CSE 332: Data Structures and Parallelism

Section 3: Recurrences and Closed Forms Solutions

0. Not to Tree

For the following code snippet, find a recurrence for the worst case runtime of the function, and then find a closed form for the recurrence.

Consider the function f:

```
f(n) {
   if (n <= 0) {
      return 1;
   }
   return 2 * f(n - 1) + 1;
}
```

- Find a recurrence for $f(n)$.

  Solution:

$$T(n) = \begin{cases} c_0 & \texttt{if } n <= 0 \\ T(n-1) + c_1 & \texttt{otherwise} \end{cases}$$

- Find a closed form for $f(n)$.

  Solution:

  Unrolling the recurrence, we get $T(n) = \underbrace{c_1 + c_1 + \cdots + c_1}_{n \text{ times}} + c_0 = c_1 n + c_0$.

1. To Tree

Consider the function h:

```
h(n) {
   if (n <= 1) {
      return 1
   } else {
      return h(n/2) + n + 2*h(n/2)
   }
}
```

(a) Find a recurrence $T(n)$ modeling the *worst-case runtime complexity* of $h(n)$.

Solution:

$$T(n) = \begin{cases} c_0 & \text{if } n \leq 1 \\ 2T\left(\frac{n}{2}\right) + c_1 & \text{otherwise} \end{cases}$$

(b) Find a closed form to your answer for (a).

Solution:

The recursion tree has height $\lg(n)$, each non-leaf level i has has work $c_1 2^i$, and the leaf level has work $c_0 2^{\lg(n)}$. Putting this together, we have:

$$\begin{aligned}\left(\sum_{i=0}^{\lg n - 1} c_1 2^i\right) + c_0 2^{\lg(n)} = c_1 \left(\sum_{i=0}^{\lg n - 1} 2^i\right) + c_0 n &= c_1 \frac{1 - 2^{\lg n - 1 + 1}}{1 - 2} + c_0 n \\ &= c_1 2^{\lg n} - c_1 + c_0 n \\ &= c_1(n-1) + c_0 n \\ &= (c_0 + c_1)n - c_1\end{aligned}$$

2. To Tree or Not to Tree

Consider the function f. Find a recurrence modeling the worst-case runtime of this function and then find a Big-Oh bound for this recurrence.

```
f(n) {
   if (n == 0) {
      return 0
   }
   int result = f(n/2)
   for (int i = 0; i < n; i++) {
      result *= 4
   }
   return result + f(n/2)
}
```

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

Solution:

We look at the three separate components (base case, non-recursive work, recursive work). The base case is a constant amount of work, because we only do a return statement. We'll label it c_0. The non-recursive work is a constant amount of work (we'll call it c_1) for the assignments and `if` tests and a constant (we'll call c_2) multiple of n for the loops. The recursive work is $2T\left(\frac{n}{2}\right)$.

Putting these together, we get:

$$T(n) = \begin{cases} c_0 & \text{if } n = 0 \\ 2T\left(\frac{n}{2}\right) + c_2 n + c_1 & \text{otherwise} \end{cases}$$

(b) Find a closed form for $f(n)$

Solution:

The recursion tree has $\lg(n)$ height, each non-leaf node of the tree does $c_2 \dfrac{n}{2^i} + c_1$ work, each leaf node does c_0 work, and each level has 2^i nodes.

So, the total work is $\left(\sum_{i=0}^{\lfloor \lg(n) \rfloor - 1} 2^i (c_2 \frac{n}{2^i} + c_1)\right) + c_0 \cdot 2^{\lg n} = \left(\sum_{i=0}^{\lfloor \lg(n) \rfloor - 1} (2^i c_1 + c_2 n)\right) + c_0 n = c_1 \frac{1 - 2^{\lg n}}{1 - 2} + c_2 n \lg(n) + c_0 n = c_2 n \lg(n) + c_1(n - 1) + c_0 n$.

3. Big-Oof Bounds

Consider the function f. Find a recurrence modeling the worst-case runtime of this function and then find a Big-Oh bound for this recurrence.

```
f(n) {
   if (n == 0) {
      return 0
   }

   int result = 0
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < i; j++) {
         result += j

      }
   }
   return f(n/2) + result + f(n/2)
}
```

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

Solution:

We look at the three separate components (base case, non-recursive work, recursive work). The base case is a constant amount of work, because we only do a return statement. We'll label it c_0. The non-recursive work is a constant amount of work (we'll call it c_1) for the assignments and `if` tests and a constant (we'll call c_2) multiple of $\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$ for the loops. The recursive work is $2T\left(\frac{n}{2}\right)$.

