

9.3 Linear-Time Selection

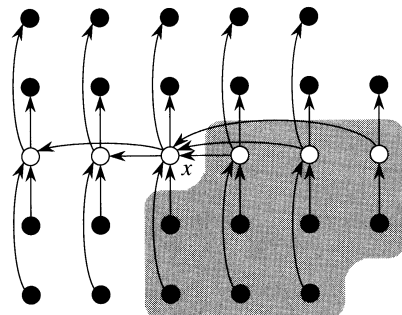
We want to find the i -th largest element of a set of n elements.

- ▶ **Step 1.** Divide the n elements into $\lceil n/5 \rceil$ groups of 5 elements each (the last group might be smaller than 5).
- ▶ **Step 2.** Find the median of each of the $\lceil n/5 \rceil$ groups by any method. (If the number of elements in the last group is even, take one of the two medians).
- ▶ **Step 3.** Recursively find the median x of the $\lceil n/5 \rceil$ group medians.

9.4 Analysis of Selection Algorithm

The choice of x guarantees that x is roughly in the middle 2/5 of the array. This provides a recurrence for the running time:

$$T(n) \leq \begin{cases} O(1) & \text{if } n \leq 140 \\ T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n) & \text{if } n > 140 \end{cases}$$



(Corman, p190)

9.3 Linear-Time Selection (continued)

- ▶ **Step 4.** Partition the set A into three disjoint sets using the median x of the median sequence:
 $R_1 = \{a_i \mid a_i < x\}$, $R_2 = \{a_i \mid a_i = x\}$, $R_3 = \{a_i \mid a_i > x\}$.
- ▶ **Step 5.**
If $|R_1| \geq i$ then find the i -th smallest element in R_1
Else if $|R_1| + |R_2| \geq i$ then return x
Else find the $(i - |R_1| - |R_2|)$ -th smallest element in R_3 .

9.4 Analysis of Selection Algorithm (continued)

We will prove by the method of substitution that $T(n) \leq cn$ for some constant c . Choose c large enough that $T(n) \leq cn$ for $n \leq 140$.

Also suppose the $O(n)$ term in the recurrence is bounded by an for $n > 140$.

For $n > 140$, the recurrence says

$$\begin{aligned} T(n) &\leq T(\lceil n/5 \rceil) + T(7n/10 + 6) + an & (1) \\ &\leq c\lceil n/5 \rceil + 7cn/10 + 6c + an & (2) \\ &\leq cn/5 + c + 7cn/10 + 6c + an & (3) \\ &= 9cn/10 + 7c + an & (4) \\ &= cn + (-cn/10 + 7c + an) & (5) \end{aligned}$$

The expression $-cn/10 + 7c + an$ is negative if $n > 140$ and $c \geq 20a$.

Therefore, for some c , $T(n) \leq cn$ always.

9.5 Quicksort - Another Application of Partitioning

Let a_1, a_2, \dots, a_n be a real sequence. The basic steps of quicksort are as follows:

- ▶ **Step 1.** Pick an element x as the **pivot element**.
- ▶ **Step 2.** Partition the sequence into subsequences,
 $R_1 = \{a_i \mid a_i < x\}$, $R_2 = \{a_i \mid a_i = x\}$, $R_3 = \{a_i \mid a_i > x\}$.
- ▶ **Step 3.** Sort R_1 and R_3 recursively, using Quicksort. (R_2 is already sorted.)
- ▶ **Step 4.** The combination of the three sorted lists R_1, R_2, R_3 is now a sorted version of the original list.

9.5 Quicksort (continued)

- ▶ The running time is $\Theta(n^2)$ in the worst case.
- ▶ The running time is $\Theta(n \lg n)$ in an average case: either for a random input, or using a random pivot.
- ▶ To make the worst case unlikely, use a random pivot or follow some rule justified by experience such as “median of three”.
- ▶ Small lists (less than 10–20 elements) are sorted faster by **insertion sort**. Therefore, use insertion sort on all small sublists rather than partitioning further.
- ▶ Implemented carefully, Quicksort is usually the fastest method for sorting arrays.