CSCI3160: Regular Exercise Set 1

Prepared by Yufei Tao

Problem 1. Recall that our RAM model has 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. Consider the following algorithm to find the greatest common divisor of n and m where $n \leq m$:

algorithm $GCD(n, m)$
    if $n = 0$ **then**
        return m
    $m = m - n$
    if $n \leq m$ **then return** $GCD(n, m)$
    else return $GCD(m, n)$

Prove:

1. The time complexity of the algorithm is $O(m)$.
2. The time complexity of the algorithm is $\Omega(m)$.

Solution.
Proof of Statement 1: Each time a recursive call to the algorithm is made, $\max\{n, m\}$ decreases by at least 1. Therefore, there can be at most m calls overall. Each call clearly takes $O(1)$ time.

Proof of Statement 2: Fix $n = 1$. It is clear that the algorithm must make m calls.

Problem 4. Consider an input array A that has $n = 120$ distinct elements. Suppose that we choose a number v in A uniformly at random. What is the probability that the rank of v (among all the numbers in A) fall in the range $[35, 78]$?

Solution. $(78 - 35 + 1)/120 = 44/120$.

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 β (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:

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

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.