9.3 Linear-Time Selection

We aim to find the i-th smallest element from a set A containing n elements, $1 \leq i \leq n$.

The general strategy is to find **a pivot element** x in A such that a constant fractional number of elements in A can be discarded for further consideration.

For example, choose a pivot element x such that at least $n/10$ elements in A are removed from further consideration. Then, the i-th smallest element in A can be found from the remaining subset of A that contains at most $(1 - 1/10)n = 9n/10$ elements. This procedure continues until the i-th smallest element in A is found. Thus, the recursion depth h is $\lceil \log_{10/9} n \rceil$ since $(9/10)^h n = 1$.

The key to the proposed algorithm is how to find such a pivot element x within each iteration efficiently!

How many iterations are needed to find the i-th smallest element in A?

9.3 Linear-Time Selection Algorithm

- Step 1. Divide the n elements into $\lceil n/5 \rceil$ groups of 5 elements each (each group contains exactly 5 elements except the last group which may contain less than 5 elements).

- Step 2. Find the median of each group by any sorting method. (If the number of elements in the last group is even, take either one of the two medians). Consequently, a median sequence consisting of group medians is formed, which contains $\lceil n/5 \rceil$ elements exactly with each group having its median there.

9.3 Linear-Time Selection (continued)

- Step 3. Find the median x from the median sequence of $\lceil n/5 \rceil$ "group median" elements, using the linear selection algorithm recursively, where element x is the pivot element that will be used to partition the set A into three disjoint subsets R_1, R_2 and R_3.

- Step 4. Partition the n-element set A into three disjoint subsets R_1, R_2 and R_3, using the found x (as the pivot element), i.e., the set A is partitioned into three disjoints subsets R_1, R_2, and R_3 such that $A = R_1 \cup R_2 \cup R_3$ and $R_p \cap R_q = \emptyset$ if $p \neq q$ with $1 \leq p, q \leq 3$.

$$R_1 = \{a_l \mid a_l < x,\ a_l \in A\},$$
$$R_2 = \{a_l \mid a_l = x,\ a_l \in A\},$$
$$R_3 = \{a_l \mid a_l > x,\ a_l \in A\}.$$

9.3 Linear-Time Selection (continued)

➤ Step 5.

IF $(|R_1| \geq i)$ /* as we aim to find the ith smallest element from A */

THEN

Find the i-th smallest element in subset R_1;

ELSE

IF $(|R_1| + |R_2| \geq i)$

THEN return x;

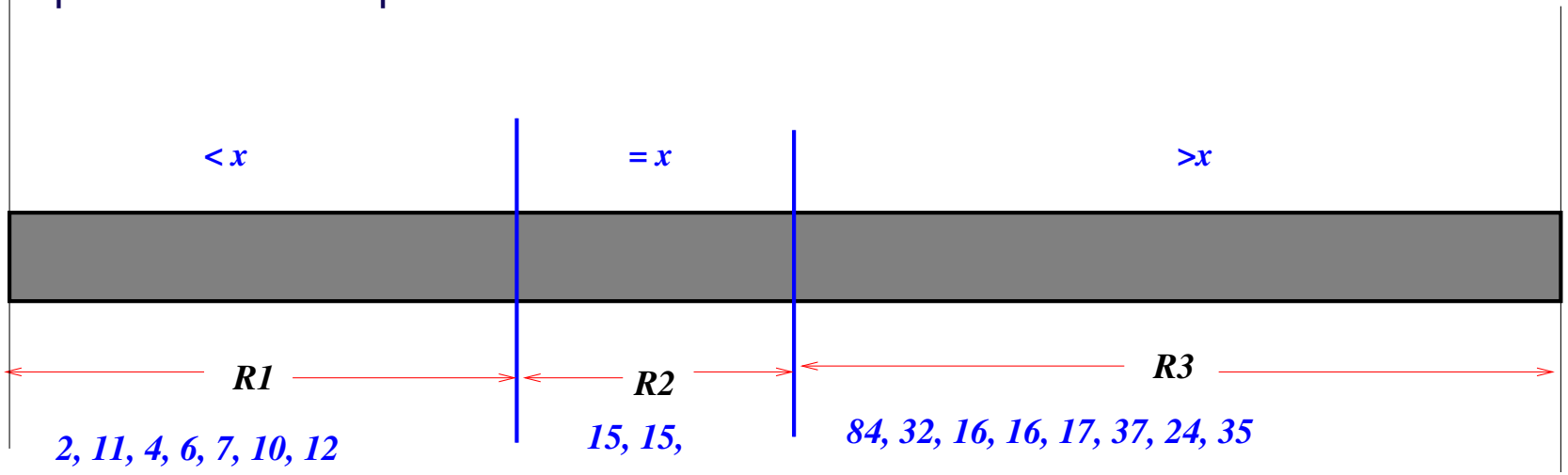
ELSE

Find the $(i - |R_1| - |R_2|)$-th smallest element in subset R_3.

