4.1 A brief overview on writing recurrence relations

Here is an example of recurrence relation:

$$f(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ f(n-1) + f(n-2) & n > 1 \end{cases}$$

Let's find the value of $f(10)$,
In order to calculate $f(10)$, we need $f(9)$ and $f(8)$.

n	f(n-2)	f(n-1)	f(n) = f(n-2) + f(n-1)
2	1	1	2
3	1	2	3
4	2	3	5
5	3	5	8
6	5	8	13
7	8	13	21
..	..	..	..

A recurrence relation defines the succeeding terms based on the preceding values of the sequence. For instance, in above example, we calculate $f(5)$ based on $f(4)$ and $f(3)$. You can immediately see that there is no immediate values to both $f(4)$ and $f(3)$ as they rely on $f(2)$ and $f(1)$. Thus, there has to be a base case or base cases from where, the rest of the sequences can be built.

4.2 Recurrence runtime

For a recursive procedure/function, the runtime needs to be defined with a recurrence relation. Let's look at the procedure of calculating the sum recursively from our previous lecture.

```
SUM($A, n$)
1 if $n = 1$
2     return $A[n]$
3 else
4     return $A[n]$ + SUM($A, n - 1$)
```

As we go over line by line, we can see that line 1, 2, & 3 takes constant time, however, in line 4, the procedure itself has been called with $n - 1$ input size. Therefore, we can define the runtime for the else block as $T(n-1) + c_2$. Summing it up:

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

Similarly for the following procedure,

```
FIB(n)
1 if $n \le 1$
2     return 1
3 else
4     return FIB($n-1$) + FIB($n-2$)
```

The runtime for this procedure will be:

$$T(n) = \begin{cases} c_1 & \text{if } n \le 1, \\ T(n-1) + T(n-2) + c_2 & \text{if } n > 1. \end{cases}$$

Please note that so far (in this lecture note) we are basically ruling out the runtime of the procedures. We are not evaluating the runtime or finding their asymptotic bounds.

4.3 Solving recurrence using substitution (again)

Let's try to solve the recurrence relation for the Fibonacci (Fib) procedure above using substitution method.

The first step of substitution method is to make a (good) guess. If you build the tree of the recursive calls for Fib procedure, you can see a binary tree growing. For node $T(n)$, there will always be two branching, $T(n-1)$ and $T(n-2)$.

Let's guess $T(n) = O(2^n)$ or $T(n) \le c2^n$ for some constant c, which is going to be our inductive hypothesis.

Assuming that the inductive hypothesis holds for all positive $m < n$, we can substitute for $m = n-1$ and $m = n-2$. From the recurrence relation above, $T(n) = T(n-1)+T(n-2)+c_2$, for $m = n-1$ and $m = n-2$,

$$T(n-1) \le c2^{n-1}$$

and

$$T(n-2) \le c2^{n-2}$$

yielding,

$$\begin{aligned} T(n) &\le c2^{n-1} + c2^{n-2} + c_2 \\ &= c2^{n-2}(2+1) + c_2 \\ &\le c2^{n-2}(2+1) + c, \text{ where } c = max(c, c_2) \\ &\le c2^{n-2}(2+2), \text{ for } c > 1 \text{ and } n > 2 \\ &\le c2^n \end{aligned}$$

Therefore, $T(n) = O(2^n)$.

However, we can find a better bound for Fib. In fact $T(n) = O(\varphi)^n$ for Fib procedure, where $\varphi = 1.618...$, also known as the golden ratio. Our upper bound, $O(2^n)$ still holds *though. It's just not a better one.*

4.4 Classwork

Exercise 1. Show that $T(n) = T(n-1) + n^2$ is $T(n) = O(n^3)$ using substitution method.

4.5 Solving recurrence using a tree

The recursion tree for $T(n) = 2T(n/2) + cn$ can be expressed as shown in Fig 2.5.

Let's try to unfold $T(n)$

$$T(n) = 2T(n/2) + cn \tag{1}$$

$$T(n/2) = 2T(n/4) + cn \tag{2}$$

$$T(n/4) = 2T(n/8) + cn \tag{3}$$

When we replace $T(n/2)$ in eq (1), we get.

$$T(n) = 2(2T(n/4) + cn) + cn = 4T(n/4) + 2cn + cn$$

When we replace $T(n/4)$ from eq (3) to above derivation, we get

$$T(n) = 4(2T(n/8) + cn) + 2cn + cn = 8T(n/8) + 4cn + 2cn + cn$$

The same process of unfolding the recurrence relation can be expressed as a recursion tree.

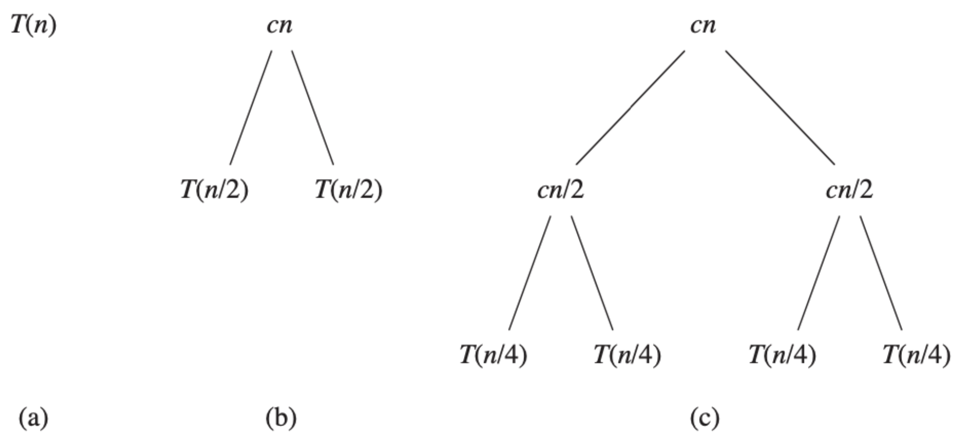
As you are sketching the recursion tree, we need to find (i) how many levels of recursion we go through and (ii) what is the cost at each level.

