Transcript of Lecture 3 of CSE 6512.

Saad Quader

September 6, 2011

1 Recap

1.1 Frazer and McKellar's `Samplesort` algorithm.

1. Pick a sample S of size s.
2. Sort S into $l_1, l_2, \cdots l_s$.
3. Partition X into $s+1$ groups X_i using members of sorted S such that

$$X_1 = \{x \in X | x \leq l_1\}, \tag{1}$$

$$X_i = \{x \in X | l_{i-1} < x \leq l_i\}, \ \forall 2 \leq i \leq s, \tag{2}$$

and

$$X_{s+1} = \{x \in X | x > l_s\}. \tag{3}$$

4. Sort each X_i using any asymptotically optimal sorting algorithm, e.g., `Heapsort` or `Mergesort`. This is a variation from the original paper which used *Samplesort* in recursion.
5. Output X_1 in sorted order, followed by X_2 in sorted order, ..., and X_{s+1} in sorted order.

1.2 Sampling Lemma

Lemma 1 *Let n be the input size, and s be the sample size. Then, the size of each partition X_i (described above) is $\tilde{O}\left(\frac{n}{s} \log n\right)$.*

2 Selection Problem

Input. $X = k_1, k_2, \cdots, k_n$, a sequence of n distinct keys and i, an integer $\leq n$ and ≥ 1.

Output. The i-th smallest element of X.

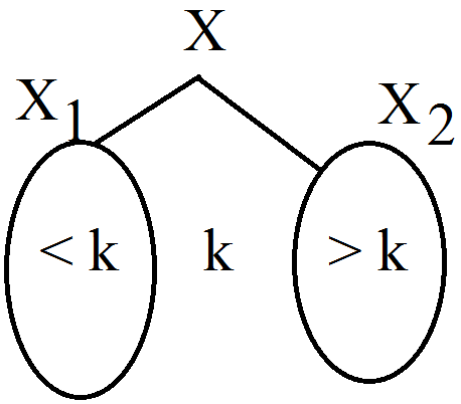

Figure 1: Partitioning of X by the `Select` algorithm.

3 `Select` Algorithm

Step 1.

Pick a pivot $k \in X$.

Step 2.

Partition X into two subsets X_1 and X_2 such that

$$X_1 = \{x \in X | x < k\} \tag{4}$$

and

$$X_2 = \{x \in X | x > k\} \ . \tag{5}$$

Figure 1 shows this partition. The partition takes n comparisons.

Step 3.

Count the elements in X_1 and X_2.

Case 1.

If $|X_1| = i - 1$ then output k.

Case 2.

If $|X_1| \geq i$ then output `Select`(i, X_1).

Case 3.

Otherwise, output `Select`$(i - |X_1| - 1, X_2)$.

Fact.

The average run time of `Select` is $O(n)$.

Fact.

We can convert `Select` into a Las Vegas algorithm, with an expected run time of $O(n)$, where the expectation is computed in the space of all possible outcomes for coin flips (and not the space of all possible inputs).

4 A Worst-case Linear-time Deterministic Selection Algorithm

This algorithm was proposed by Blum, Floyd, Pratt, Rivest, and Tarjan (1971). It is the same as `Select` except that the pivot element is picked in the following way:

1. Partition X into groups of 5 elements each.
2. Find the median of each of these groups.
3. Find M, the median of these medians.
4. Use M as the pivot element.

Fact.

It M is used as the pivot in `Select`, it is easy to see that $|X_1| \leq \frac{7n}{10}$ and $|X_2| \leq \frac{7n}{10}$.

Total Run Time.

$$T(n) = \Theta(n) + T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right). \tag{6}$$

$\Theta(n)$ is required for finding the group medians and for partitioning X into X_1 and X_2. $T\left(\frac{n}{5}\right)$ is required to find the median of medians, and the last term is for the recursive call (on either X_1 or X_2).

5 A Randomized Selection Algorithm

It is an extension of `Select` and `Samplesort`, proposed by Floyd and Rivest (1975).

Input: $X = k_1, k_2, \cdots, k_n$, a sequence of n distinct keys; i, an integer.
Output: The i-th smallest element of X.

Algorithm Outline

Pick a random sample S, made up of s elements from X. Let $l_1, l_2 \in S$ such that $l_1 < l_2$ and the i-th smallest element in X will be between l_1 and l_2 with *high probability.* If this condition is satisfied, we would discard all the elements that fall outside the interval $[l_1, l_2]$. Question: how to find l_1 and l_2, and what is the probability that the i-th smallest element will be in the said interval?

l_1 and l_2 could be trivially chosen as the minimum and maximum element of X, respectively; but this is not good enough; l_1 and l_2 should be as close as possible to each other so that we could discard as many elements as possible in each step.

Let q be the i-th element of X. This is the same as saying that the rank of q in X is (very nearly) i.

Definition 1 *Rank of any element* $x \in X$:

$$Rank(x, X) = |\{u \in X | u < x\}| + 1 \tag{7}$$

Note.

