Lecture Notes: RAM Model, EM Model and Sorting

Miao Qiao
Computer Science Department
University of Auckland
miao.qiao@auckland.ac.nz

1 RAM Model and Sorting

Among the algorithms that you my have learnt in undergraduate level courses, sorting algorithms should have been extensively studied. A sorting problem aims at transforming the input, an array of elements, to the output, an array of the same set of elements in ascending/descending order based on a *comparison function*. The comparison on two elements returns an outcome from $\{=, <, >\}$.

To compare different algorithms for the same computation problem, e.g., the sorting problem, in a machine-independent manner, we use *computation models*. A computation model specifies a set of operations that can be completed in a unit cost (*unit-cost operations*). An algorithm under the computation model can call unit-cost operations and their time complexity is measured in the total number of operations called by the algorithm.

Random access machine model (RAM model). RAM model specifies a set of unit-cost operations: $+, *, -, =$, if, call, and memory access.

Remarks. We leave as an exercise for you to recall the sorting algorithms under RAM model and then analyze their complexities.

1.1 Lower Bound

For a computation problem, an algorithm brings about an upper bound. For example, the bubble sort provides an $O(n^2)$ upper bound for the sorting problem on n elements in RAM model while merge sort provides a better upper bound of $O(n \log_2 n)$. The question that naturally follows is: "What is the best that we can do?" Such a reflection on a computation problem considers an unlimited number of algorithms, as opposed to only one algorithm. To derive a conclusion, sometimes we have to introduce stronger restrictions on the operations that can be used in a computation model.

Comparison Model. To study the lower bound of a sorting problem in RAM model, we invented the *comparison model*. An algorithm under comparison model — a comparison-based algorithm — is allowed to perform only two types of operations:

1. Memory access, and
2. Comparison-function call.

Next, we show that for any comparison-based algorithm on the sorting problem, the complexity is at least $\Omega(n \log_2 n)$ in the worst case.

Let $e_1, e_2, \cdots, e_n$ be the n distinct elements. A comparison-based sorting algorithm $\mathcal{A}$ is essentially a binary decision tree $T(n)$. Note that $T(n)$ is constructed before $\mathcal{A}$ knows the values of the elements. The algorithm starts from the root of $T(n)$. Each internal node u represents a comparison of two elements with specified indexes, e.g., e_i and e_j. The outcome of the comparison($>$ or $<$) instructs the algorithm to go towards either the left child or the right child. Each leaf node on $T(n)$ means that $\mathcal{A}$ is able to determine the output — a permutation of the n elements.

Consider all possible instantiations of the n elements, the output can be an arbitrary permutation of the n elements. Therefore, the decision tree $T(n)$ of a comparison-based sorting algorithm has at least $N!$

leaves. Note that the algorithm walked along a root-to-leaf path, the time complexity is bounded by the height of the tree. The worst-case time complexity of the algorithm is thus at least $O(\log(n!)) = O(n \log_2 n)$ (Stirling's formula).

2 EM Model and Sorting

The RAM model that we are familiar with greatly simplifies the complexity analysis — it costs $O(1)$ time no matter the data access is from/to the main memory, secondary storage, or another machine's storage. However, such a simplification fails in capturing the actual cost — accessing a byte from the secondary storage is normally 10^3 times slower than that from main memory. Such a distinction cannot be neglected especially when the data size exceeds the main memory capacity. This motivated the invention of external memory (EM) model [1].

2.1 EM Model.

A computer in EM model is equipped with a memory of M words. The data on the secondary storage, the disk, is formatted into blocks while each block has B words. One input/output (I/O) operation transfers a block of B words from the main memory to the disk or from the disk to the main memory. An algorithm under EM model evaluates its time complexity in the total number of I/Os performed and space complexity in the total number of disk blocks occupied. To allow an input buffer and an output buffer in memory, we assume $M \geq 2B$.

Why EM model can successfully capture the characteristics of I/O-bound algorithms? Apart from the fact that the I/O cost can dominate CPU cost, the model is also praised for its effective simplification of complicated physical storage.

