Solving Recurrences

In last two lectures, we came across recurrence relations.

Generally they reflect the running time of recursive algorithms.

" Here solving the recurrence relation means finding the closed form expression in terms of n"

In this lecture

We will learn how to solve recurrences.

There are three well known methods to solve recurrences.

1. Substitution method
2. Recursive-tree method
3. Master method.

① $T(n)$ denote the maximum number of operations an algorithm performs on input of size n.

② **Recurrence:** Express $T(n)$ in terms of running time of recursive calls.

ⓐ Base case [E.g. $T(1) = \text{Constant}$]

ⓑ $$T(n) = a\,T(n/b) + f(n)$$

cost of recursive calls — outside of recursive calls

Substitution method :

This method comprises two steps

1. Guess the form of the solution
2. Use Induction to find the constants and show that the solution works.

(1) $T(n) = 2T(n-1) + 1$

$T(0) = 0$

Guess: $T(n) = 2^n - 1$

$T(0) = 2^0 - 1 = 0$ [Base Case]

$$
\begin{aligned}
T(n) &= 2T(n-1) + 1 \\
&= 2(2^{n-1} - 1) + 1 \quad \text{[induction hypothesis]} \\
&= 2^n - 1
\end{aligned}
$$

Example 1 : $T(n) = 2\,T(\lfloor \frac{n}{2} \rfloor) + n$ —— ① and $T(1) = 1$

Guess is $T(n) = O(n\log n)$

We have to show $T(n) \le c\, n\log n$ for some constant $c > 0$ & $n \geq n_0$.

Assume that for all $m < n$, $T(m) \le c\, m\log m$.

In Particular for $m = \frac{n}{2}$, we get

$$T(\tfrac{n}{2}) \le c\lfloor \tfrac{n}{2} \rfloor . \log\lfloor \tfrac{n}{2} \rfloor .$$

Substitute this in equation ①

$$\begin{aligned}
T(n) &< 2.\, c\lfloor \tfrac{n}{2} \rfloor \log\lfloor \tfrac{n}{2} \rfloor + n \\
&= cn\log(\tfrac{n}{2}) + n \\
&= cn\log n - cn\log 2 + n \\
&= cn\log n - cn + n \\
&\le cn\log n \quad (\text{when } c \geq 1)
\end{aligned}$$

Boundary condition: Suppose assume $T(1) = 1$ then

$$T(1) = c \cdot 1 \cdot \log 1 = 0$$

So for $n = 1$ our inductive proof fails to hold.

We can handle this case by selecting n_0 carefully, because we only need to show

$$T(n) \leq c\, n \log n \text{ for } n \geq n_0$$

$T(2) = 4$ and $T(3) = 5$ [from the recurrence]

So, we select $n_0 = 2$ and $c \geq 2$, hence we get

$$T(n) \leq c\, n \log n \text{ for some } c \geq 2 \text{ and } n_0 \geq 2$$

Disadvantage of Substitution method:

There is no general way to guess the solution to recurrences.

Many divide and conquer algorithms give us running-time recurrences of the form

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

$$T(1) = \Theta(1)$$

Where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is some other function.

$$T(n) = \underbrace{a\,T\left(\frac{n}{b}\right)}_{\text{'a' recursive calls, each on a problem of size } \frac{n}{b}} + \underset{\substack{\downarrow \\ \text{Non-recursive work}}}{f(n)}$$

Master Method :

Master method is used to solve recurrences of the form $T(n) = \text{Constant}$ for small n [Base case]

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n) \quad \text{—①}$$

[All Subproblems have Same Size]

where $a \geq 1$, $b > 1$ are constants and $f(n)$ is an asymptotically positive function.

Recurrence ① describes the running time of an algorithm that divides a problem of size n into 'a' subproblems, each of size $\frac{n}{b}$, where $a \geq 1$, $b > 1$. function $f(n)$ is the cost of dividing the problem and combining the results of the subproblems.

