CSE 5095 - Big Data Analytics - Notes from February 27$^{\text{th}}$

Recall that the (ℓ, m)-merge sort (LMM) is based on the (ℓ, m)-merge algorithm. If $X_1, X_2, \ldots, X_\ell$ are sorted sequences, then we can merge them using the (ℓ, m)-merge algorithm. The idea is to unshuffle each sorted sequence into m parts, recursively merge similar parts, shuffle the resultant sorted sequences, and finally perform some local sorting. Specifically, X_i is partitioned into $X_i^1, X_i^2, \ldots, X_i^m$, for $1 \le i \le \ell$. $X_1^j, X_2^j, \ldots, X_m^j$ are recursively merged to get Y_j, for $1 \le j \le m$. We then shuffle $Y_1, Y_2, \ldots, Y_m$ to get the sequence Z. As we have shown before, the length of the dirty sequence in Z is no more than ℓm. We perform some local sorting in Z to clean up the dirty sequence.

The first two steps of this algorithm are shown in Figure 1.

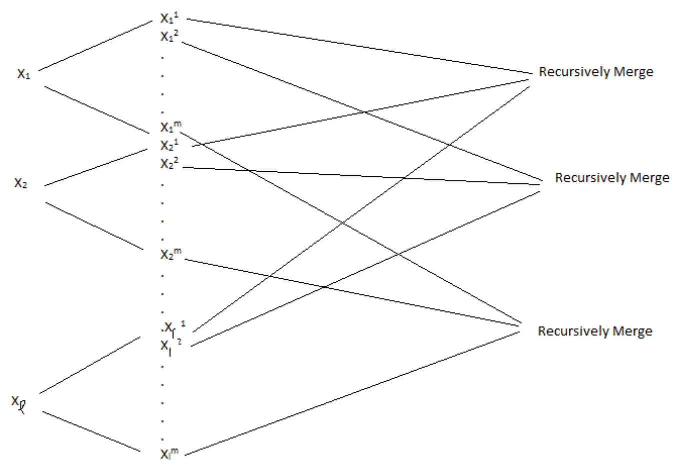

Figure 1: The first two steps of the (ℓ, m)-merge algorithm

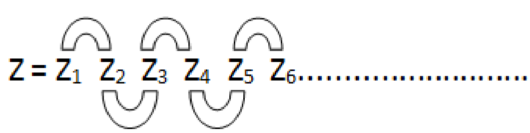

Figure 2: Cleaning up the dirty sequence

There are many ways to clean up the sequence Z. Partition the sequence Z into $Z_1, Z_2, \ldots$ where $|Z_i| = \ell m$ for any i. Call each of these Z_i's a block.
$=>$ the dirty sequence is within two successive Z_i's. Note that even though the length of the dirty sequence is no more than ℓm, we cannot say, for example, that the dirty sequence will be confined to a single block. One way of cleaning Z is to merge and sort Z_1 and Z_2; Z_3 and Z_4; etc. Followed by this we merge and sort Z_2 and Z_3; Z_4 and Z_5; etc.

Another way is to clean up Z is to sort and merge Z_1 and Z_2; Z_2 and Z_3; Z_3 and Z_4; etc. See Figure 2.

An Example. Consider the problem of sorting N keys where $N = M\sqrt{M}$, and $B = D = \sqrt{M}$.

We can sort these keys using LMM. The idea is to form runs of length M each in one pass through the data. Now we have to merge $\sqrt{M}$ sorted sequences of length M each. We can merge these using the (ℓ, m)-merge algorithm with $\ell = m = \sqrt{M}$. Let these runs be $X_1, X_2, \ldots, X_{\sqrt{M}}$. We first unshuffle each X_i into $\sqrt{M}$ parts, $1 \leq i \leq \sqrt{M}$. Specifically, X_i is unshuffled into $X_i^1, X_i^2, \ldots, X_i^{\sqrt{M}}$, for $1 \leq i \leq \sqrt{M}$. Note that the step of forming runs and unshuffling can be done together in one pass through the data. We then recursively merge $X_1^j, X_2^j, \ldots, X_{\sqrt{M}}^j$ to get Y_j, for $1 \leq j \leq \sqrt{M}$. This merging takes one pass through the data. Finally, we have to shuffle $Y_1, Y_2, \ldots, Y_{\sqrt{M}}$ to get Z and clean up the dirty sequence in Z. These two steps can be done together in one pass through the data if we have a core memory of size $2M$. The idea is to have Z_i and Z_{i+1} in the core memory at any point in time (for some value of i).

Thus, the total number of passes needed for this example is 3.

General Case:

Now we consider the general case of sorting N keys (for any value of N). There are two cases to consider.

