# CSCI2100: Regular Exercise Set 4

Prepared by Yufei Tao

**Problem 1.** Recall that our RAM model has been extended with an atomic operation RANDOM$(x, y)$ which, given integers $x, y$, returns an integer chosen uniformly at random from $[x, y]$. Suppose that you are allowed to call the operation *only* with $x = 1$ and $y = 128$. Describe an algorithm to obtain a uniformly random number between 1 and 100. Your algorithm must finish in $O(1)$ expected time.

**Solution.** Call RANDOM(1,128) and let $z$ be its return value. Output $z$ if it is in $[1, 100]$. Otherwise, repeat from the beginning. We need to call the operator twice in expectation because each time $z$ has probability 100/128 to fall in the range we want.

**Problem 2*.** Suppose that we enforce an even harder constraint that you are allowed to call RANDOM$(x, y)$ *only* with $x = 0$ and $y = 1$. Describe an algorithm to generate a uniformly random number in $[1, n]$ for an arbitrary integer $n$. Your algorithm must finish in $O(\log n)$ expected time.

**Solution.** We first obtain the smallest power of 2 that is at least $n$. For this purpose, set $x = 1$, and double $x$ each time until $x \geq n$. The final $x$ is the power of 2 we are looking for. This takes $O(\log n)$ time.

Next we will generate a uniformly random number $y$ in $[1, x]$. For this purpose, call RANDOM$(0, 1)$, and let $z$ be its return. If $z = 0$, we proceed to generate a random number in $[1, x/2]$ recursively; otherwise, proceed in $[(x/2) + 1, x]$ recursively. Note that the range of numbers has shrunk by half. The recursion goes on $O(\log n)$ steps before the range contains only one number, which is the $y$ we want.

Return $y$ if $y \leq n$. Otherwise, repeat by generating another $y$. Since $y \geq x/2$, at most 2 repeats are needed in expectation. The overall time is therefore $O(\log n)$ in expectation.

**Problem 3.** For the $k$-selection problem, consider an input array $A$ that has $n = 120$ elements. Our randomized algorithm selects a number $v$, and recurse into a smaller array $A'$ if the rank of $v$ is within $[n/3, 2n/3] = [40, 80]$. For $k = 20$, what is the probability that the size of $A'$ is at most 60?

**Solution.** $A'$ has size at most 60 if the rank of $v$ is between 40 and 61. The probability that this happens is $(61 - 40 + 1)/(80 - 40 + 1) = 22/41$.

**Problem 4* (Textbook Exercise 9.3-8).** Let $X[1..n]$ and $Y[1..n]$ be two arrays, each containing $n$ integers in ascending order. Consider that all the $2n$ integers are distinct. Let $k$ be an integer between 1 and $2n$. Give an $O(\log n)$-time algorithm for finding the $k$-th smallest of the $2n$ elements.

**Solution.** We will consider a more general scenario where the two arrays can have different lengths. 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 is recursive.

*Base case:* The base case happens when either $n$ or $m$ is 1. Without loss of generality, assume that $m = 1$. We can solve the problem in $O(1)$ time as follows. If $k = n + 1$, the return $\max\{X[n], Y[1]\}$. Otherwise (i.e., $k \leq n$):

- If $X[k] < Y[1]$, then return $X[k]$.
- Otherwise, return $\max\{X[k-1], Y[1]\}$.

*Reduce case:* Take (i) the median element $u$ of $X$, namely, $u = X[s]$ where $s = \lfloor n/2 \rfloor$, and (ii) the median element $v$ of $Y$, namely, $v = Y[t]$ where $t = \lfloor n/2 \rfloor$. Without loss of generality, assume that $v \leq u$ (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 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 the $n+m-t$ elements in $X[1..n]$ and $Y[t+1..m]$.

In any case, we spend $O(1)$ time and shrink one array by half for the recursion. Overall, the above shrinking can happen at most $\log_2 n + \log_2 m$ times before reaching the base case. It thus follows that the entire algorithm finishes in $O(\log n + \log m)$ time. Therefore, the original problem (where $n = m$) can be settled in $O(\log n)$ time.

