CSE 332: Data Structures and Parallelism

QuickCheck: Recurrences Solutions (due Thursday, January 19)

Master Theorem

Consider a recurrence of the form

$$T(n) = \begin{cases} d & \text{if } n = 1 \\ aT\left(\frac{n}{b}\right) + n^c & \text{otherwise} \end{cases}$$

Then,

- If $\log_b(a) < c$ then $T \in \Theta(n^c)$
- If $\log_b(a) = c$ then $T \in \Theta(n^c \lg(n))$
- If $\log_b(a) > c$ then $T \in \Theta(n^{\log_b(a)})$

0. Sum Sum Sum

Consider the following code:

```
f(n) {
   if (n == 0) {
      return 0
   }
   int result = 0
   for (int i = 0; i < n; i++) {
      result += i * i + n
   }
   return f(n/3) + 2 * result + 3 * f(n/3)
}
```

(a) Find a recurrence $T(n)$ modeling the worst-case time complexity of $f(n)$.

Solution:

We look at the three separate cases (base case, non-recursive work, recursive work). The base case is $\mathcal{O}(1)$, because we only do a return statement. The non-recursive work is $\mathcal{O}(1)$ for the assignments and `if` tests and $\sum_{i=0}^{n-1} 1 = n$ for the loop. The recursive work is $2T\left(\frac{n}{3}\right)$.

Putting these together, we get:

$$T(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2T\left(\frac{n}{3}\right) + n & \text{otherwise} \end{cases}$$

(b) Find a Big-Oh bound for your recurrence.

Solution:

Since we are asked to only find a Big-Oh bound, and strictly speaking don't need to find a closed form, we can use the Master Theorem to find our answer.

Note that $a = 2$, $b = 3$, and $c = 1$. We see that $\log_b(a) = \log_3(2) < 1 = c$, so know that $T \in \Theta(n^1)$ by the Master Theorem.

By definition of Big-Theta, we also know that $T \in \mathcal{O}(n)$ must be true.