Number of items $< q$ in $X = i - 1$.
Expected number of items $< q$ in $S = \frac{s}{n}(i-1)$.

We will have to find a relationship, with high probability, between the rank of an element in S and that of the same element in X.

For any element w, if $Rank(w, S) = j$ then

$$E[Rank(w, X)] = \frac{n}{s} j \ . \tag{8}$$

Let $r_j = Rank(w, X)$. The following lemma gives the sought relationship with a high probability bound.

Lemma 2 (Rajasekaran & Reif, 1985)

$$Prob. \left[\left| r_j - j\frac{n}{s} \right| > \sqrt{4\alpha} \frac{n}{\sqrt{s}} \sqrt{\log n} \right] < n^{-\alpha} \tag{9}$$

Lemma 2 gives a lower bound on the probability that the i-th smallest element will be between elements l_1 and l_2. The proof of this lemma will be provided in the next lecture.

Let $[a_1, b_1]$ be a high probability confidence interval for the rank of l_1 in X (according to Lemma 2). Also, let $[a_2, b_2]$ be a high probability confidence interval for $Rank(l_2, X)$. Figure 2 shows these intervals.

Algorithm

In order to find the i-th smallest element in X,

1. Pick a random sample S, where $|S| = s$.

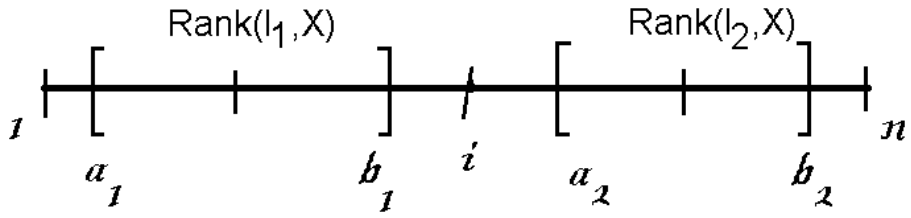

Figure 2: High probability confidence intervals for $Rank(l_1, X)$ and $Rank(l_2, X)$.

2. Pick two elements $l_1, l_2 \in S$ such that

$$Rank(l_1, S) = i\frac{s}{n} - \delta \tag{10}$$

and

$$Rank(l_2, S) = i\frac{s}{n} + \delta \tag{11}$$

where

$$\delta = c\sqrt{s \log n} \tag{12}$$

where c is a constant $> \sqrt{4\alpha}$ (for example $\sqrt{5\alpha}$).

We could sort S in $O(s \log s)$ time and find l_1 and l_2. However, if we use the linear time selection algorithm, it would take $O(s)$ time.

3. Find Y, the set of elements in X between l_1 and l_2.

$$Y = \{u \in X | l_1 \leq u \leq l_2\}. \tag{13}$$

Let n_1 be the number of elements less than l_1 and n_2 be the cardinality of Y: $n_1 = |\{u \in X | u < l_1\}|$, $n_2 = |Y|$. If $i > n_1$ and $i \leq (n_1 + n_2)$ then we will have the i-th smallest element in Y. However, it might not be the case if the sample is bad. In that case, we will start over. (We will have to calculate the probability of such starting over, and have to show that it is very small.)

Otherwise,

4. If the i-th smallest element is in Y, identify and output the $(i - n_1)$th smallest element in Y.

6 Time Complexity of the Randomized Selection Algorithm

Steps 1, 2, and 4

Step 1.

$O(s)$

Step 2.

$O(s)$

Step 4.

$O(|Y|)$

Step 3

In first consideration, we will need 2 comparisons for each element of X to determine whether it is in Y. (In total, $2n$ comparisons. Too bad.) We will have to do better.

There are two questions regarding Y:

Q1. Size of Y

Q2. Probability that the i-th smallest element is in Y

By substituting value of δ from Equation 12 in Equation 10 and using Equation 8, we can derive that

$$E[Rank(l_1, X)] = i - c\frac{n}{\sqrt{s}}\sqrt{\log n} \tag{14}$$

and

$$E[Rank(l_2, X)] = i + c\frac{n}{\sqrt{s}}\sqrt{\log n} \tag{15}$$

Using Equation 14, we find that

$$a_1 = \left(i - c\frac{n}{\sqrt{s}}\sqrt{\log n}\right) - \sqrt{4\alpha}\frac{n}{\sqrt{s}}\sqrt{\log n} \tag{16}$$

and

$$b_1 = \left(i - c\frac{n}{\sqrt{s}}\sqrt{\log n}\right) + \sqrt{4\alpha}\frac{n}{\sqrt{s}}\sqrt{\log n} \tag{17}$$

Similarly, using Equation 14 and Lemma 2 we find that

$$a_2 = \left(i + c\frac{n}{\sqrt{s}}\sqrt{\log n}\right) - \sqrt{4\alpha}\frac{n}{\sqrt{s}}\sqrt{\log n} \tag{18}$$

and

$$b_2 = \left(i + c\frac{n}{\sqrt{s}}\sqrt{\log n}\right) + \sqrt{4\alpha}\frac{n}{\sqrt{s}}\sqrt{\log n} \tag{19}$$

Note.