The master theorem

Let $a \geq 1$, $b > 1$ be constants, let $f(n)$ be a function and for $n \in \mathbb{Z}^+$, let $T(n)$ be defined by recurrence

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

then $T(n)$ has the following asymptotic bounds.

① If $f(n) = O(n^{\log_b^a - \varepsilon})$ for some constant $\varepsilon > 0$,
then $\boxed{T(n) = \Theta(n^{\log_b^a})}$

② If $f(n) = \Theta(n^{\log_b^a})$, then $\boxed{T(n) = \Theta(n^{\log_b^a} \log n)}$

③ If $f(n) = \Omega(n^{\log_b^a + \varepsilon})$ for some constant $\varepsilon > 0$,
and if $\underbrace{a f(n/b) \leq c f(n)}_{\text{Regularity Condition}}$ for some constant $c < 1$
and all sufficiently large n, then $\boxed{T(n) = \Theta(f(n))}$.

* " $\frac{n}{b}$ " can be either $\left\lfloor \frac{n}{b} \right\rfloor$ or $\left\lceil \frac{n}{b} \right\rceil$ also.

In each of the three cases, we compare the function $f(n)$ with the function $n^{\log_b^a}$.

Intuitevely the larger of the two functions determines the solution to the recurrence.

In Case 1, function $n^{\log_b^a}$ is larger, the the solution is $\Theta(n^{\log_b^a})$

In Case 2, two functions are the same size, we multiply by a logarithmic factor and the soln is $\Theta(n^{\log_b^a} \log n)$

In Case 3, the function $f(n)$ is larger, then the solution is $\Theta(f(n))$

Examples

① $T(n) = 4T(\frac{n}{2}) + n$

$a = 4, \quad b = 2, \quad f(n) = n$

$n^{\log_b a} = n^2$

$f(n) = O(n^{2-\varepsilon})$ for $\varepsilon = 1$

By case 1 of master theorem $T(n) = \Theta(n^2)$

② $T(n) = 4T(\frac{n}{2}) + n^2$

$a = 4, \quad b = 2, \quad f(n) = n^2$

$n^{\log_b a} = n^2$

$f(n) = \Theta(n^2)$,

By case 2 of master theorem $T(n) = \Theta(n^2 \log n)$

③ $T(n) = 4T(n/2) + n^3$

$a = 4, \quad b = 2, \quad f(n) = n^3$

$n^{\log_b^a} = n^2$

$f(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon = 1$

and $4\left(\frac{n}{2}\right)^3 = \frac{n^3}{2} \leq c \cdot n^3$ for $c = \frac{1}{2}$

By case 3 of master theorem $T(n) = \Theta(n^3)$

④ $T(n) = 2T(n/2) + n\log n$

$a = 2, \quad b = 2, \quad f(n) = n\log n$

$n^{\log_b^a} = n;$

$f(n) = \Omega(n)$, but $f(n) \neq \Omega(n^{1+\varepsilon})$ for any $\varepsilon > 0$.

that is $f(n)$ is not Polynomially larger.

$\therefore$ Master theorem is NOT applicable.

⑤ $T(n) = 3T(n/4) + n\log n$

$a=3, \quad b=4, \quad f(n) = n\log n$

$n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$

$f(n) = \Omega(n^{\log_4 3 + \varepsilon})$, where $\varepsilon \approx 0.2$.

