CSE 4502/5717: Big Data Analytics

Notes by Anthony Hershberger

Lecture 4 - January, 31st, 2018

1 Problem of Sorting

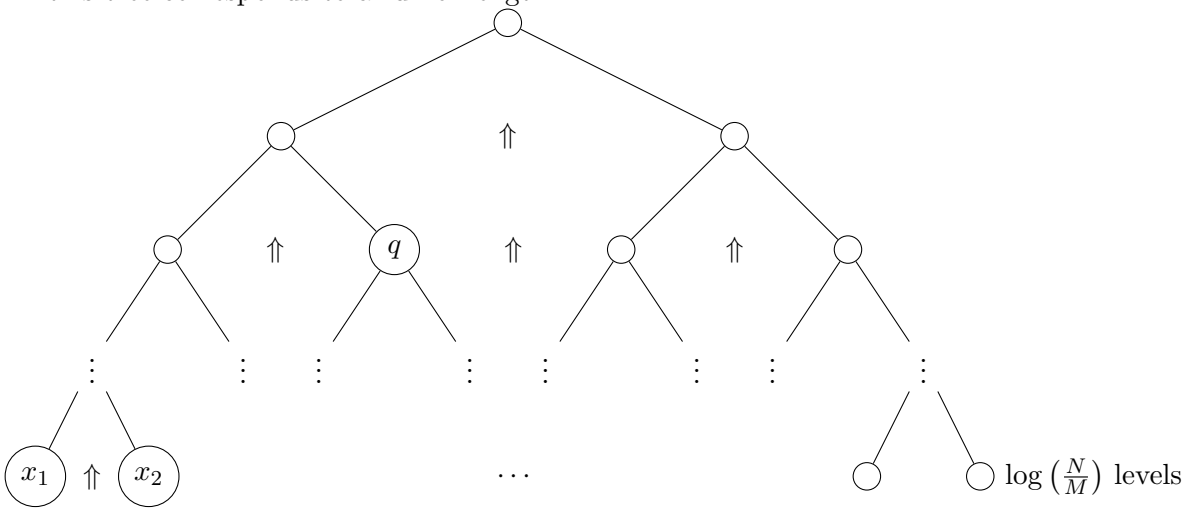
A known lower bound for sorting is $\Omega\left(\frac{N}{B} \cdot \frac{log(\frac{N}{M})}{log(\frac{M}{B})}\right)$ I/O operations, where

N is the input size;
M is the core memory size; and
B is the block size.

Merge Sort

Input Given N elements

1. In one pass through the data form Runs of Length M each. Followed by this, merge two runs at a time as shown in the following figure. Each leaf in this tree corresponds to a run of length M.



Process: Consider an arbitrary node (i.e., a non-leaf) in this tree. Two runs are merged at this node. We now show how to merge these two runs A and C. Let $|A| = |C| = \ell$. To begin with, bring $\frac{M}{2B}$ blocks from each run. Start merging these runs. We keep an output buffer of size B. When one block is ready in the output (i.e., when the output buffer is full), ship it to the disk. When we run out of keys from either A or C, bring one block of that run from the disk. Repeat the above steps until all the keys from one of the runs have been used. At this time, output the remaining keys from the other run.

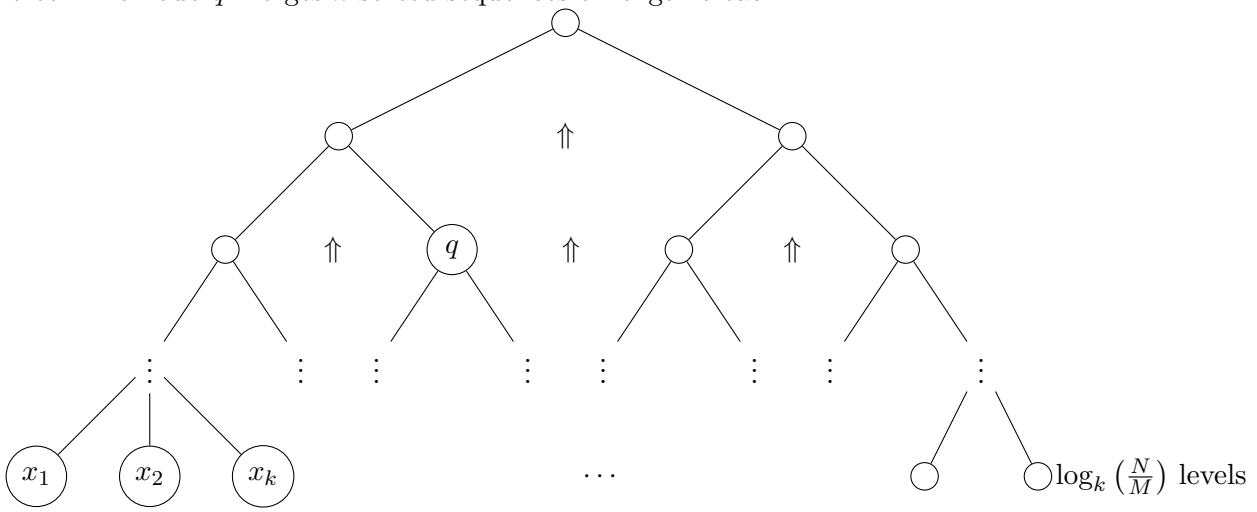
Number of I/O operations needed to merge A and C
$= \frac{\text{Total number of elements in both } A \text{ and } C}{B} = \frac{2\ell}{B}$. This implies that the
Total Number of I/O's needed at any level of the tree $= \frac{N}{B}$.

