k-Selection

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong

The k-Selection Problem

Problem: You are given a set S of n integers in an array and also an integer $k \in [1, n]$. Design an algorithm to find the k-th smallest integer of S.

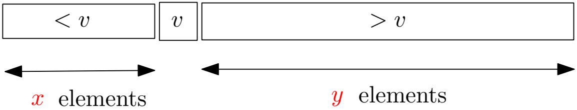
For example, suppose that $S = \{53, 92, 85, 23, 35, 12, 68, 74\}$ and $k = 3$. You should output 35.

This problem can be easily settled in $O(n \log n)$ time by sorting. Next, we will solve it in $O(n)$ expected time with randomization.

Idea

To illustrate the idea behind our algorithm, suppose that we pick an arbitrary element (say the first) v of S.

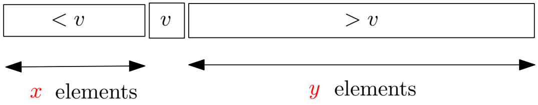
Move elements around so that those smaller than v are placed before v, and those larger are placed after v. This requires only $O(n)$ time (no sorting required).



- If $x = k - 1$, done (v is what we are looking for).
- If $x < k - 1$, recurse by performing $(k - (x + 1))$-selection on the y elements to the right of v (subproblem).
- If $x > k - 1$, recurse by performing k-selection on the x elements to the left of v (subproblem).

Idea

Obstacle: x or y can be very small (0 if we are unlucky) such that we can throw away only few elements before recursion.



Wish: Make $x \geq n/3$ and $y \geq n/3$.
Antidote: Randomly select v from the whole array! Wish comes true with probability $1/3$!

New obstacle: Would still fail with probability $2/3$.
New antidote: Choose another v if we fail; 3 repeats in expectation!

Algorithm

The **rank** of an integer v in S is the number of elements in S smaller than or equal to v.

For example, suppose that $S = (53, 92, 85, 23, 35, 12, 68, 74)$. Then, the rank of 53 is 4 and that of 12 is 1.

Finding the rank of v in S takes only $O(|S|)$ time.

Algorithm

1. Randomly pick an integer v from S; call v the **pivot**.
2. Get the rank r of v.
3. If r is not in $[n/3, 2n/3]$, repeat from Step 1.
4. Otherwise:

    4.1 If $k = r$, return v.
    4.2 If $k < r$, perform k-selection on the elements of S less than v.
    4.3 If $k > r$, perform $(k - r)$-selection on the elements of S greater than v.

Example

Goal: find the 10-th smallest element from 12 elements:

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

Suppose that the pivot v chosen happens to be 12, whose rank is 3, outside the range $[4, 8]$. We repeat by randomly choosing another pivot v, which — let us assume — happens to be 83. Again, its rank 11 is outside the range $[4, 8]$. Repeat another time; let the pivot v returned by 35, whose rank is 7.

We recurse by finding the 3-rd smallest element in:

38	41	72	83	88

Cost Analysis

Step 1 (on Slide 6) takes $O(1)$ time.
Step 2 takes $O(n)$ time.

How many times do we have to repeat the above two steps?
With a probability $1/3$, we can proceed to Step 3 $\Rightarrow$ need to repeat only 3 times in expectation!

When we are at Step 3, A has at most $\lceil 2n/3 \rceil$ elements left.

Cost Analysis

Let $f(n)$ be the expected running time of our algorithm on an array of size n.

We know from the earlier analysis:

$$
\begin{aligned}
f(1) &\leq O(1) \\
f(n) &\leq O(n) + f(\lceil 2n/3 \rceil).
\end{aligned}
$$

Solving the recurrence gives $f(n) = O(n)$ (The Master's theorem).

It is worth mentioning that the k-selection problem can be solved in $O(n)$ time deterministically. However, the algorithm is much more complicated. This demonstrates the power of randomization again.