1 The Average Case Performance of QuickSort & QuickSelect

Carl Sturtivant, 2005/12/6

The recurrence for the **average case runtime of QuickSort** is as follows (as discussed in class).

$$Q(n) = \frac{2}{n-1}\sum_{k=1}^{n-1} Q(k) + p(n)$$

where $p(n)$ is the time to partitition n elements, which is known to be $\Theta(n)$. Similarly, the recurrence for the **average case runtime of QuickSelect** (or **RandomizedSelect**, as the book calls it) is as follows (as discussed in class).

$$R(n) = \frac{1}{n-1}\sum_{k=1}^{n-1} R(k) + p(n)$$

Note the similarity, the main difference being that QuickSelect makes one recursive call, and QuickSort makes two. In both algorithms, all of the work is done by various calls of partition at various levels of recursion.

Since the cost of partition $p(n)$ is known to be $\Theta(n)$, we know that for $n \geq n_0$ that $c_1 n \leq p(n) \leq c_2 n$, and so if we argue for $n \geq n_0$ as our starting point, we may overestimate the cost of partition by replacing it by $c_2 n$ or underestimate it by replacing it by $c_1 n$. The solutions to the corresponding recurrences where $p(n)$ has been replaced by $c_2 n$ and $c_1 n$ thus provide upper and lower bounds on the actual average case runtimes. However, (e.g. in the case of QuickSort) both upper and lower bounds are obtained from the following equation with either $c = c_1$ or $c = c_2$.

$$B(n) = \frac{2}{n-1}\sum_{k=1}^{n-1} B(k) + cn$$

and defining $B(n) = cT(n)$ gives

$$T(n) = \frac{2}{n-1}\sum_{k=1}^{n-1} T(k) + n$$

so that **provided the asymptotic growth does not depend upon the initial conditions** (which will be scaled by c_1 and c_2), **both the upper and lower bounds will have the same rate of growth as** $T(n)$. A similar argument goes for RandomizedSelect. This justifies replacing $p(n)$ by n in the original recurrences, if we are interested in the solutions in asymptotic (Θ) notation. **Note that an argument of the preceding pattern works in many situations when analyzing the runtime of a recursive algorithm.**

So we need to solve the following, where the number of recursive calls $r = 1$ for RandomizedSelect and $r = 2$ for QuickSort, and verify that the rate of growth of the solution does not depend upon the initial conditions. First, get rid of the sum by scaling by $(n-1)$, rewriting the resulting equation with $(n+1)$ in place of n and subtracting.

$$\begin{aligned}
T(n) &= \frac{r}{n-1}\sum_{k=1}^{n-1} T(k) + n \\
(n-1)T(n) &= r\sum_{k=1}^{n-1} T(k) + n(n-1) \\
nT(n+1) &= r\sum_{k=1}^{n} T(k) + (n+1)n \\
nT(n+1) - (n-1)T(n) &= rT(n) + (n+1)n - n(n-1) \\
nT(n+1) &= (n+r-1)T(n) + 2n
\end{aligned}$$

This gives the simple equation $nT(n+1) = nT(n)+2n$ for RandomizedSelect since $r = 1$. Dividing by n we have $T(n+1) = T(n) + 2$ (i.e. each value is 2 more than the previous one) which has the solution $T(n) = 2n + \mathit{const}$ which is $\Theta(n)$ no matter what the initial conditions are. So we may conclude that the runtime of RandomizedSelect $R(n) = \Theta(n)$.

This leaves the recurrence $nT(n+1) = (n+1)T(n)+2n$ to solve for QuickSort, since $r = 2$. Using the sneaky trick of dividing through by n and by $(n + 1)$ gives $T(n + 1)/(n + 1) = T(n)/n + 2/(n + 1)$. Writing $T(n)/n = u(n)$ gives the following equation, and (replacing n by $k - 1$) we may sum to find a solution.

$$\begin{aligned}
u(n+1) &= u(n) + 2/(n+1) \\
u(k) &= u(k-1) + 2/k \\
\sum_{k=1}^{n} u(k) - u(k-1) &= \sum_{k=1}^{n} 2/k \\
u(n) &= \mathit{const} + 2\sum_{k=1}^{n} \frac{1}{k} \\
T(n) &= \mathit{const}.n + 2n\sum_{k=1}^{n} \frac{1}{k}
\end{aligned}$$

And, since $\sum_{k=1}^{n} \frac{1}{k}$ is very much like $\int_1^n \frac{dx}{x} = \ln n$ it is easy to show that it is $\Theta(\lg n)$ and consequently $T(n) = \mathit{const}.n + 2n.\Theta(\lg n) = \Theta(n \lg n)$ and once again the location of the arbitrary constant in the solution shows that this is independent of the initial conditions. So we may conclude that the runtime of QuickSort $Q(n) = \Theta(n \lg n)$.