CSE 2321: Notes on Recurrence Relations 2

1. In class we studied the Merge Sort algorithm for sorting a list of numbers (see the pseudocode posted on Piazza in Resources > Recurrence Relations and Recursion). Merge Sort is a "divide and conquer" algorithm that works by recursively sorting both halves of the list and then merging the two sorted halves together.

   We have previously looked at the Insertion Sort and Selection Sort algorithms; the worst-case running time of these algorithms is $\Theta(n^2)$. Below, we analyze the running time of Merge Sort.

   If the length of the list is n, we can represent the time it takes Merge Sort to sort as $T(n)$. If $n = 1$, then then the list has exactly one element and is already sorted, so the algorithm takes constant time; that is, $T(1) = c_1$. For $n > 1$, it takes time $T(n/2)$ to sort each half of the list, and time cn to merge the sorted halves together. The recurrence for the running-time is therefore

   $$T(1) = c_1$$

   .

   $$T(n) = 2T\left(\frac{n}{2}\right) + cn$$

   .

   Below, we solve this recurrence by substitution.

   $$\begin{aligned}
   (1)\ T(n) &= 2T\left(n/2\right) + cn \\
   &= 2[2T\left(n/2^2\right) + c(n/2)] + cn \\
   (2)\ &= 2^2T\left(n/2^2\right) + cn + cn \\
   &= 2^2[2T\left(n/2^3\right) + c(n/2^2)] + cn + cn \\
   (3)\ &= 2^3T\left(n/2^3\right) + cn + cn + cn \\
   &\cdots \\
   (k)\ &= 2^kT\left(n/2^k\right) + kcn
   \end{aligned}$$

   The process stops when $(n/2^k) = 1$ or $k = \log_2(n)$, so

   $$2^kT\left(n/2^k\right)+kcn = 2^{\log_2(n)}T\left(1\right)+cn\log_2(n) = c_1n+cn\log_2(n) = \Theta(n\log(n))$$

   This result means that Merge Sort is asymptotically faster than Selection Sort and Insertion Sort.

2. Now consider the following recurrence. Note that it is similar to the merge sort recurrence, but there is only one recursive call.

$$T(1) = c_1$$
$$T(n) = T\left(\frac{n}{2}\right) + cn$$

Note that $T(n) \geq cn$, so cn is a lower bound on the recurrence. That is,

$$T(n) = \Omega(n)$$

Next, we find an upper bound.

$$\begin{aligned}
(1)\ T(n) &= T\left(\frac{n}{2}\right) + cn \\
(2)\ &= T\left(\frac{n}{2^2}\right) + \frac{1}{2}cn + cn \\
(3)\ &= T\left(\frac{n}{2^3}\right) + \frac{1}{2^2}cn + \frac{1}{2}cn + cn \\
(4)\ &= T\left(\frac{n}{2^4}\right) + \frac{1}{2^3}cn + \frac{1}{2^2}cn + \frac{1}{2}cn + cn \\
&\cdots \\
(k)\ &= T\left(\frac{n}{2^k}\right) + \left(\frac{1}{2}\right)^{k-1}cn + \left(\frac{1}{2}\right)^{k-2}cn + \cdots + \left(\frac{1}{2}\right)^{2}cn + \left(\frac{1}{2}\right)^{1}cn + cn
\end{aligned}$$

Factoring out cn, this becomes:

$$T(n) = T\left(\frac{n}{2^k}\right) + cn\left(\left(\frac{1}{2}\right)^{k-1} + \left(\frac{1}{2}\right)^{k-2} + \cdots + \left(\frac{1}{2}\right)^{2} + \left(\frac{1}{2}\right)^{1} + 1\right)$$

The process stops when $\left(\frac{n}{2^k}\right) = 1$ or $k = \log_2(n)$, so

$$T(n) = T(1) + cn\left(\frac{1}{2^{\log_2(n)-1}} + \frac{1}{2^{\log_2(n)-2}} + \cdots + \frac{1}{2^2} + \frac{1}{2} + 1\right)$$

While we could analyze the sum in the parenthesis precisely, there is no need to do this in this particular case. The sum is a geometric series with $r = \frac{1}{2}$. Recall that if r is less than 1, then

$$\sum_{i=0}^{\infty} r^i = \frac{1}{1-r}$$

So

$$T(n) = c_1 + cn\left(\frac{1}{2^{\log_2(n)-1}} + \frac{1}{2^{\log_2(n)-2}} + \cdots + \frac{1}{2^2} + \frac{1}{2} + 1\right) < c_1 + cn\sum_{i=0}^{\infty}\left(\frac{1}{2}\right)^i = 1 + 2\times cn.$$

Therefore, $T(n) = O(n)$. Since we showed above that $T(n) = \Omega(n)$, it follows that $T(n) = \Theta(n)$