Magic fives

*Oskar SKIBSKI**

* Faculty of Mathematics, Informatics and Mechanics, University of Warsaw

We all know that magic exists. Scientists try to explain most phenomena by mathematical equations or create new definitions to pretend that what cannot be described by equations also makes sense. However, we know very well that they succeed in convincing only those already convinced. If the rabbit is not in the hat, and the magician pulls it out of the hat, then there is no equation that could describe it. One is one, and zero is zero.†

† Editor's note: The (erroneous) opinions expressed in this article are the author's own and may not necessarily coincide (and they do not) with the views of *Delta*.

Since we in *Delta* consistently pretend that magic does not exist, we will discuss in this article an algorithm that appears to be magical, and yet is not. We can compare it with a magic trick in which the magician simply pulls a rabbit out of a hat, but does not show the audience beforehand that the hat is empty. For a moment it may surprise us ("Oh! Rabbit!"). However, with careful analysis and logical reasoning this trick can be explained: the rabbit must have been sitting on the magician's head throughout the performance, and while removing the hat the magician gently hooked the rabbit's legs and thus had something to pull out of the hat.

Magic trick: We have a hat with n rabb... no no, let's be at least a little serious! We have a set A containing n numbers (some numbers may repeat). One of the spectators says a number k from 1 to n. The magician in *linear time* finds the k-th largest number in the set A.

How to do the trick: For some values of k this task is very simple, e.g. for $k = 1$ our problem boils down to finding the smallest element in the set, which can be easily done in linear time. However, in general (e.g., if $k = \lceil n/2 \rceil$) it is not clear how we can do this. The case of $k = \lceil n/2 \rceil$ is, by the way, very important for statisticians, as it concerns the so-called *median*, which can have even more charm for them than the average. The natural idea is to sort all the elements and then point to the one in the k-th position. However, the fastest sorting algorithms run in $O(n \log n)$ time. Our problem seems much simpler than sorting all elements – how to solve it in linear time?

When we write that the running time of $f(n)$ is $O(g(n))$ we mean that f is at most of the order of g, that is, for large values of n the function $f(n)$ grows no faster than $g(n)$. Formally: there exist constants c and n_0 that for $n \geqslant n_0$ it holds $f(n) \leqslant c \cdot g(n)$. Thus, this is an upper bound estimate, but usually not the best one that can be found

We will borrow the idea for our solution from the magic trick of a woman sawn in half, or if you prefer, from *Hoare's algorithm*. Take a random element m and divide our entire set into two non-empty sets so that the first set contains only elements less than or equal to m (set $A_{\leqslant}$), and the second set: greater than or equal to m (set $A_{\geqslant}$). In order to make the sets non-empty, we can, for example, put all the elements less than or equal to m into the first set, and if the second set turns out to be empty put m into it. Now, if $A_{\leqslant}$ has at least k elements, then the element we are looking for must be in $A_{\leqslant}$ – so we recursively search for it in this set. On the other hand, if the set $A_{\leqslant}$ has less than k elements, then the element we are looking for is in the set $A_{\geqslant}$: we must therefore recursively find $(k - |A_{\leqslant}|)$-th largest element there.

To some readers, this may resemble the QuickSort algorithm. In the QuickSort algorithm, in order to sort a set of numbers, we divide it into elements smaller and larger than a certain element, and then sort both parts. It is no coincidence that the author of the QuickSort algorithm is also Tony Hoare.

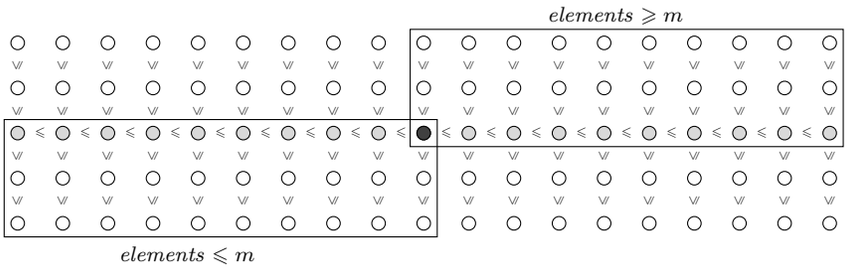
The idea is simple, but it may not be very efficient – if we are unlucky to always draw the smallest or the largest element, in every step our set A will decrease by only one element. And since each step requires linear time, the pessimistic running time of our algorithm will be quadratic: $O(n^2)$. That is even worse than with sorting!

However, our algorithm can be improved by changing the element based on which we divide the set. In the card trick, to find the card chosen by a spectator, the magician usually does not rely on fate, but carefully shuffles the deck to control where that card is. We will do the same – we will shuffle the elements a bit and draw one that guarantees that none of the parts are too big.

This is how the *median of medians* algorithm (in Polish called *the magic fives algorithm*) works. Let us divide arbitrarily our set into fives of elements (the last five can be incomplete) and for each find its median. Now, using our algorithm recursively, we find the median of the medians: we will denote it by m. Now, we divide our set into three parts: elements smaller than m (denoted by $A_{<}$), elements equal to m (denoted by $A_{=}$) and elements larger than m (denoted

by $A_>$). Now if $|A_<| \geqslant k$, we recursively look for the k-th largest element in it. If $|A_<| < k$, but $|A_<| + |A_=| \geqslant k$, it means that m is the k-th largest element. If, on the other hand, $|A_<| + |A_=| < k$, then we have to look recursively for our element in the set $A_>$ – it is the $(k - |A_<| - |A_=|)$-th largest there. Voila!