Note: We spend $\frac{N}{B}$ I/O operations per level in the tree and the number of levels in the tree is given by $\log\left(\frac{N}{M}\right)$. This implies that the total # of I/O's taken by the entire algorithm is $\frac{N}{B}\left[\log\frac{N}{M} + 1\right]$. This is not optimal but can we make it optimal?

Modification

We will use a modification where we merge k sorted sequences at a time (for some $k > 2$).

1. First we form runs of length M each where we let q be any node in the tree. The node q merges k sorted sequences of length ℓ each.



Process: We explain how any node q in the tree merges k runs. Let the length of each run be ℓ. To begin with, bring $\frac{M}{kB}$ blocks from each run; When B elements are ready in the output, write this block to the disk; When we run out of keys in any run, read the next block from this run. When all the keys from any run have been used, this run will not play a role after that.

Repeat the above steps untill all the runs have been merged.

$$\textbf{Total Number of I/O's at any level of the tree} = \frac{N}{B}.$$
$$\textbf{Number of levels} = \log_k \frac{N}{M} = \frac{\log(\frac{N}{M})}{\log(k)}$$

Note: We will always use a base 2 unless explicitly stated.
If we fix the value of $k = \frac{M}{B}$, then

$$\textbf{Total Number of I/O Operations} = \frac{n}{B} \cdot \left(\frac{\log(\frac{N}{M})}{\log(\frac{M}{B})} \right) + 1$$

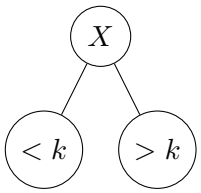
Question: Is this asymptotically optimal? Yes.

2 Selection Algorithm

Input: $X = k_1, k_2, ..., k_n$
Output: The i^{th} smallest element of X

Quick Select(X, i)

1. Pick a pivot $k \in X$
2. Partition X into two:



Without loss of generality, assume that the elements of X are distinct. Let $X_1 = \{q \in X : q < k\}$ and $X_2 = \{q \in X : q > k\}$.

Case 1: If $|X_1| = i - 1$ then output k and quit;
Case 2: If $|X_1| \geq i$ then output QuickSelect(X_1, i);
Case 3: If $|X_1| + 1 < i$ then output QuickSelect$(X_2, i - |X_1| - 1)$;

QuickSelect has a worst case run time of $\Omega(n^2)$. We can show that the average run time of QuickSelect is $O(n)$. Blum, Floyd, Pratt, Rivest and

Tarjan have modified this algorithm to obtain an algorithm that takes $O(n)$ time in the worst case.

Blum, Floyd, Pratt, Rivest, Tarjan, 1973

1. Pick a Pivot as follows:
   (a) Group the input into groups as size 5 each; let the groups be $G_1, G_2, G_3, \ldots, G_{n/5}$;
   (b) Find the median of each group. This can be done using $\leq \frac{9n}{5}$ comparisons;
   Let the group medians be denoted as $M_1, M_2, M_3, \ldots, M_{n/5}$;
   (c) Find the median M of these medians RECURSIVELY;
2. Perform steps 2 and 3 of QuickSelect with M as the pivot;

Question: Why does the algorithm work? The median of the medians can be thought of as an approximate median of X. As a result, we can show that the sizes of X_1 and X_2 will be nearly the same.

Analysis of the BFPRT algorithm

Claim: $|X_1| \leq \frac{7}{10}n$ and $|X_2| \leq \frac{7}{10}n$.

Proof:

Note: Consider all the groups whose medians are $\leq M$. There are $\frac{n}{10}$ such groups. In such group there will be at least three elements that are $\leq M$. This means that the size of X_2 cannot be more than $\frac{7}{10}n$. In a similar manner we can also show that the size of X_1 cannot exceed $\frac{7}{10}n$.

Step 2 and Step 3 of QuickSelect take n and $\leq T(\frac{7}{10}n)$ comparisons, respectively.

Run Time of the BFPRT algorithm:

Let $T(n)$ be the runtime of this algorithm on any input of size n and for any i. The time it takes for step 1b is $\frac{9n}{5}$. Step 1c takes $T(n/5)$ time. Step 2 and Step 3 of QuickSelect take n and $\leq T(\frac{7}{10}n)$ comparisons, respectively.

$$\textit{Thus we get: } T(n) = \tfrac{9}{5}n + n + T(\tfrac{n}{5}) + T(\tfrac{7}{10}n)$$

Claim: $T(n) \leq Cn$ for some constant C. We'll prove this by induction.

Assume the hypothesis for all inputs of size $\leq (n-1)$; We will prove it for n.

$$T(n) \leq T(\tfrac{n}{5}) + T(\tfrac{7}{10}n) + dn, \text{ where } d \text{ is come constant.}$$

Applying the induction hypothesis, we get:

$$\begin{gathered} T(n) \leq C\tfrac{n}{5} + C\tfrac{7}{10}n + dn \\ \text{RHS} = .9Cn + dn \\ \text{RHS} \leq Cn \text{ if } 0.9Cn + dn \leq Cn \\ \implies .1C \geq d \implies C \geq 10d \\ \implies T(n) \leq 10dn. \text{ QED} \end{gathered}$$