Figure 1 shows a magnetic disk. A read-write head is close to a platter surface, a platter surface consists of circular tracks and a circular track consists of sectors.

The average access time (seeking time), the time between a read/write request and the transfer of the first byte of data, is 8-21 milliseconds (10^{-3} seconds). After the first byte is found, the data is transfered in data-transfer rate, the rate at which data can be retrieved from/stored to sequential sectors in a track, of up to 100 MB per second. The way to reconcile the two parameters is to enforce each random access to read/write at least a certain amount of sequentially stored data — a block of data. A typically setting of block size B in practice is $1024 - 4096$ bytes.

2.2 EM Merge Sort

Let S be N elements in $\mathbb{R}$ initially stored in $O(N/B)$ blocks on disk. The aim for sorting is to output a file of $O(N/B)$ blocks where the elements have been sorted. Naively adapting a sorting algorithm in RAM model leads to $O(N \log_2 N)$ I/Os. Now, we introduce an external memory sorting algorithm of $O(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$ I/Os. Let $m = \frac{M}{B}$ and $n = \frac{N}{B}$.

The algorithm initially creates $O(n/m)$ runs, each run includes $O(m)$ blocks of elements such that they can be sorted in a memoryload. After the initial phase, each run is stored in a file of $O(m)$ blocks on disk with all elements sorted. The initial phase completes in $O(n)$ I/Os.

The following computation includes $O(\log_m \frac{n}{m})$ rounds. In the first round, every $O(m)$ runs from the initial phase are merged into a new sorted run in linear I/Os, which forms $O(\frac{n}{m^2})$ new runs in this round. In the following rounds we do the similar thing until the total number of runs is reduced to 1. Note that, each round reduces the total number of runs by a factor of $\Omega(m)$, therefore, the total number of rounds is $O(\log_m \frac{n}{m})$. Each round costs $O(n)$ I/Os.

Therefore, the total I/O complexity of the sorting algorithms is $O(n(\log_m \frac{n}{m} + 1)) = O(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$.

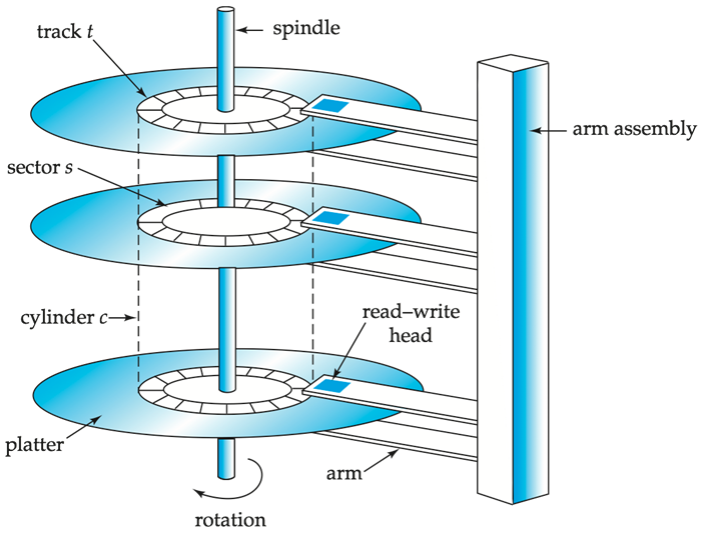

Figure 1: Magnetic Disk

Remarks. EM merge sort has been adopted by Linux and Unix as the materialization of their "sort" command.

2.3 EM Distribution Sort

EM merge sort is a bottom-up algorithm: it starts with creating runs of size $O(M)$ and then iteratively merges the runs until a single run of size N is obtained. The distribution sort [2] that we shall see in this section adopts a top-down manner. Given a set S of N elements, distribution sort first partition S into $f = \sqrt{\frac{M}{B}}$ roughly equal-sized subsets $S_1, S_2, \cdots, S_f$ such that any elements in S_i is no greater than elements in S_j if $1 \leq i < j \leq f$. After that, each subset is then sorted recursively. The key to distribution sort is to determine the splitters between the neighboring subsets. To warm up, we first look at a sub-problem called *k-selection*.

