Selection ..........................................................................

**input:** array $A[1..n]$ of numbers, integer $k$ in $\{1, \ldots, n\}$.

**output:** the $k$th smallest number in the array

When $k = 1$, the output should be $\min_i A[i]$. When $k = n$, the output should be $\max_i A[i]$. When $k = n/2$, the output should be the median. One simple way to solve the problem is to sort $A$, then return $A[k]$. This takes $\Theta(n \log n)$ time.

Here we present a linear-time algorithm. We assume that all elements in $A$ are distinct.

```
select(A[1..n], k)
1. if n ≤ 15, sort A in place and return A[k]        (since n ≤ 15, this takes constant time)
2. choose pivot p ∈ A[1..n] as described in the text below
3. partition A around p into A[1..i − 1], A[i] = p, and A[i + 1..n]
 (so that elements smaller than p are in A[1..i − 1] and elements larger than p are in A[i + 1..n])
4. if k = i: return A[i]
5. else if k < i: return select(A[1..i], k)                         (discard A[i..n])
6. else (k > i): return select(A[i + 1..n], k − i)                  (discard A[1..i])
```

We omit the proof of correctness (by induction on $n$) and focus on run time. Step 3 (the partition step) takes linear time. After that, SELECT discards some of the elements of $A$ and calls itself recursively on the rest. If the pivot is chosen badly, this could lead to quadratic running time. For example, if $k = 1$ and the pivot is chosen as the maximum element each time, then each recursive call would discard just one element, so there would be $n-1$ recursive calls, with the $i$th call taking time proportional to $n - i$, yielding total time $\Theta(\sum_{i=0}^{n} n - i) = \Theta(n^2)$.

To avoid this, the algorithm uses a complicated scheme to choose the pivot in Step 2. It first partitions the array $A$ into $n/5$ columns, each with 5 elements, as shown below (assuming $n$ is a multiple of 2 and 5, that is, a multiple of 10):

| | | | | | |
|---|---|---|---|---|---|
| $A[1]$ | $A[6]$ | | $A[n/2-2]$ | | $A[n-4]$ |
| $A[2]$ | $A[7]$ | | $A[n/2-1]$ | | $A[n-3]$ |
| $A[3]$ | $A[8]$ | $\cdots$ | $A[n/2]$ | $\cdots$ | $A[n-2]$ |
| $A[4]$ | $A[9]$ | | $A[n/2+1]$ | | $A[n-1]$ |
| $A[5]$ | $A[10]$ | | $A[n/2+2]$ | | $A[n]$ |

$\underbrace{\qquad\qquad\qquad\qquad}_{n/5}$

It then finds the median within each individual column of 5. This takes constant time for each column, so the total time for this step is linear. There are $n/5$ of these medians. It then *recursively computes the median of these $n/5$ medians, and chooses the pivot $p$ to be the result.* This completes the description of the algorithm. To analyze the run time, we use the following observation

**Observation 1.** *The rank $i$ of the pivot satisfies $3n/10 \le i \le 7n/10$, so the recursive call in Line 5 or 6 has at most $7n/10$ elements.*[1]

To see why, note that the pivot $p$, being the median of the $n/5$ column medians, is larger than $n/10$ of those medians. And each such median is in turn larger than 2 elements in its column. So $p$

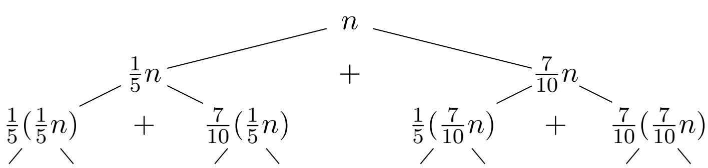
[1] This observation and some of the following discussion is not necessarily true when $n$ is not a multiple of 10, because we are ignoring rounding issues. See the discussion at the end of the note.

is larger than at least $3n/10$ other elements. By similar reasoning, $p$ is smaller than at least $3n/10$ other elements. So at least $3n/10$ elements are discarded in the recursive call.

Define $T(n)$ to be the worst-case running time of SELECT on $n$ elements. Dropping constant factors, the above reasoning gives the inequality

$$T(n) \leq n + T(n/5) + T(7n/10).$$

The next crucial observation is that sum of the sub-problem sizes is $n/5 + 7n/10 = 9n/10$, so the total problem size drops by a factor of 9/10:



| level | total size |
|---|---|
| 0 | $n$ |
| 1 | $\frac{1}{5}n + \frac{7}{10}n = \frac{9}{10}n$ |
| 2 | $\frac{9}{10}(\frac{1}{5}n) + \frac{9}{10}(\frac{7}{10}n) = (\frac{9}{10})^2 n$ |
| $\vdots$ | |

This happens at each level of the recursion tree. For each subproblem of any size $n'$, its two subproblems on the level below have combined size at most $\frac{9}{10}n'$, so the total size at each level is $\frac{9}{10}$th of the total size at the level above. So, by induction, the sum of the problem sizes in level $i$ is at most $(9/10)^i n$. Since the *work* for each subproblem is linear in the subproblem size, the total work in level $i$ is also $O((9/10)^i n)$. So the total work is $O(\sum_{i=0}^{O(\log n)} (9/10)^i n) = O(n)$ (using here that the sum is geometric).