Putting these together, we get:

$$T(n) = \begin{cases} c_0 & \text{if } n = 0 \\ 2T\left(\frac{n}{2}\right) + c_2\frac{n(n-1)}{2} + c_1 & \text{otherwise} \end{cases}$$

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

Solution:

Since we only want a Big-Oh, we can actually leave off lower-order terms when doing our analysis, as they won't affect the runtime bounds; so, we can ignore the constants c_1 and c_2 in our analysis.

Note that $\frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2} \in \mathcal{O}(n^2)$. We can, again, ignore the lower-order term $\left(\frac{n}{2}\right)$ since we only want a Big-Oh bound.

The recursion tree has $\lg(n)$ height, each non-leaf node of the tree does $\left(\frac{n}{2^i}\right)^2$ work, each leaf node does c_0 work, and each level has 2^i nodes.

So, the total work is $\sum_{i=0}^{\lfloor \lg(n) \rfloor - 1} 2^i \left(\frac{n}{2^i}\right)^2 + c_0 \cdot 2^{\lg n} = n^2 \sum_{i=0}^{\lfloor \lg(n) \rfloor - 1} \left(\frac{2^i}{4^i}\right) + c_0 n < n^2 \sum_{i=0}^{\infty} \left(\frac{1}{2^i}\right) + c_0 n = \frac{n^2}{1 - \frac{1}{2}} + c_0 n.$

This expression is upper-bounded by n^2 so $T \in \mathcal{O}(n^2)$.

4. Odds Not in Your Favor

Consider the function g. Find a recurrence modeling the worst-case runtime of this function, and then find a closed form for the recurrence.

```
g(n) {
   if (n <= 1) {
      return 1000
   }
   if (g(n/3) > 5) {
      for (int i = 0; i < n; i++) {
         println("Yay!")
      }
      return 5 * g(n/3)
   }
   else {
      for (int i = 0; i < n * n; i++) {
         println("Yay!")
      }
      return 4 * g(n/3)
   }
}
```

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

Solution:

$$T(n) = \begin{cases} c_0 & \text{if } n \leq 1 \\ 2T\left(\frac{n}{3}\right) + c_1 n + c_2 & \text{otherwise} \end{cases}$$

(b) Find a closed form for the above recurrence.

Solution:

The recursion tree has height $\log_3(n)$, each non-leaf level i has work $(\frac{c_1 n}{3^i} + c_2)2^i$, and the leaf level has work $c_0 2^{\log_3(n)}$. Putting this together, we have:

$$\begin{aligned}
\sum_{i=0}^{\log_3(n)-1} \left((\frac{c_1 n}{3^i} + c_2) 2^i \right) + c_0 2^{\log_3(n)} &= \sum_{i=0}^{\log_3(n)-1} \left(\frac{c_1 n 2^i}{3^i} + c_2 2^i \right) + c_0 2^{\log_3(n)} \\
&= c_1 n \Big(\sum_{i=0}^{\log_3(n)-1} (\frac{2}{3})^i \Big) + c_2 \Big(\sum_{i=0}^{\log_3(n)-1} 2^i \Big) + c_0 2^{\log_3(n)} \\
&= c_1 n \left(\frac{1 - \left(\frac{2}{3}\right)^{\log_3(n)}}{1 - \frac{2}{3}} \right) + c_2 \Big(\frac{1 - 2^{log_3(n)}}{1-2} \Big) + c_0 2^{\log_3(n)} \quad \text{Finite geometric series} \\
&= 3c_1 n \left(1 - \left(\frac{2}{3} \right)^{\log_3(n)} \right) c_2 (2^{log_3 n} - 1) + c_0 2^{\log_3(n)} \\
&= 3c_1 n \left(1 - \frac{n^{\log_3(2)}}{n} \right) + c_2 (n^{log_3 2} - 1) + c_0 n^{\log_3(2)} \\
&= 3c_1 n - 3c_1 n^{\log_3(2)} + c_2 n^{log_3(2)} - c_2 + c_0 n^{\log_3(2)} \\
&= 3c_1 n + (c_0 + c_2 - 3c_1) n^{\log_3(2)} - c_2
\end{aligned}$$