Case 1:
$\frac{M}{B} \geq \sqrt{M}$

Case 2 :
$\frac{M}{B} < \sqrt{M}$

Let $T(i, j)$ stand for the number of passes needed to merge i sequences of length j each. To sort N keys, we will first form runs of length M each in one pass through the data. Followed by this, we will merge these $\frac{N}{M}$ sorted runs. Thus a central question is:

$$T\left(\frac{N}{M}, M\right) = ?$$

Exercise :

1. Show that $T(\sqrt{M}, M) = 3$ when $\frac{M}{B} \geq \sqrt{M}$. *Hint:* Use $\ell = m = \sqrt{M}$.
2. Show that $T(\frac{M}{B}, M) = 3$ if $fracMB < \sqrt{M}$. *Hint:* Use $\ell = m = \frac{M}{B}$.

The generic sorting algorithm will be described in two cases.

Case 1: $\frac{M}{B} \geq \sqrt{M}$.

We use (ℓ, m) merge algorithm with $\ell = m = \sqrt{M}$. Let $K = \sqrt{M}$ and let $\frac{N}{M} = K^{2c} => \frac{N}{M} = M^c => c \log M = \log(\frac{N}{M}) => c = \frac{\log(\frac{N}{M})}{\log(M)}$

What is the value of $T(K^{2c}, M)$?

The claim is $T(K^{2c}, M) = T(K, M) + T(K, KM) + T(K, K^2M) + \cdots + T(K, K^{2c-1}M)$. This claim follows from the fact that we can merge K^{2c} sequences of length M each using a K-way merge strategy as shown in Figure 3 and Figure 4.

Now consider the problem of merging K sequences of length K^iM each, for any i. This merging can be done using the (ℓ, m)-merge algorithm with $\ell = m = K$. Unshuffling will take one pass. Recursive mergings will take

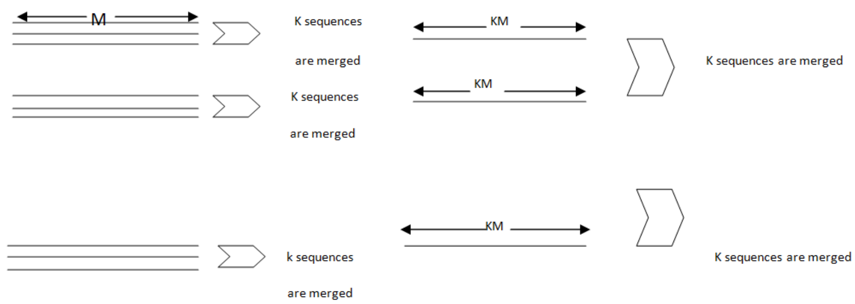

Figure 3: K-way merge

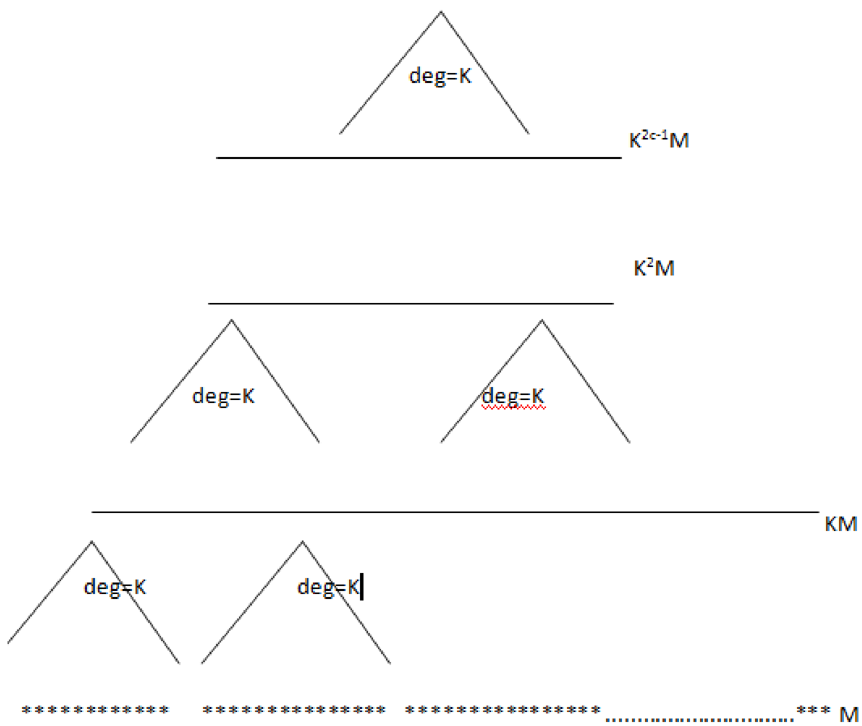

Figure 4: K-way merge tree

$T(K, K^{i-1}M)$ passes. Shuffling and cleaning the dirty sequence can be done in one more pass. (See Figure 5). Thus it follows that $T(K, K^iM) = T(K, K^{i-1}M)+2$. This means that $T(K, K^iM) = 2i+T(K, M) = 2i+3$.

As a result, it follows that $T(K^{2c}, M) = \sum_{i=0}^{2c-1}(2i + 3) = 4c^2 + 4c$.

