each run in memory. In the merge process whenever we run out of keys from any run, we can bring the next DB keys of this run from the disks. When BD keys are ready in the merged output, we write these keys in the disks. Note that the reads as well as writes are completely parallel and we do not waste any bandwidth.

$$\text{Total \# of I/O operations} = \frac{N}{DB} * \left[\frac{\log\left(\frac{N}{M}\right)}{\log\left(\frac{M}{DB}\right)} + 1 \right]. \text{ This can be much more than the optimal I/O.}$$

Example:

$$N = M^C$$

$$B = M^\epsilon$$

Where C and ϵ are constants. In this case, the # of passes in DSM:

$$\Omega(\log(M)).$$

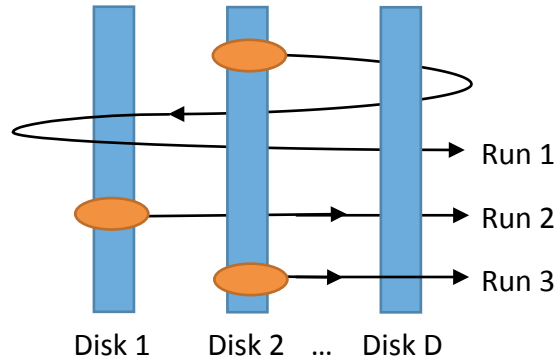
And

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

SIMPLE RANDOMIZED MERGE SORT (SRM)

(Barve, Vitter, 1999)

Random Starting Disk:



1 – Form runs of length M each. Stripe these runs starting from random disks.

2 – Merge the runs using R -way merge.

$$(R = \theta(D))$$

Barve & Vitter show that the expected performance is optimal if:

$$M = \Omega(BD * \log(D))$$

(l, m) – MERGE SORT (LMM)

(Rajasekaran, 1999)

ODD-EVEN MERGE SORT

$X = x_1, x_2, \dots, x_n \leftarrow \text{sorted}$

$Y = y_1, y_2, \dots, y_n \leftarrow \text{sorted}$

To merge X & Y :

1 – Partition X into X^{odd} and X^{even} :

$$X^{\text{odd}} = x_1, x_3, x_5, \dots$$

$$X^{\text{even}} = x_2, x_4, x_6, \dots$$

Similarly unshuffle Y into Y^{odd} and Y^{even} .

2 – Recursively merge X^{odd} and Y^{odd} to get Z_1 and recursively merge X^{even} and Y^{even} to get Z_2 .

3 – Shuffle Z_1 and Z_2 . Let:


$$Z_1 = a_1, a_2, \dots, a_n$$

$$Z_2 = b_1, b_2, \dots, b_n$$

The shuffle of Z_1 and Z_2 is the sequence:

$$Q = a_1, b_1, a_2, b_2, a_3, b_3, \dots, a_n, b_n$$

Perform one COMPARE-EXCHANGE operation on Q:

$$Q = a_1, b_1, a_2, b_2, a_3, b_3, \dots, a_n, b_n$$


ZERO-ONE LEMMA

If any comparison-based oblivious sorting algorithm sorts all possible sequences of zeroes and ones correctly then it also sorts any sequence of arbitrary elements.

Proof of correctness of the odd-even merge algorithm (using the zero-one lemma):

Let n_1 be the number of zeros in X .

Let n_2 be the number of zeroes in Y .

The number of zeroes in Z_1 is:

$$= \left\lfloor \frac{n_1}{2} \right\rfloor + \left\lfloor \frac{n_2}{2} \right\rfloor.$$

The number of zeroes in Z_2 is:

$$= \left\lfloor \frac{n_1}{2} \right\rfloor + \left\lfloor \frac{n_2}{2} \right\rfloor.$$

→ These two numbers differ by at most 2.

Case 1: Number of zeroes in Z_1 equals the number of zeroes in Z_2 .

$$Z_1 \rightarrow 000\ 000 \dots 0011 \dots 11$$

$$Z_2 \rightarrow 000\ 000 \dots 0011 \dots 11$$

Shuffle:

$$Q \rightarrow 000000000000 \dots 00001111 \dots 1111 \text{ (compare-exchange operation not needed - already sorted)}$$

Case 2: Number of zeroes in Z_1 equals the number of zeroes in $Z_2 + 1$.

$$Z_1 \rightarrow 00 \dots 0011 \dots 11$$

$$Z_2 \rightarrow 00 \dots 0111 \dots 11$$

Shuffle:

$$Q \rightarrow 0000 \dots 00011111 \dots 1111 \text{ (compare-exchange operation not needed - already sorted)}$$

Case 3: Number of zeroes in Z_1 equals the number of zeroes in $Z_2 + 2$.

$$Z_1 \rightarrow 00 \dots 00011 \dots 11$$

$$Z_2 \rightarrow 00 \dots 01111 \dots 11$$

Shuffle:

$Q \rightarrow 0000 \dots 001011111 \dots 1111$ (compare-exchange cleans the underlined 'dirty sequence')

$Q \rightarrow 0000 \dots 000111111 \dots 1111$

An extension of the odd-even merge algorithm has been proposed by (Thompson and Kung 1977). This algorithm was proposed to sort a mesh. If $X = k_1, k_2, \dots, k_n$ is a given input sequence, the idea is to partition the input into l equal sized parts (for some integer $l \geq 2$), sort each part recursively, and merge these sequences using the above 2-way merge algorithm. The (l, m) -merge sort (LMM) is an extension of the algorithm of Thompson and Kung.