Let's do a run through the tree shown in Fig. 25 with a value of $n = 8$.

Level 0, $cn = 8c$
Level 1, $c8/2 + c8/2 = 4c + 4c = 8c$
Level 2, $2c * 4 = 8c$
Level 3, $c * 8 = 8c$

Therefore, for $n = 8$, there were $\lg(8) + 1 = 3 + 1 = 4$ levels. At each level, there were same number of operations $8c$. Therefore, $n = 8$, the total runtime was $(8c) * (\lg(8) + 1) = 8c\lg(8) + 8c$

We can generalize this with n as $T(n) = cn\lg(n) + nc = c(n\lg n) = \Theta(n\lg n)$



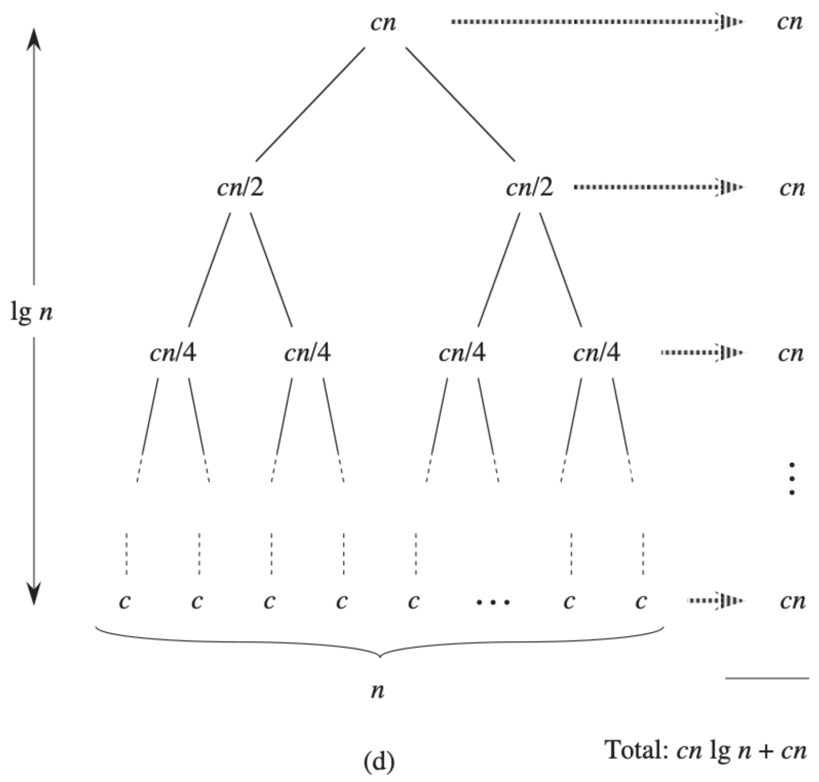


Figure 2.5 How to construct a recursion tree for the recurrence $T(n) = 2T(n/2) + cn$. Part **(a)** shows $T(n)$, which progressively expands in **(b)–(d)** to form the recursion tree. The fully expanded tree in part (d) has $\lg n + 1$ levels (i.e., it has height $\lg n$, as indicated), and each level contributes a total cost of cn. The total cost, therefore, is $cn \lg n + cn$, which is $\Theta(n \lg n)$.

4.6 Solving recurrence relations using master theorem

The master method provides a "cookbook" method for solving recurrences of the form:

$$T(n) = aT(n/b) + f(n)$$

,

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function. To use the master method, we have to remember three cases:

Theorem 1. Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n)$$

,

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Example 1. $T(n) = 9T(n/3) + n$.

We have $a = 9$, $b = 3$, $f(n) = n$. Now, we calculate $n^{\log_3 9} = n^{\log_3 3^2} = n^2 = \Theta(n^2)$. Now, if we can show $f(n) = O(n^{\log_3 9 - \epsilon})$ for $\epsilon > 0$, then $T(n) = \Theta(n^{\log_3 9})$. For $\epsilon = 1$, we can see that $n^{\log_3 9 - 1} = n = f(n)$, and $f(n) = n = O(n^{\log_3 9 - 1})$, therefore, by case 1, $T(n) = \Theta(n^{\log_3 9}) = \Theta(n^2)$.

Example 2. $T(n) = T(2n/3) + 1$.

Here, $a = 1$, $b = 3/2$, and $f(n) = 1$. $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1 = \Theta(1)$. And we have $f(n) = 1$, therefore we should go with case 2. By case 2, the $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(n^{\log_{3/2} 1} \lg n) = \Theta(\lg n)$.

Example 3. $T(n) = 3T(n/4) + n \lg n$.

We have $a = 3$, $b = 4$, $f(n) = n \lg n$, and $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$. For $\epsilon \approx 0.2$, we get $f(n) = \Omega(n^{\log_4 3 + \epsilon})$. In this case, case 3 will applies if we can show that $af(n/b) \leq cf(n)$. For a sufficiently large value of n, we have $af(n/b) = 3(n/4)\lg(n/4) \leq (3/4)n \lg n = cf(n)$ for $c = 3/4$. Therefore, by case 3, the solution to the recurrence is $T(n) = \Theta(n \lg n)$.