$a f(n/b) = 3\left(\frac{n}{4}\right)\log(n/4) \le \frac{3}{4} n\log n = cf(n)$

for $c = \frac{3}{4}$, for large n

By case 3, $T(n) = \Theta(n\log n)$

Practice questions

① $T(n) = 8T(n/2) + \Theta(n^2)$

② $T(n) = 7T(n/2) + \Theta(n^2)$

③ $T(n) = 2T(n/2) + \Theta(n)$

④ $T(n) = 2T(n/4) + \sqrt{n}$

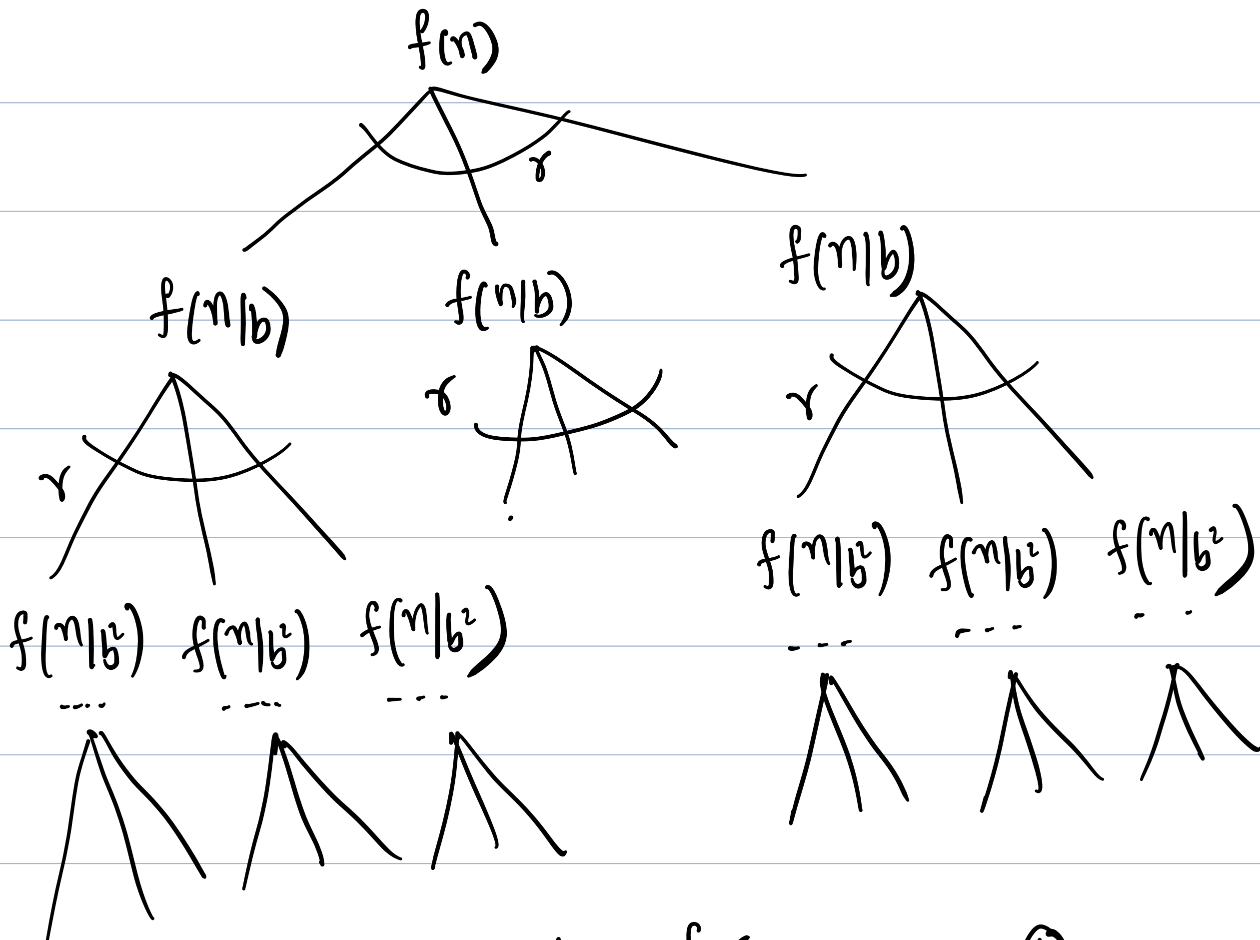
The recursion tree-method

Recursion trees are pictorial tool for solving divide and conquer recurrences.

A recursive tree is a rooted tree with one node for each recursive subproblem. The value of each node is the amount of time spent on the corresponding subproblem excluding recursive calls.