**Problem 5** (A Simpler Randomized Algorithm for k-Selection, but with a More Tedious Analysis ).** In the $k$-selection problem, we have an array $S$ of $n$ distinct integers (not necessarily sorted). We would like to find the $k$-th smallest integer in $S$ where $k \in [1, n]$. Here is another way of solving it using randomization. If $n = 1$, then we simply return the only element in $S$. For $n > 1$, we proceed as follows:

- Randomly pick an integer $v$ in $S$, and obtain the rank $r$ of $v$ in $S$.
- If $r = k$, return $v$.
- If $r > k$, produce an array $S'$ containing the integers of $S$ that are smaller than $v$. Recurse by finding the $k$-th smallest in $S'$.
- Otherwise, produce an array $S'$ containing the integers of $S$ that are larger than $v$. Recurse by finding the $(r-k)$-th smallest in $S'$.

Prove that the above algorithm finishes in $O(n)$ expected time.

**Solution.** Let $f(n)$ be the expected time of the above algorithm on an input of size $n$. Clearly, $f(0) = O(1)$ and $f(1) = O(1)$.

Consider $n > 1$. The rank $r$ of $v$ is uniformly distributed in $[1, n]$, namely, for each $i \in [1, n]$, $\mathbf{Pr}[r = i] = 1/n$. When $r = i$, it determines a "left subset" containing the $i - 1$ integers of $S$ smaller than $v$, and a "right subset" of size $n - i$. In the worst case, we recurse into the larger of the two subsets, namely, we would need to solve the problem on an array of size $\max\{i-1, n-i\}$. This gives rise to the following recurrence (for some constant $\alpha > 0$):

$$\begin{aligned} f(n) &\leq \alpha \cdot n + \frac{1}{n}\sum_{i=1}^{n} f(\max\{i-1, n-i\}) \\ &\leq \alpha \cdot n + \frac{2}{n}\sum_{i=\lceil n/2 \rceil}^{n} f(i-1) \end{aligned}$$

We will prove that the recurrence leads to $f(n) \leq cn$ for some constant $c > 0$. First, this is obviously true for $n \leq 24$ when $c$ is at least a certain constant, say $\beta$ (when $n = O(1)$, the algorithm definitely finishes in constant time).

Suppose that $f(n) \leq cn$ for $n \leq k-1$ where $k \geq 24$. Set $t = \lceil k/2 \rceil$. We have:

$$\begin{aligned} f(k) &\leq \alpha \cdot k + \frac{2}{k}\sum_{i=t}^{k} c(i-1) = \alpha \cdot k + \frac{2c}{k}\sum_{i=t-1}^{k-1} i \\ &= \alpha \cdot k + \frac{2c}{k}\frac{(k+t-2)(k-t+1)}{2} < \alpha \cdot k + \frac{c(k^2+3t-t^2)}{k} \qquad (1) \\ &= (\alpha+c)k + 3c - c\frac{t^2}{k} \leq (\alpha+c)k + 3c - c\frac{(k/2)^2}{k} \\ &= (\alpha+c)k + 3c - ck/4 \qquad (2) \end{aligned}$$

We need the above to be at most $ck$, namely:

$$\begin{aligned} (\alpha+c)k + 3c - ck/4 &\leq ck \\ \Leftrightarrow \alpha k + 3c &\leq ck/4 \\ \Leftarrow \begin{cases} ck/4 \geq 2\alpha k \\ ck/4 \geq 6c. \end{cases} \\ \Leftarrow \begin{cases} c \geq 8\alpha \\ k \geq 24. \end{cases} \end{aligned}$$

Hence, setting $c = \max\{\beta, 8\alpha\}$ completes the proof.

**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: (i) its analysis is more complicated (in the mundane way), and (ii) it does not illustrate the "if-failed-then-repeat" technique.