If $c > \sqrt{4\alpha}$ then the i-th smallest element of X will be in Y with high probability.

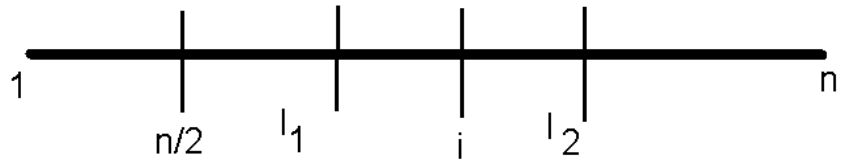

Figure 3: $i \geq \frac{n}{2}$

Determination of the size of Y.

Let $[a_1, b_]$ and $[a_2, b_2]$ be high probability confidence intervals for $Rank(l_1, X)$ and $Rank(l_2, X)$, respectively. Then,

$$\begin{aligned} |Y| &= b_2 - a_1 \\ &= 2c\frac{n}{\sqrt{s}}\sqrt{\log n} + 2\sqrt{4\alpha}\frac{n}{\sqrt{s}}\sqrt{\log n} \\ &= 2\frac{n}{\sqrt{s}}\sqrt{\log n}\left(c + \sqrt{4\alpha}\right) \end{aligned} \tag{20}$$

with probability $\geq (1 - n^{-\alpha})$.

Choosing sample size, $|S|$.

Try to keep $|S|$ equal to the order of $|Y|$. Pick $|S| = n^{2/3}$. In that case, with probability $\geq (1 - n^{-\alpha})$, the size of Y becomes

$$\begin{aligned} |Y| &= 2n^{2/3}\sqrt{\log n}(c + \sqrt{4\alpha}) \\ &= O\left(n^{2/3}\sqrt{\log n}\right) \ , \end{aligned} \tag{21}$$

Comparing the input keys with l_1 and l_2.

Case 1: $i \geq \frac{n}{2}$

Figure 3 depicts this situation. In this case, compare any key first with l_1, and then with l_2 only if needed. Therefore, one comparison for keys $\leq l_1$, and possibly two comparisons for other keys. Number of comparisons

$$= (i + \tilde{o}(n)) + 2(n - i). \tag{22}$$

The first term, i, overshoots the actual number of single comparisons by $i - Rank(l_1, X)$ which is much smaller than n; hence comes the second term $\tilde{o}(n)$. Rearranging, we get the number of comparisons

$$\begin{aligned} &= i + 2(n - i) + \tilde{o}(n) \\ &= n + (n - i) + \tilde{o}(n) \\ &= n + Min\{i, n - i\} + \tilde{o}(n) \ . \end{aligned} \tag{23}$$

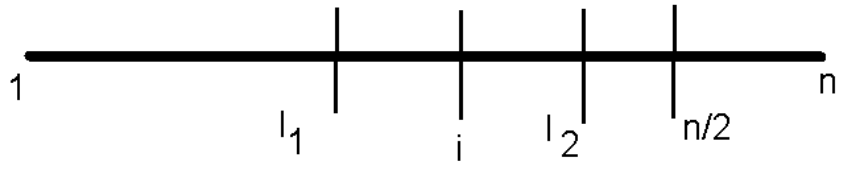

Figure 4: $i < \frac{n}{2}$

Because $i > \frac{n}{2}$, $Min\{i, n-i\}$ will be equal to $(n-i)$.

Case 2: $i < \frac{n}{2}$

Figure 4 depicts this situation. In this case, compare each key first with l_2; then if needed, with l_1. Number of comparisons

$$\begin{aligned} &= ((n-i) + \tilde{o}(n)) + 2i \\ &= n + Min\{i, n-i\} + \tilde{o}(n) \ . \end{aligned} \tag{24}$$

Because $i < \frac{n}{2}$, $Min\{i, n-i\}$ will be equal to i.

Total Run Time

The total run time for the above algorithm is shown below. (Each term denotes the run time of a step.)

$$\begin{aligned} T(n) &= O(|S|) + O(|S|) + (n + Min\{i, n-i\} + \tilde{o}(n)) + O(|Y|) \\ &= O\left(n^{2/3}\right) + O\left(n^{2/3}\right) + (n + Min\{i, n-i\}) + \tilde{O}\left(n^{2/3}\sqrt{\log n}\right) \\ &= O(n) + Min\{i, n-i\} + \tilde{O}\left(n^{2/3}\sqrt{\log n}\right) \end{aligned} \tag{25}$$

Thus, the expected run time of the randomized version is linear. Moreover, this is very close to the theoretical lower bound stated by the following theorem.

Theorem 1 *Finding the i-th smallest element from a sequence of n elements needs $n + Min\{i, n-i\}$ comparisons in the worst case.*