Explanation of the trick: Okay, but how can we be sure that none of the parts will be too big and the running time will be linear? Note that in half of the fives, the median is less than or equal to m. In each such five, at least half of the elements are less than or equal to m. It follows that at least $1/4$ of all the elements are less than or equal to m. This also means that no more than $3/4$ of all the elements are greater than m. Similarly, no more than $3/4$ of all the elements are smaller than m. This means that no matter which case occurs, we will recursively call our algorithm on a set reduced by at least 25%.



Is this enough to make our algorithm run in linear time? It turns out that it is! Let us denote the running time of our algorithm by $T(n)$. Dividing into fives and selecting from each the median can be done in linear time – $O(n)$. Selecting the median of the medians will take us $T(\lceil \frac{n}{5} \rceil)$ time. The last step is a recursive call for any of the parts. As we have already showed, none of the parts has more than $3/4$ of all elements, so the time is $T(\lceil \frac{3n}{4} \rceil)$. Thus, we get the following upper bound:

$$T(n) \leqslant O(n) + T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\left\lceil \frac{3n}{4} \right\rceil\right).$$

As it turns out this upper bound implies the running time of our algorithm is linear! The key here is the inequality $\frac{1}{5} + \frac{3}{4} = \frac{19}{20} < 1$, which ensures that $T(n)$ does not grow too fast.

To see this, first assume that n is a power of 20 so that it divides nicely by 4 and 5. We know that the time needed to divide into fives and pick medians can be bounded by cn for some natural c. It is easy to show by induction that $T(n) \leqslant 20cn$:

$$T(n) \leqslant cn + T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right) \leqslant cn + 20cn\left(\frac{1}{5} + \frac{3}{4}\right) = 20cn.$$

From the upper bound for powers of 20, we immediately get an upper bound for other numbers: Take any n and choose k so that $20^k < n \leqslant 20^{k+1}$. Since T is a non-decreasing function, we know that $T(n) \leqslant T(20^{k+1})$. But since $n > 20^k$ we get that

$$T(n) \leqslant T(20^{k+1}) \leqslant 20c \cdot 20^{k+1} \leqslant 20^2 cn.$$

Since there is a constant $20^2 c$ such that for any n there exists $T(n) \leqslant 20^2 c \cdot n$ this means that our algorithm runs in linear time: $O(n)$.

Finally, it remains to ask the question – why fives? It seems natural to take an odd number (so that there are middle values), but could we take threes or sevens? Or elevens?

It turns out that we could not take threes: we would first have to find the median of $1/3$ of all elements, and this would only reduce the problem by $1/4$. As $1/3 + 1/4 < 1$, that would result in $O(n \log n)$ time. We could instead take sevens, nines, etc: Looking for the median among $1/7$ or $1/9$ of all the elements would be even faster than looking for the median of $1/5$. However, we would increase the cost of finding the medians (i.e., that enigmatic c in the above proof) and the implementation would be more complex. So fives are used in the algorithm, because five is the smallest odd number greater than four. And that's the whole magic.

Solution to Problem F 1076.
If we neglect boundary effects, a uniform electric field perpendicular to the surface of the plates will appear between them after charging.The charge densities will also be uniform. Let's assume that the total surface area of each plate is S, then the charge densities will be: $\sigma_1 = Q_1/S$ and $\sigma_2 = Q_2/S$. Based on Gauss's law, we find that inside the capacitor, each plate is the source of an electric field with intensity:

$$E_i = \frac{\sigma_i}{2\varepsilon_0}.$$

In the above formula, ε_0 represents the vacuum permittivity. The electric field is directed "away from the plate" if its charge is positive and "towards the plate" if it is negative. The resultant electric field inside the capacitor is the sum of the fields from both plates and is given by:

$$E = E_1 - E_2 = \frac{\sigma_1 - \sigma_2}{2\varepsilon_0} = \frac{Q_1 - Q_2}{2\varepsilon_0 S},$$

and the value of the potential difference is:

$$U = Ed = (E_1 - E_2)d = \frac{(Q_1 - Q_2)d}{2\varepsilon_0 S} = \\ = \frac{Q_1 - Q_2}{2C}.$$

Solution to Problem F 1075.
Observations from directions between the two tangents to the surface of the stars (as shown in the diagram) are associated with the occurrence of eclipses. These tangents intersect at a point dividing the segment d into segments d_1 and d_2, $d = d_1 + d_2$. This corresponds to the range of observation angles α for which the inequality holds:

$$|\sin\alpha| \leqslant \frac{r_1}{d_1} = \frac{r_2}{d_2} = \frac{r_1 + r_2}{d}.$$

The model considered in the task describes the simplest case of a binary system. In general, gravitational interactions between the stars can lead to deformations from spherical to ellipsoidal shapes or even to the flow of matter between them.