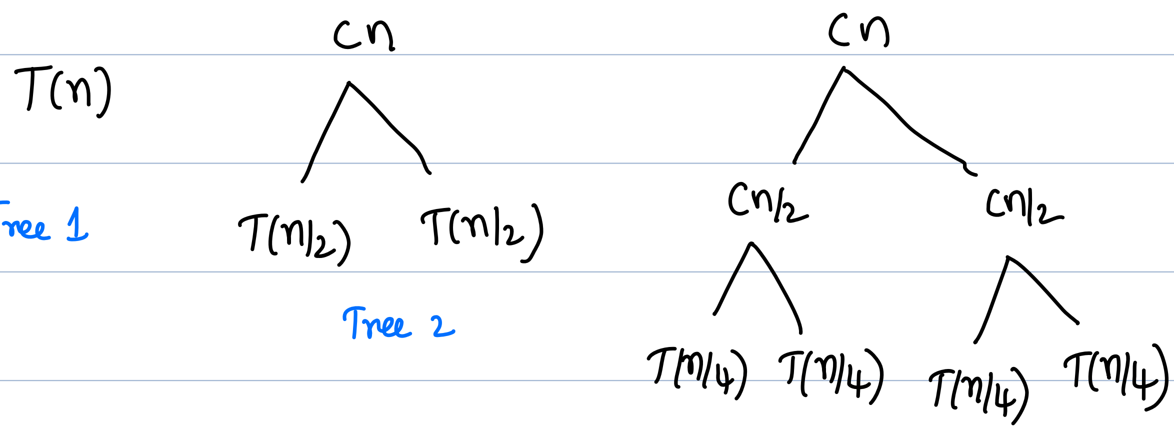
The overall running time of the algorithm is the sum of the values of all nodes in the tree.