<u>Case 2:</u> $\frac{M}{B} < \sqrt{M}$.
We use (ℓ, m)-merge algorithm with $\ell = m = \frac{M}{B}$. Let $Q = \frac{M}{B}$. Let $Q^d = \frac{N}{M} => d = \frac{\log(\frac{N}{M})}{\log(\frac{M}{B})}$.

Along the same lines as in case 1, we see that $T(Q^d, M) = T(Q, M) + T(Q, QM) + \cdots + T(Q, Q^{d-1}M)$.

We can also see that $T(Q, Q^iM) = 2 + T(Q, Q^{i-1}M) = 2i + T(Q, M) = 2i + 3$

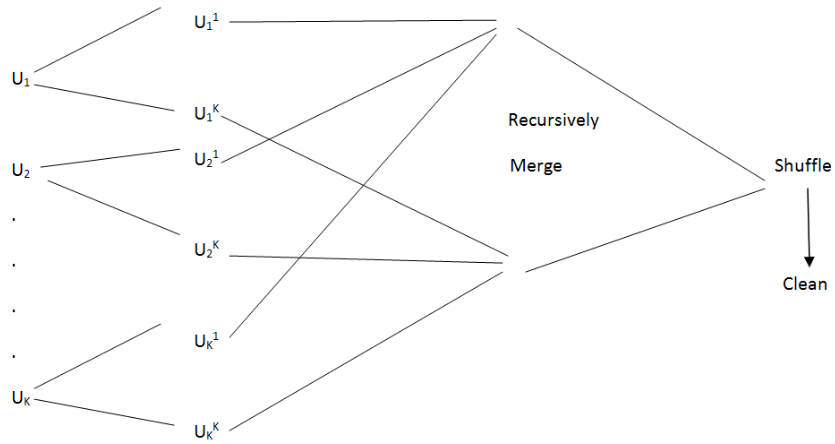

Figure 5: (K, K)-merge algorithm

$=> T(Q^d, M) = \sum_{i=0}^{d-1}(2i+3) = d^2 + 2d.$

Putting cases 1 and 2 together we get the following

Theorem :

We can sort N elements using D disks in no more than $\left[\frac{\log(\frac{N}{M})}{\log(\min\{\sqrt{M}, \frac{M}{B}\})} + 1\right]^2$ number of passes through the data.

Example:

If $N = M^4$ and $B = M^{\frac{2}{3}}$, the number of passes taken by the above algorithm is $(\frac{3 \log M}{\frac{1}{3}\log M} + 1)^2 = 100$.

Rajasekaran and Sen(2004) have presented an asymptotically optimal Las Vegas algorithm to sort N given keys. Several other authors have given such an optimal algorithm as well. The algorithm of Rajasekaran and Sen is much simpler than that of the others.

IDEA :

Apply a random permutation to the input. Form runs of length M each. Apply an $R-$way merge with $R = \Theta(\frac{M}{B})$. A random permutation ensures that the leading blocks of the runs that are merged at any given time, or nearly in distinct disks. See Figure 6.

Random Permutations:

Let $X = k_1, k_2, \ldots, k_N$. To permute X, each key is assigned a random label and then the keys are sorted with respect to their labels.

Fact : We can sort N integers in the range $[1, R]$ in $O(\frac{\log(\frac{N}{M})}{\log(\frac{M}{B})})$ passes through the data if the value of each key is uniformly distributed in the range $[1, R]$, where R is any integer.

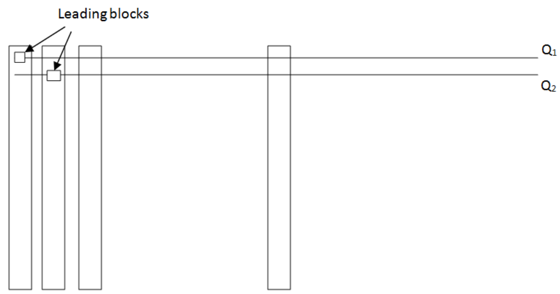

Figure 6: Sorting after random permutation

Algorithm: Form runs of length M each. Merge them using $R-$way merge with $R = \frac{M}{B}$. Assume that we have CM memory where C is a constant (greater than one). Whenever BD keys are ready in the merged sequence, write them in the disks. When B keys have been consumed from any run, do a parallel I/O.

Analysis:At any time we have nearly C blocks of each run in memory. Each key that goes into the output is equally likely to have come out of any of the runs. When BD keys are output to the disks, the expected number of these keys that have come out of any run Q_i is B (for any i).

=> Using Chernoff bounds, this number is $\in [(1 \pm \epsilon)B]$ with high probability (ϵ being a constant fraction).

=> the number of passes needed is $\tilde{O}\left(\frac{\log(\frac{N}{M})}{\log(\frac{M}{B})}\right)$.

To perform a random permutation:

Assign a random label to each input key in the range $[1, N^{1+\beta}]$ for some constant $\beta < 1$. Sort the sequence based on the labels. Scan through the sorted sequence to permute equal keys.