Solutions - Topic 7, 02/08: Recurrence Relations

---

1. Consider the following pseudocode:

```
whiskey_tango_foxtrot(A[1..n])
       if n==1
           return A[1]
       whiskey_tango_foxtrot(A[1..(n/3)])
       whiskey_tango_foxtrot(A[(n/3)+1..(2n/3)])
       whiskey_tango_foxtrot(A[(2n/3)+1..n])
       for i = 1 to n-1
           for j = n to i+1
               if A[j] < A[j-1]
                   exchange A[j] and A[j-1]
```

You have just shown that the recurrence relation is

$T(1) = O(1)$

$T(n) = 3T(n/3) + O(n^2)$

a. Solve this recurrence using Master Theorem

Solution:

a = 3, b = 3, $f(n) = n^2$, $n^{\log_3 3} = n^1 = n$.

$f(n) = n^2 = \Omega(n^{1+0.001}) = \Omega(n^{1+\varepsilon})$. Case 3 of Master theorem, thus,

$$T(n) = \Theta(n^2)$$

b. Solve this recurrence using the substitution method

Solution:

To use the substitution method, we must rewrite recursion more precisely, using specific constants.

$$T(n) = 3T(n/3) + cn^2$$

Let's guess $T(n) = cn^2$ and try to prove it.

$$T(n) = 3T(n/3) + cn^2 = 3 \cdot c(n/3)^2 + cn^2 = 3cn^2/9 + cn^2 = (4/3)cn^2 \neq cn^2$$

Wrong guess!

Try a new guess:

$T(n) = dn^2$ for some constant d, that we have to determine

$$T(n) = 3T(n/3) + cn^2 = 3 \cdot d(n/3)^2 + cn^2 = 3 \cdot d \cdot \frac{n^2}{9} + cn^2 = (d/3 + c)n^2$$

We want this to equal to our guess, which is dn^2. Thus we want,

$(d/3 + c)n^2 = dn^2$

$d/3 + c = d$

$d = \frac{3}{2}c$

I.e. our guess should be $T(n) = \frac{3}{2}cn^2$. Let's check if it works:

$$T(n) = 3T(n/3) + cn^2 = 3 \cdot \frac{3}{2}c\left(\frac{n}{3}\right)^2 + cn^2 = \frac{9}{2}c \cdot \frac{n^2}{9} + cn^2 = \frac{1}{2}cn^2 + cn^2 = \frac{3}{2}cn^2$$

Exactly like our guess! :-)

Extra Credit

Finished early? Here is additional practice with solving recurrences. If you run out of time you'll get credit for identifying a well justified "guess".

2. Solve the following recurrence using whatever solution method(s) work. If the Master Method works, identify which case it is and solve it. If it does not work, analyze the structure of the recursion tree to arrive at a "guess" and use substitution to prove it.

$T(n) = T(\sqrt{n}) + c$ if $n > 2$

$T(n) = c$ if $n \leq 2$

Solution:

This recursion does not fit the Master method, so we need to solve it using substitution method. To make a good guess, let's see what's going on. By viewing it as a recursive procedure in some program.

At the beginning the procedure is called on input of size *n*. Within it, there is one recursive call on size $\sqrt{n}$, which calls another recursive call on size $\sqrt{\sqrt{n}} = n^{1/4}$, which calls another recursive call on size $\sqrt{n^{1/4}} = n^{1/8}$, and so on, until the size of the input is 2.

Each recursive call performs *c* operations. Thus, the runtime defined by this recursion is:

T(n) = c+c+c+...+c, as many times as it takes to reduce the input from *n* to 2. How many times is that?

Let's look at the sizes of each recursive call. They follow the following progression:
$n,\ n^{1/2},\ n^{1/4},\ n^{1/8},\ n^{1/16},\ ...,\ n^{(1/2)^i},\ ..., 16,\ 4,\ 2$.

Let's say the number of recursive calls is *k*. Then
$n,\ n^{1/2},\ n^{1/4},\ n^{1/8},\ n^{1/16},\ ...,\ n^{(1/2)^{k-1}}, n^{(1/2)^k} = 2$.
Let's solve for *k*.

$n^{(1/2)^k} = 2$	Take logarithm (base 2) of both sides
$(1/2)^k \cdot \log n = \log 2 = 1$	Multiply both sides by 2^k
$2^k \cdot (1/2)^k \cdot \log n = 2^k$	Simplify
$\log n = 2^k$	Take logarithm of both sides
$\log\log n = k \cdot \log 2$	Simplify ($\log_2 2\ =\ 1$)
$\log\log n = k$	

So the depth of the recursion is $k = \log\log n$. Thus, a good guess for the runtime is $T(n)\ =\ c\log\log n$. Let's test it by substitution.

$$T(n)\ = T(\sqrt{n})\ +\ c\ =\ c \cdot \log\log(\sqrt{n})\ + c\ =\ c \cdot \log\log(n^{1/2}) + c = c \cdot \log((1/2) \cdot \log n) + c$$
$$= c \cdot (\log(1/2)\ + \log\log n)\ +\ c\ =\ c \cdot (-1 + \log\log n)\ +\ c\ =\ -c + c \cdot \log\log n + c$$
$$= c \cdot \log\log n$$

Q.E.D