CSE 332 Summer 18
Section 03

1 Solving Recurrences

For each of the following recurrences, use the tree method to find a closed form of the recurrence.

When using the tree method, you should do the following steps.

0. Draw at least the first two levels of the recursion tree, and the leaf level of the tree.
1. Let the root node be at level 0. Give a formula for the size of the input at level i.
2. What is the number of nodes at level i?
3. What is the work done at the i^{th} recursive level?
4. What is the last level of the tree?
5. What is the work done at the base case?
6. Write an expression for the total work done. Your expression should include a summation.
7. Find a "closed form" of the formula in the previous part. To qualify as a closed form, it must not have any summations or recursion, but it does not have to "look nice."
8. If possible, use Master Theorem to sanity check your answer.

a) $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 3 & \text{otherwise} \end{cases}$

Solution:

1. $\frac{n}{2^i}$
2. 1
3. $1 \cdot 3$
4. We want the i such that $\frac{n}{2^i} = 1$. Solving we get $\log_2(n)$ as the last level.
5. $1 \cdot 1$
6. $\sum_{i=0}^{\log_2(n)-1} 3 + 1$
7. $3\log_2(n) + 1$
8. Since $\log_2(1) = 0$ so the Master theorem says we should get $\Theta(n^0 \log n)$ i.e. $\Theta(\log n)$, which matches our answer from the last part.

b) $T(n) = \begin{cases} 1 & \text{if } n = 0 \\ T(n-1) + 2 & \text{otherwise} \end{cases}$

Solution:

1. $n - i$
2. 1
3. $2 \cdot 1$
4. We want the level i at which $n - i = 0$, so we want level n.
5. $1 \cdot 1$
6. $\sum_{i=0}^{n-1} 2 + 1$
7. $2n + 1$
8. Master Theorem does not apply here :/

c) $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 3T(n/3) + n & \text{otherwise} \end{cases}$

Solution:

1. $\frac{n}{3^i}$
2. 3^i
3. $\frac{n}{3^i} \cdot 3^i = n$
4. The level i at which $\frac{n}{3^i} = 1$ i.e. $\log_3(n)$.
5. $3^{\log_3(n)} \cdot 1 = n$
6. $\sum_{i=0}^{\log_3(n)-1} n + n$
7. $n \log_3(n) + n$
8. $\log_3(3) = 1$, so Master Theorem says $\Theta(n \log n)$, which is consistent with our answer.

d) $T(n) = \begin{cases} 1 & \text{if } n = 3 \\ 2T(n/3) + n & \text{otherwise} \end{cases}$

Solution:

1. $\frac{n}{3^i}$
2. 2^i
3. $2^i \frac{n}{3^i} = n\left(\frac{2}{3}\right)^i$
4. The level i such that $\frac{n}{3^i} = 3$ so $\log_3(n) - 1$ is the last level.
5. $2^{\log_3(n)-1} \cdot 1$
6. $\sum_{i=0}^{\log_3(n)-2} n\left(\frac{2}{3}\right)^i + 2^{\log_3(n)-1}$
7. Applying the finite geometric series formula we get:

$$n\frac{\left(\frac{2}{3}\right)^{\log_3(n)-1} - 1}{\frac{2}{3} - 1} + 2^{\log_3(n)-1}$$

   Looks pretty ugly, but it's a closed form. So we'll stop here.

8. $\log_3(2) < 1$ so we should get $\Theta(n)$. The crazy formula from the last part is actually $\Theta(n)$. The second term is on the order of $n^{\log_3(2)}$, which is asymptotically less than n (since $\log_3(2) < 1$) In the first term, the denominator is negative, so it's really $n\left(c - c'\left(\frac{2}{3}\right)^{\log_3(n)-1}\right)$ (where c, c' are constants. As n gets larger, $\left(\frac{2}{3}\right)^{\log_3(n)-1}$ is getting smaller. So cn really is the dominating term.

e) $T(n) = \begin{cases} 2 & \text{if } n = 4 \\ 4T(n/2) + n^2 & \text{otherwise} \end{cases}$

Solution:

1. $\frac{n}{2^i}$
2. 4^i
3. $4^i \cdot \left(\frac{n}{2^i}\right)^2 = n^2$.
4. We want the level i where $4 = \frac{n}{2^i}$, so we want level $\log_2(n) - 2$.
5. $4^{\log_2(n)-2} \cdot 2 = 2^{2\log_2(n)-3} = \frac{n^2}{8}$
6. $\sum_{i=0}^{\log_2(n)-3} n^2 + \frac{n^2}{8}$
7. $(\log_2(n) - 2)n^2 + \frac{n^2}{8}$
8. $\log_2(4) = 2$, so Master Theorem predicts $\Theta(n^2 \log n)$, which matches our answer.

2 Writing Recurrences

Answer the following questions about these pseudocode snippets. In cases where you are describing the running time, you should describe the non-recursive work using a simple function. For example, if the total number of non-recursive operations was $2n + 4$ you can describe this as just $c \cdot n$ (where c is a constant, and we ignore the lower-order term).

a)

```
function F(n)
    if n == 0 then
        return 1
    else
        return 2 · F(n − 1) + 1
    end if
end function
```

- Write a recurrence to describe the output of the function.

Solution:

$$P(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2P(n-1) + 1 & \text{otherwise} \end{cases}$$

- Write a recurrence to describe the running time of the function.

Solution:

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

where c_1, c_2 are constants.

b)

```
function F(n)
    if n == 0 then
        return 0
    end if
    result ← 0
    for i from 0 to n − 1 do
        for j from 0 to i do
            result ← result +j
        end for
    end for
    return F(n/2) + result + F(n/2)
end function
```

- Write a recurrence to model the running time of this function.

Solution:

$$T(n) = \begin{cases} c_1 & \text{if } n = 0 \\ 2T(n/2) + c_2 n^2 & \text{otherwise} \end{cases}$$

Where c_1, c_2 are constants.

- Find a Big-Θ bound on the running time.

Solution: Applying the master theorem, since $\log_2(2) < 2$ is $\Theta(n^2)$. We could also solve the recurrence with the tree method to get this bound.

c)
```
function G(n)
    if n ≤ 1 then
        return 1000
    end if
    if G(n/3) > 5 then
        for i from 0 to n − 1 do
            print "YAY!"
        end for
        return 5 · G(n/3)
    else
        for i from 0 to n^2 − 1 do
            print "YAY!"
        end for
        return 4 · G(n/3)
    end if
end function
```

- For what values of n do we reach the "else" branch?

> **Solution:** No values of n use the else branch. It was a trick – the else branch is actually dead code.

- Write a recurrence to describe the worst case running time of `G`.

> **Solution:**
> $$T(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ 2T(n/3) + cn & \text{otherwise} \end{cases}$$
> Where c is some constant.

- Find a big-Θ bound on the running time by solving the recurrence.

> **Solution:** We actually did almost exactly this in Section 1 (part d), with just n instead of cn, and hitting the base case a little earlier. The small difference in the definition doesn't affect the $\Theta(n)$ bound we got there.