Another k-Selection Algorithm

By Yufei Tao's Teaching Team

We have learned how to solve **the k-selection problem** in $O(n)$ expected time. The algorithm discussed in the lecture is easy to analyze but is not very efficient in practice.

Today, we will see another algorithm that runs faster in practice.

The k-Selection Problem: You are given

- a set S of n integers in an array A and
- an integer $k \in [1, n]$.

Design an algorithm to find the k-th smallest integer of S.

For example, suppose that $S = \{53, 92, 85, 23, 35, 12, 68, 74\}$ and $k = 3$. You should output 35.

Recall that the "lecture" k-selection algorithm starts by picking a pivot p uniformly at random from the input array A. It starts recursion only if the the rank of p falls in $[n/3, 2n/3]$.

Instead, our new algorithm will start recursion **anyway** regardless of the rank of p.

The New Algorithm

1. $p \leftarrow$ a uniformly random integer from the input array A
2. $r \leftarrow$ the rank of p
3. If $r = k$, then return p
4. If $r > k$, then produce an array B containing all the integers of A less than p. Recursively find the k-th smallest element in B
5. If $r < k$, then produce an array B containing all the integers of S greater than p. Recursively find the $(k - r)$-th smallest element in B

Example

Goal: find the 10-th smallest element from an array A of size 12:

17	26	38	28	41	72	83	88	5	9	12	35

Suppose that the random pivot p is 12, whose rank is 3.

As $3 < n/3 = 4$, the "lecture algorithm" will find another pivot. However, the new algorithm proceeds anyway. Specifically, it first produces an array B including only the elements larger than 12:

17	26	38	28	41	72	83	88	35

Then, it recursively finds the 7-th (note: $k - r = 10 - 3 = 7$) smallest element in B.

Although the new algorithm is procedurally simpler (than the lecture version), its analysis is more difficult.

Define $f(n)$ as the worst-case expected running time of the new algorithm on an array of size n. The algorithm cost includes three parts:

1. Picking a pivot p and getting its rank r: $O(n)$ time
2. Producing array B: $O(n)$ time
3. Recursing on B: $O(\max\{f(r), f(n-r)\})$ time.

Note that the cost of Part 3 is a random variable X depending on the value of r. We thus have:

$$f(n) \quad \leq \quad O(n) + \boldsymbol{E}[X].$$

Next, we will analyze $\boldsymbol{E}[X]$.

As the pivot p is picked uniformly at random, the value r is a uniformly distributed from 1 to n. Hence:

$$\begin{aligned}
\boldsymbol{E}[X] &= \sum_{i=1}^{n} (\text{the value of } X \text{ under } r = i) \cdot \boldsymbol{Pr}[r = i] \\
&= \frac{1}{n} \sum_{i=1}^{n} (\text{the value of } X \text{ under } r = i) \\
&= \frac{1}{n} \sum_{i=1}^{n} O(\max\{f(i), f(n-i)\}).
\end{aligned}$$

This yields the following recurrence:

$$f(n) \le O(n) + \frac{1}{n} \sum_{i=1}^{n} O(\max\{f(i), f(n-i)\}).$$

Using the substitution method, we can show that $f(n) = O(n)$. The details are shown in a regular exercise and will not be tested.