**Correcting for rounding issues.** The previous discussion is technically wrong, because it ignores details related to rounding issues. For completeness, we give a more careful, correct analysis, starting with the choice of the pivot. When $n$ is not a multiple of 10, to choose the pivot $p$, the algorithm partitions into columns of 5 something like this:

| 1 | 2 | $\cdots$ | $m$ | $\cdots$ | $\lceil n/5 \rceil$ |
|---|---|---|---|---|---|
| $A[1]$ | $A[6]$ | $\cdots$ | $A[5m-4]$ | $\cdots$ | empty |
| $A[2]$ | $A[7]$ | | $A[5m-3]$ | $\cdots$ | empty |
| $A[3]$ | $A[8]$ | $\cdots$ | $A[5m-2]$ | $\cdots$ | $A[n-1]$ |
| $A[4]$ | $A[9]$ | $\cdots$ | $A[5m-1]$ | $\cdots$ | $A[n]$ |
| $A[5]$ | $A[10]$ | $\cdots$ | $A[5m]$ | $\cdots$ | empty |

There are $\lceil n/5 \rceil$ columns. The last column has between 1 and 5 five elements. After finding the $\lceil n/5 \rceil$ medians of those columns in $\Theta(n)$ time the algorithm recursively finds the median of those medians — specifically, the $m$th largest, where $m = \lfloor n/10 \rfloor$, and takes that to be the pivot $p$.

In one case, the algorithm discards $p$ and all elements smaller than $p$. This will be at least $3m$ elements. Recalling that $m = \lfloor n/10 \rfloor$, in this case at least $3\lfloor n/10 \rfloor$ elements are discarded. In the other case of interest, the algorithm discards $p$ and all elements larger than $p$. The number of such elements is at least

$$\begin{aligned} 3(\lceil n/5 \rceil - m + 1) - 2 &= 1 + 3(\lceil n/5 \rceil - m) \\ &= 1 + 3(\lceil n/5 \rceil - \lfloor n/10 \rfloor) \\ &\geq 1 + 3(n/5 - n/10) \\ &= 1 + 3n/10 \geq 3\lfloor n/10 \rfloor. \end{aligned}$$

Hence the number of remaining elements in the recursive call is at most $n - 3\lfloor n/10 \rfloor$, and

$$T(n) \le n + T(\lceil n/5 \rceil) + T(n - 3\lfloor n/10 \rfloor).$$

**Lemma 1.** *$T(n) = O(n)$.*

*Proof.* We give a proof by induction that $T(n) \le c\,n$ for some (large) constant $c$.

The base cases $n \le 80$ will hold as long as $c \ge \max\{T(n)/n : n \le 80\}$.

For $n > 80$, we have

$$\begin{aligned}
T(n) &\le n + T(\lceil n/5 \rceil) + T(n - 3\lfloor n/10 \rfloor) && \textit{(see above)} \\
&\le n + c\,\lceil n/5 \rceil + c\,(n - 3\lfloor n/10 \rfloor) && \textit{(by induction, using that } n \ge 10\textit{)} \\
&< n + c\,\big(n/5 + 1 + n - 3(n/10 - 1)\big) && \textit{(using } \lceil x \rceil < x + 1 \textit{ and } \lfloor x \rfloor > x - 1\textit{)} \\
&= c\,n\big(1/c + 4/n + 9/10\big) && \textit{(algebra)} \\
&\le c\,n(1/c + 1/20 + 9/10) && \textit{(using } n \ge 80\textit{)} \\
&\le c\,n && \textit{(as long as } c \ge 20\textit{).}
\end{aligned}$$

It follows that $T(n) \le c\,n$, where $c = \max\big(20, \max\{T(n)/n : n \le 80\}\big)$. $\square$

**External resources on deterministic selection**

- CLRS Chapter 9.3.
- Jeff Edmond's Chapter 1.7

  http://jeffe.cs.illinois.edu/teaching/algorithms/notes/01-recursion.pdf
- MIT lecture videos
  - https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/video-lectures/lecture-6-order-statistics-median/