More on k-selection

CSCI2100 Tutorial 5

Jianwen Zhao

Department of Computer Science and Engineering
The Chinese University of Hong Kong

Adapted from the slides of the previous offerings of the course

Introduction

Last week, in the lectures, we have learned the k-selection problem and solved it in $O(n)$ expected time by making use of randomization. The k-selection algorithm discussed in the class is easy to understand and analyze, but less efficient in practice.

In this tutorial, we will introduce a simpler and faster randomized algorithm (but with a more tedious analysis) and discuss another interesting problem related to k-selection.

A "simpler" randomized algorithm

1. Randomly pick an integer v from S.
2. Get the rank of v, let it be r.
3. if $r = k$, return v, otherwise:

   3.1 if $r > k$, produce an array S' containning all the integers of S strictly smaller than v. Recurse on S' by finding the k-th smallest element in S'.

   3.2 if $r < k$, produce an array S' containning all the integers of S strictly larger than v. Recurse on S' by finding the $(k - r)$-th smallest element in S'.

Example

Consider that we want to find the 10-th smallest element from a set S of 12 elements:

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

Suppose that the v we randomly choose is 28, whose rank is 6. Since $6 < 10$, we generate an array S' with only the elements larger than 28:

38	41	72	83	88	35

Then we can just recurse by finding the 4-th ($k - r = 10 - 6 = 4$) smallest element in this arrary S'.

Remark

The above algorithm is procedurally simpler than the one we taught in the class, and is faster in practice too. It, however, is less interesting in two ways:

1. Its analysis is more complicated (in the mundane way).
2. It does not illustrate the "if-failed-then-repeat" technique.

k-selection on two sorted arrays

Problem: Let $X[1...n]$ and $Y[1..m]$ be two arrays, both sorted in ascending order. We want to find the k-th smallest of the $n+m$ elements where $1 \leq k \leq n+m$. Our algorithm has to end in $O(\log n + \log m)$ time.

Example: X:

2	3	6	7	9	12

Y:

1	4	8	10	11

Suppose $k = 5$, then our algorithm should output 6, since the final sorted array is:

1	2	3	4	6	7	8	9	10	11	12

Solution

We solve this problem by resursion.

Base case

The base case happens when either n or m is 1. Without loss of genarality, assume that $m = 1$ (Otherwise, swap the roles of X and Y).

- If $k = n + 1$, then return $\max\{X[n], Y[1]\}$.
- Otherwise(i.e., $k \leq n$):
  - If $X[k] < Y[1]$, then return $X[k]$.
  - Otherwise, return $\max\{X[k-1], Y[1]\}$.

Obviously, the base case can be solved in $O(1)$ time.

Reduce case

Take:

1. The median element u of X, namely, $u = X[s]$ where $s = \lfloor n/2 \rfloor$
2. The median element v of Y, namely, $v = Y[t]$ where $t = \lfloor m/2 \rfloor$

Without loss of genarality, we assume $v \leq u$ (Otherwise, swap the roles of X and Y). We distinguish two cases:

- Case 1: $s + t \geq k$: None of the elements in $X[s+1, ...n]$ can possibly be the result. We recurse by searching for the k-th smallest element of the $s + m$ elements in $X[1...s]$ and $Y[1...m]$.
- Case 2: $s + t < k$: None of the elements in $Y[1, ...t]$ can possibly be the result. We recurse by searching for the $(k - t)$-th smallest element of the $n + m - t$ elements in $X[1...n]$ and $Y[t+1...m]$.

Example

Input X:

2	8	11	17	20	33	35

Y:

1	4	7	28	30	43

$k = 5$

Where $n = 7$, $m = 6$, $s = \lfloor n/2 \rfloor = 3$, $t = \lfloor m/2 \rfloor = 3$.

We take $u = X[s] = 11$, $v = Y[t] = 7$, and $u > v$.

Since $k = 5$, $s + t = 6 > k$, which followes case 1, then none of the elements in $X[4, ...7]$ can possibly be the result. We recurse by searching for the 5-th smallest element of the 9 elements in $X[1...3]$ and $Y[1...6]$, i.e.

New Input 1 Y:

2	8	11

X:

1	4	7	28	30	43

$k = 5$

Example

New Input 1 Y: | 2 | 8 | 11 | X: | 1 | 4 | 7 | 28 | 30 | 43 | $k = 5$

Where $n = 6$, $m = 3$, $s = \lfloor n/2 \rfloor = 3$, $t = \lfloor m/2 \rfloor = 1$.

We take $u = X[s] = 7$, $v = Y[t] = 2$, and $u > v$.

Since $k = 5$, $s + t = 4 < k$, which followes case 2, then $Y[1]$ cannot possibly be the result. We recurse by searching for the $5 - 1 = 4$-th smallest element of the 8 elements in $X[1...6]$ and $Y[2...3]$, i.e.

New Input 2 X: | 8 | 11 | Y: | 1 | 4 | 7 | 28 | 30 | 43 | $k = 4$

Example

New Input 2 X: | 8 | 11 | Y: | 1 | 4 | 7 | 28 | 30 | 43 | $k = 4$

Where $n = 2$, $m = 6$, $s = \lfloor n/2 \rfloor = 1$, $t = \lfloor m/2 \rfloor = 3$.

We take $u = X[s] = 8$, $v = Y[t] = 7$, and $u > v$.

Since $k = 4$, $s + t = 4 = k$, which followes case 1, then $X[2]$ cannot possibly be the result. We recurse by searching for the 4-th smallest element of the 7 elements in $X[1]$ and $Y[1...6]$, i.e.

New Input 3 Y: | 8 | X: | 1 | 4 | 7 | 28 | 30 | 43 | $k = 4$

Example

New Input 3 Y: | 8 | X: | 1 | 4 | 7 | 28 | 30 | 43 | $k = 4$

This comes to be the base case, since $k = 4 < n = 6$, $X[k] = X[4] = 28 > Y[1]$, we return $\max\{X[k-1], Y[1]\} = 8$.

Cost Analysis

From the above example, we can see that for each recursion, we shrink either X or Y by half. Overall, the above shrinking can happen at most $\log_2 m + \log_2 n$ times before reaching the base case.

It thus follows that the entire algorithm finishes in $O(\log n + \log m)$ time.