Notice that only one of the three subsets in the next iteration, R_1, R_2, and R_3 will be examined, and the problem size then is the cardinality of the set to be examined, i.e., a constant fraction of the problem size in its immediately previous iteration.

9.3 Linear-Time Selection (continued)

The explanation of Step 5 is as follows.



A={ 15, 2, 11, 4, 6, 84, 32, 16, 7, 20, 16, 17, 37, 15, 10, 24, 35, 12}

x=15

If i<= |R1|, Select(R1,i)

If i> |R1| but i<=|R1|+|R2|, return x

otheriwse, Select(R3, i−(|R1|+|R2|))

9.4 Analysis of the Linear Selection Algorithm

- Step 1. Partition the n elements into $\lceil n/5 \rceil$ groups, it take linear time to scan the n elements, i.e., $O(n)$ time
- Step 2. Perform sorting within each group, thus it takes constant time to sort the 5-element in each group. In total, this step takes $\lceil n/5 \rceil \times O(1) = O(n)$ time, where each group contains no more than 5 elements, its sorting time is $O(1)$.
- Step 3. Find the median x of the median sequence of $\lceil n/5 \rceil$ elements, which takes $T(\lceil n/5 \rceil)$ time. Notice that the running time of the selection algorithm is applicable for any parameter i with $1 \leq i \leq |A|$.
- Step 4. Use the value of x to partition the set A into three disjoint subsets R_1, R_2 and R_3, it takes $O(n)$ time

9.4 Analysis of the Linear Selection Algorithm

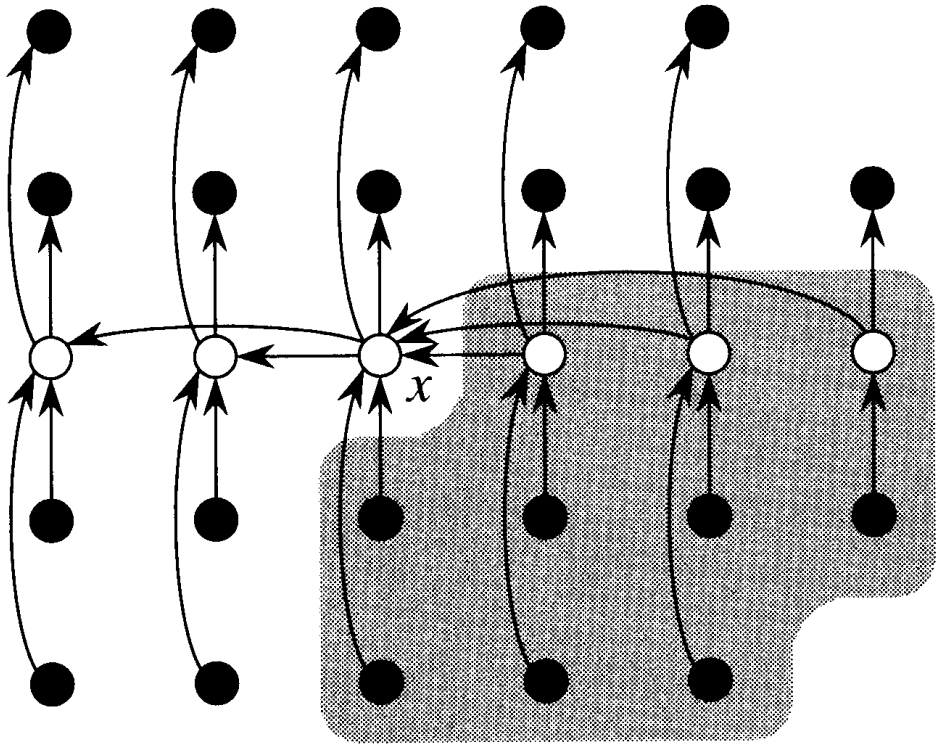
- Step 5. If $|R_1| < i \leq |R_1| + |R_2|$, done.
  Otherwise, call the selection algorithm either on R_1 or R_3, not both of them, it takes $\max\{T(|R_1|), T(|R_3|)\}$. Fortunately we can show that both $|R_1|$ and $|R_3|$ are less than $7n/10 + 6$, i.e., a fraction length of the original length n. Thus, this step takes $T(7n/10 + 6)$ time.

The choice of x guarantees that x is roughly in the middle of array A, i.e., $\approx 3n/5$ elements in A will be considered in the next round of the recursive call of the algorithm itself. This provides a recurrence for the running time of the linear selection algorithm:

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

where $T(\lceil n/5 \rceil)$ is the time for finding x in Step 3, $T(\frac{7n}{10} + 6)$ is the time at Step 5 by calling the algorithm based on the set of either R_1 or R_3, and $O(n)$ time needed for Step 1, Step 2, and Step 4.

9.4 Analysis of the Linear Selection Algorithm



(Corman, p221)

9.4 Analysis of The Selection Algorithm (continued)

Lemma: Both $|R_1| \leq \frac{7n}{10} + 6$ and $|R_3| \leq \frac{7n}{10} + 6$.

