Selection in worst-case linear time

CLRS Sec. 10.3 pp. 189-191

We now examine a selection algorithm whose running time is $O(n)$ in the worst case. Like RANDOMIZED-SELECT, the algorithm SELECT finds the desired element by recursively partitioning the input array. The idea behind the algorithm, however, is to *guarantee* a good split when the array is partitioned. SELECT uses the deterministic partitioning algorithm PARTITION

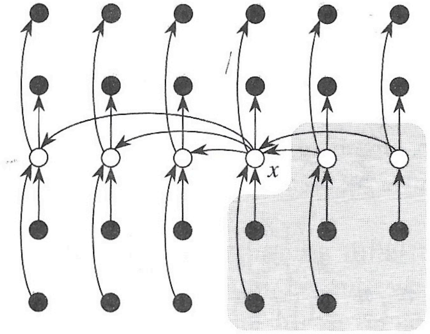

Figure 10.1 Analysis of the algorithm SELECT. The n elements are represented by small circles, and each group occupies a column. The medians of the groups are whitened, and the median-of-medians x is labeled. Arrows are drawn from larger elements to smaller, from which it can be seen that 3 out of every group of 5 elements to the right of x are greater than x, and 3 out of every group of 5 elements to the left of x are less than x. The elements greater than x are shown on a shaded background.

from quicksort (see Section 8.1), modified to take the element to partition around as an input parameter.

The SELECT algorithm determines the ith smallest of an input array of n elements by executing the following steps.

1. Divide the n elements of the input array into $\lfloor n/5 \rfloor$ groups of 5 elements each and at most one group made up of the remaining n mod 5 elements.
2. Find the median of each of the $\lceil n/5 \rceil$ groups by insertion sorting the elements of each group (of which there are 5 at most) and taking its middle element. (If the group has an even number of elements, take the larger of the two medians.)
3. Use SELECT recursively to find the median x of the $\lceil n/5 \rceil$ medians found in step 2.
4. Partition the input array around the median-of-medians x using a modified version of PARTITION. Let k be the number of elements on the low side of the partition, so that $n - k$ is the number of elements on the high side.
5. Use SELECT recursively to find the ith smallest element on the low side if $i \leq k$, or the $(i - k)$th smallest element on the high side if $i > k$.

To analyze the running time of SELECT, we first determine a lower bound on the number of elements that are greater than the partitioning element x. Figure 10.1 is helpful in visualizing this bookkeeping. At least half of the medians found in step 2 are greater than or equal to the median-of-medians x. Thus, at least half of the $\lceil n/5 \rceil$ groups contribute 3 elements that are greater than x, except for the one group that has fewer than 5 elements if 5 does not divide n exactly, and the one group containing x

itself. Discounting these two groups, it follows that the number of elements greater than x is at least

$$3\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{5} \right\rceil \right\rceil - 2\right) \geq \frac{3n}{10} - 6 .$$

Similarly, the number of elements that are less than x is at least $3n/10 - 6$. Thus, in the worst case, SELECT is called recursively on at most $7n/10 + 6$ elements in step 5.

We can now develop a recurrence for the worst-case running time $T(n)$ of the algorithm SELECT. Steps 1, 2, and 4 take $O(n)$ time. (Step 2 consists of $O(n)$ calls of insertion sort on sets of size $O(1)$.) Step 3 takes time $T(\lceil n/5 \rceil)$, and step 5 takes time at most $T(7n/10 + 6)$, assuming that T is monotonically increasing. Note that $7n/10 + 6 < n$ for $n > 20$ and that any input of 80 or fewer elements requires $O(1)$ time. We can therefore obtain the recurrence

$$T(n) \leq \begin{cases} \Theta(1) & \text{if } n \leq 80 , \\ T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n) & \text{if } n > 80 . \end{cases}$$

We show that the running time is linear by substitution. Assume that $T(n) \leq cn$ for some constant c and all $n \leq 80$. Substituting this inductive hypothesis into the right-hand side of the recurrence yields

$$\begin{aligned} T(n) &\leq c\lceil n/5 \rceil + c(7n/10 + 6) + O(n) \\ &\leq cn/5 + c + 7cn/10 + 6c + O(n) \\ &\leq 9cn/10 + 7c + O(n) \\ &\leq cn , \end{aligned}$$

since we can pick c large enough so that $c(n/10 - 7)$ is larger than the function described by the $O(n)$ term for all $n > 80$. The worst-case running time of SELECT is therefore linear.

As in a comparison sort (see Section 9.1), SELECT and RANDOMIZED-SELECT determine information about the relative order of elements only by comparing elements. Thus, the linear-time behavior is not a result of assumptions about the input, as was the case for the sorting algorithms in Chapter 9. Sorting requires $\Omega(n \lg n)$ time in the comparison model, even on average (see Problem 9-1), and thus the method of sorting and indexing presented in the introduction to this chapter is asymptotically inefficient.