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \quad -①$$



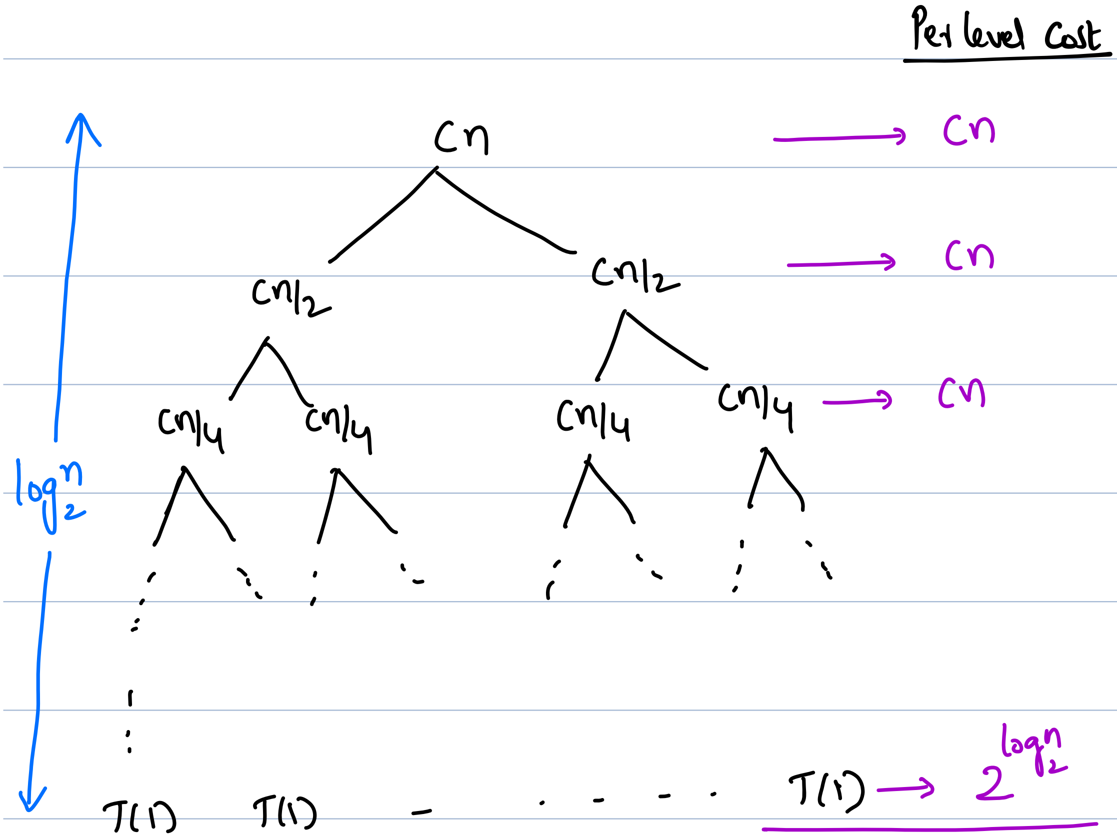
Recursion tree for recurrence ①

Eg: $T(n) = 2T(n/2) + cn$ (Same as previous example)



Tree 1 Tree 2 Tree 3.

We continue expanding each node in the tree, by using recurrence, we get the following tree.



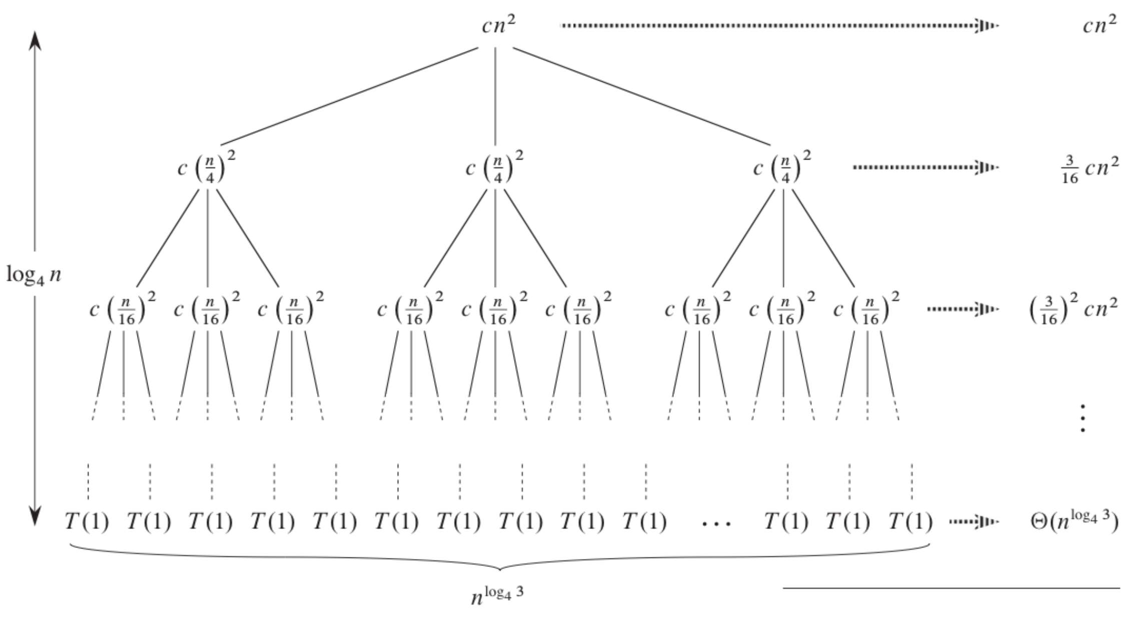
Total : $\log_2^n cn + 2^{\log_2^n}$

$= cn\log n + n$

$= \Theta(n\log n)$

Eg ② $T(n) = 3T(n/4) + cn^2$

Final recursion tree looks as follows



(d)

$$\text{Total cost} = cn^2 + \frac{3}{16}cn^2 + \cdots + \left(\frac{3}{16}\right)^{\log_4^n - 1} + \Theta(n^{\log_4^3})$$

$$= \sum_{i=0}^{\log_4^n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4^3})$$

$$< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4^3})$$

$$= \frac{1}{1-\frac{3}{16}} cn^2 + \Theta(n^{\log_4 3})$$

$$= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3})$$

$$= O(n^2)$$

Practice Problems

① $T(n) = T(n/3) + T(2n/3) + cn$

② $T(n) = 2T(n-1) + 1$