Proof: Following the diagram in the previous slide, we assume that the groups are listed according to the positions of their medians in the sorted median sequence. Notice that the finding of x from the median sequence is taken by calling the selection algorithm. We do NOT sort the median sequence.

We index the column groups into $1, 2, \ldots, \lceil n/5 \rceil$, and the column index of the group containing element x is $\lceil n/5 \rceil / 2$, thus, the elements in the right-bottom area (the black area of the diagram) are the elements in R_3, i.e., all of them are larger than x, (similarly, the elements in the left top area of the diagram are the elements in R_1). Then, $|R_3| \geq 3(\lceil n/5 \rceil / 2 - 2) \geq 3n/10 - 6$, where $(\lceil n/5 \rceil / 2 - 2)$ is the number of columns after the column of x minus the column of x itself (containing element x that is not in R_3) and the last column (which may contain less than 5 elements). Each of the rest columns contains at least 3 elements greater than x. Thus, $|R_1| \leq n - |R_3| \leq n - (3n/10 - 6) = 7n/10 + 6$. Similarly, we can show that $|R_1| \geq 3n/10 - 6$, then $|R_3| \leq n - |R_1| \leq 7n/10 + 6$, too.

9.4 Analysis of The Selection Algorithm (continued)

We shall 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 any $n > n_0 =$ **140** but $n' < n$. Also, suppose that the $O(n)$ term in the recurrence is bounded by an for $n \geq 140$ with $a > 0$ being constant.

For $n > 140$, the recurrence says

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

The expression $-cn/10 + 7c + an$ is no positive if $n \geq 140$ and $c \geq 20a$. How to figure out $n \geq 140$ and $c \geq 20a$?

9.4 Analysis of The Selection Algorithm (continued)

We have $T(n) \leq cn + (-cn/10 + 7c + an)$. We aim to show $T(n) \leq cn$.
Thus, we must have

$$cn + (-cn/10 + 7c + an) \leq cn, \quad \Rightarrow -cn/10 + 7c + an \leq 0.$$

$$\begin{aligned}
-cn/10 + 7c + an &\leq 0 && (8)\\
-cn + 70c + 10an &\leq 0 && (9)\\
10an &\leq (n-70)c && (10)\\
c &\geq \frac{10an}{n-70} && (11)\\
c &\geq \frac{10a}{1-70/n} && (12)
\end{aligned}$$

To ensure that $c > 0$, we can choose $n_0 = 140$, then $c \geq \frac{10a}{1-70/n_0} = \frac{10a}{1-1/2} = 20a$.
Therefore, for some c, $T(n) \leq cn$ always.

9.4 Analysis of The Selection Algorithm (continued)

Exercise:

- Does the group size (e.g., here group size is 5) affect the running time of the selection algorithm?
- Does the group size affect the fraction of numbers of elements discarded for further consideration (the next round calling of the selection algorithm)? (Yes/No, and Why?)
- If the group size is chosen as 3, is the selection algorithm still applicable? If not, can you justify why it is not applicable?

9.5 Quicksort

(An application of the linear selection algorithm)

Let $a_1, a_2, \ldots, a_n$ be a list of real numbers.

The basic steps of Quicksort are as follows:

- Step 1. Pick an element x as the pivot element.
- Step 2. Partition the list into three sublists,
  $R_1 = \{a_i \mid a_i < x\}$, $R_2 = \{a_i \mid a_i = x\}$, and $R_3 = \{a_i \mid a_i > x\}$.
- Step 3. Sort the elements in R_1 and R_3 recursively, by invoking Quicksort (as the elements in R_2 are already sorted.)
- Step 4. Combine the three sorted lists R_1, R_2, R_3 in this order into a sorted sequence.

9.5 Quicksort (continued)

- The running time of Quicksort is $O(n^2)$ **in the worst case**.
- The running time is $O(n \log n)$ **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.

9.5 Quicksort (continued)

Theorem. The running time of Quicksort is $O(n^2)$.

Consider the partitioning of elements in a set of n elements into three disjoint subsets R_1, R_2 and R_3. Assume that $|R_1| = k$ and let $T(n)$ be the running time of algorithm Quicksort for sorting n elements. Then,

$$T(n) = T(k) + T(n-k) + bn,$$

where $|R_1| = k$, $|R_3| = n - (|R_1| + |R_2|) \leq n - |R_1| = n - k$, and $b > 0$ is a constant to combine the three sorted subsequences into a sorted one. We then have

$$\begin{aligned}
T(n) &= T(1) + T(n-1) + bn,\\
T(n-1) &= T(1) + T(n-2) + b(n-1),\\
T(n-2) &= T(1) + T(n-3) + b(n-2),\\
&\cdots = \cdots\\
T(2) &= T(1),
\end{aligned}$$

$$T(n) = T(1) + b \cdot 1 + b \cdot 2 + \ldots + b(n-2) + b(n-1) + bn = T(1) + b \cdot \tfrac{n(n+1)}{2} = O(n^2).$$