2.3.1 k-Selection

Given a set S of N *distinct* elements, the aim of *k-selection* is to report the k-th smallest element in S. Obviously, the k-th smallest element can be trivially found in a sorting time of S. Next, we describe a smart algorithm that can find the k-th smallest element of S for any k in linear time.

Let c be a positive odd constant integer. The algorithm $ksel(S, k)$ has the following steps:

1. Group S arbitrarily such that each group is of size c except possibly for the last group.
2. Find the median of each group, and let S' be the set of the medians from each group.
3. Find the median m of S' by calling $\lfloor \frac{|S'|}{2} \rfloor$-selection on S', that is, by calling $ksel(S', \lfloor \frac{|S'|}{2} \rfloor)$.
4. Partition S into two sets S_1 and S_2 such that all elements in S_1 are smaller than m while all elements in S_2 are greater than m.
5. Recursively call k-selection $ksel(S_1, k)$ on S_1 if $k \leq |S_1|$; recursively call $(k - |S_1| - 1)$-selection $ksel(S_2, k - |S_1| - 1)$ on S_2 if $k > |S_1| + 1$; return m if $k = |S_1| + 1$.

Lemma 1. *$|S_1| \leq \frac{3}{4}N + c$ and $|S_2| \leq \frac{3}{4}N + c$.*

Proof. We first consider the lower bound of $|S_2|$. There are at least $\lceil \frac{N}{2c} \rceil - 1$ groups whose medians are greater than m. For each group, there are at least $\lceil \frac{c}{2} \rceil$ elements greater than the median of the group except possibly for the last group. Therefore, the total number of elements in S_2 is at least

$$|S_2| \geq (\lceil \frac{N}{2c} \rceil - 2)\lceil \frac{c}{2} \rceil = \lceil \frac{N}{2c} \rceil \lceil \frac{c}{2} \rceil - 2\lceil \frac{c}{2} \rceil \geq \frac{N}{4} - c - 1.$$

Therefore, the total number of elements in S_1 is at most $|S_1| = N - |S_2| - 1 \leq N - \frac{N}{4} + c = \frac{3}{4}N + c$. Similarly we can derive the upper bound on $|S_2|$. □

We first consider the time complexity $f(N)$ of this algorithm in RAM model. For some constant $c' > 0$,

$$f(N) = c'N + f(\frac{3}{4}N + c) + f(\frac{N}{c} + 1) \tag{1}$$

In order to get a linear cost in $f(N)$ it remains to ensure that $\frac{3}{4} + \frac{1}{c} < 1$. Why? — To analyze the complexity of this recursive function, we need to slightly introduce a powerful tool: generating functions.

Generating function. If $f(N)$ is a linear function, assume that $f(N) = aN + b$ where a and b are two undecided parameters with $a > 0$. Consider Equation 1, we have $aN + b = c'N + \big(a(\frac{3}{4}N + c) + b\big) + \big(a(\frac{N}{c} + 1) + b\big)$, that is, the following equation holds for any N,

$$(\frac{1}{4} - \frac{1}{c})aN = c'N + (ac + a + b). \tag{2}$$

To get a feasible solution to Equation 2, we have to find a and b such that

- $(\frac{1}{4} - \frac{1}{c})a = c'$, and
- $ac + a + b = 0$.

Since $c' > 0$ and $a > 0$, we have $\frac{1}{4} - \frac{1}{c} > 0$. From c and c' we can compute a; from a, c', c we can then compute b. In this way, we can find a linear function $f(N)$ that satisfies Equation 1.

Remarks. Consider the following questions:

1. What is $f(N)$ if $f(N) = N + 2f(N/2)$ with $f(1) = 1$?
2. How to plug the k-selection into quicksort to secure an $O(n \log_2 n)$ time complexity?
3. How to adapt the k-selection algorithm in RAM to the case where S is a multiset?
4. How to adapt the above algorithm to EM model such that the I/O cost is linear, i.e., $O(N/B)$ I/Os?

Quicksort partitions the input array into two parts using a splitter: it reports the resulting sequence, i.e., the sorted array, by concatenating the resulting sequence of the first part, the splitter, and the resulting sequence of of second part. In EM model, this algorithms entails $O(\frac{N}{B} \log_2 \frac{N}{B})$ I/Os. We wish to speed up the sorting process by dividing the sequence into more partitions using more splitters.

2.3.2 f-Spliter

Given a set S of elements and a parameter f, the problem of f-splitter finds f splitters $p_1, p_2, \cdots, p_f \in S$ in ascending order such that the for each $i \in [0, f]$, there are $O(N/f)$ number of elements in S that fall in $(b_i, b_{i+1}]$. Here $b_0 = -\infty$ and $b_{f+1} = +\infty$ are two pivot elements. The following process finds the *f-splitters* in EM model.

1. Divide elements in S in to $l = O(N/M)$ runs $\{R_1, R_2, \cdots, R_l\}$ such that each run is a *memoryload*. Let S' be an empty set.

2. For each run R_r, $r \in [1, l]$, sort the elements in main memory. Register the $(i \cdot f)$-th element to S' for every $i \in [1, \frac{|R_r|}{f}]$. After all runs are processed, S' will include $O(\frac{N}{M}\frac{M}{f}) = O(\frac{N}{f})$ elements.

3. Find the $i\lfloor\frac{|S'|}{f}\rfloor$-th element in S', denoted as p_i, using *k-selection* algorithm in *EM* for each $i \in [1, f]$.

Complexity. Step 1-2 take linear time. Since $|S'| = O(N/f)$, running f instances of *k-selection* costs $O(\frac{|S'|}{B}f) = O(\frac{N}{B})$ I/Os. Therefore, $p_0, p_1, p_2, \cdots, p_f, p_{f+1}$ can be found in linear I/Os.

Lemma 2. *If $f \leq \sqrt{M}$, then for any $i \in [0, f]$, the total number y_i of elements in S between p_i and p_{i+1} is $O(N/f)$.*

Proof. Consider run R_r, $\forall r \in [1, l]$, let x_r be the total number of elements in S' from R_r that falls in $[p_i, p_{i+1}]$. Therefore, there are at most $f(x_r + 1)$ elements in R_r that falls in $[p_i, p_{i+1}]$.

$$y_i = \Sigma_{r\in[1,l]} f(x_r + 1) = f\Sigma_{r\in[1,l]} x_r + lf = f\frac{|S'|}{f} + lf = O\left(\frac{N}{f} + \frac{Nf}{M}\right).$$

Since $f \leq \sqrt{M}$, $\frac{Nf}{M} \leq \frac{N}{\sqrt{M}} \leq \frac{N}{f}$. Therefore, $y_i = O(N/f)$. □

Remarks. Since each run contributes $O(\frac{M}{f})$ elements to S', to ensure that $\frac{M}{f} > B$ under $M > 2B$, we let $f = \sqrt{\frac{M}{B}}$.

2.3.3 Distribution Sort

Now, we are ready to step into the distribution sort. Given a set S of N elements, if S can fit into the main memory, i.e., $N = O(M)$, then we sort S in linear I/Os; otherwise, we proceed with three steps:

1. Find the f-splitters of S: $p_1, p_2, \cdots, p_f$ in linear I/Os where $f = \sqrt{\frac{M}{B}}$.

2. Disjointly partition S into $f + 1$ sets $S_0, S_1, S_2, \cdots, S_f$ based on the f splitters in linear I/Os.

3. Recursively call distribution sort for each partition and report the concatenated resulting sequence of that of each partition.

Complexity. Let $F(N)$ be the I/O complexity of distribution sort. There is a constants c such that

$$F(N) = \frac{cN}{B} + \Sigma_{i\in[1,f]} F(|S_i|).$$

Solve this recursion using a tree while each level disjointly partitions S and each leaf has M elements. The height of the tree is $\log_f(\frac{N}{M})$. Therefore, the complexity is $O(\frac{N}{B}\log_f(\frac{N}{B})) = O(\frac{N}{B}\log_{\frac{M}{B}}(\frac{N}{B}))$.

References

[1] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *CACM*, 31(9):1116–1127, 1988.

[2] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E.Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973.