Recurrences

(CLRS 4.1-4.2)

- Last time we discussed divide-and-conquer algorithms

> **Divide and Conquer**
>
> To Solve P:
>
> 1. *Divide* P into smaller problems $P_1, P_2, P_3.....P_k$.
> 2. *Conquer* by solving the (smaller) subproblems recursively.
> 3. *Combine* solutions to $P_1, P_2, ...P_k$ into solution for P.

- Analysis of divide-and-conquer algorithms and in general of recursive algorithms leads to recurrences.

- Merge-sort lead to the recurrence $T(n) = 2T(n/2) + n$
  - or rather, $T(n) = \begin{cases} \Theta(1) & \text{If } n = 1 \\ T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + \Theta(n) & \text{If } n > 1 \end{cases}$
  - but we will often cheat and just solve the simple formula (equivalent to assuming that $n = 2^k$ for some constant k, and leaving out base case and constant in Θ).

Methods for solving recurrences

1. Substitution method
2. Iteration method
   - Recursion-tree method
   - (Master method)

1 Solving Recurrences with the Substitution Method

- Idea: Make a guess for the form of the solution and prove by induction.
- Can be used to prove both upper bounds $O()$ and lower bounds $\Omega()$.
- Let's solve $T(n) = 2T(n/2) + n$ using substitution
  - Guess $T(n) \leq cn \log n$ for some constant c (that is, $T(n) = O(n \log n)$)
  - Proof:
    * Base case: we need to show that our guess holds for some base case (not necessarily $n = 1$, some small n is ok). Ok, since function constant for small constant n.
    * Assume holds for $n/2$: $T(n/2) \leq c\frac{n}{2} \log \frac{n}{2}$ (Question: Why not $n-1$?)
      Prove that holds for n: $T(n) \leq cn \log n$

$$
\begin{aligned}
T(n) &= 2T(n/2) + n \\
&\leq 2(c\frac{n}{2} \log \frac{n}{2}) + n \\
&= cn \log \frac{n}{2} + n \\
&= cn \log n - cn \log 2 + n \\
&= cn \log n - cn + n
\end{aligned}
$$

      So ok if $c \geq 1$

- Similarly it can be shown that $T(n) = \Omega(n \log n)$

  Exercise!
- Similarly it can be shown that $T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + n$ is $\Theta(n \lg n)$.

  Exercise!
- The hard part of the substitution method is often to make a good guess. How do we make a good (i.e. tight) guess??? Unfortunately, there's no "recipe" for this one. Try iteratively $O(n^3), \Omega(n^3), O(n^2), \Omega(n^2)$ and so on. Try solving by iteration to get a feeling of the growth.

2 Solving Recurrences with the Iteration/Recursion-tree Method

- In the iteration method we iteratively "unfold" the recurrence until we "see the pattern".
- The iteration method does not require making a good guess like the substitution method (but it is often more involved than using induction).
- Example: Solve $T(n) = 8T(n/2) + n^2 \quad (T(1) = 1)$

$$
\begin{aligned}
T(n) &= n^2 + 8T(n/2) \\
&= n^2 + 8(8T(\frac{n}{2^2}) + (\frac{n}{2})^2) \\
&= n^2 + 8^2T(\frac{n}{2^2}) + 8(\frac{n^2}{4})) \\
&= n^2 + 2n^2 + 8^2T(\frac{n}{2^2}) \\
&= n^2 + 2n^2 + 8^2(8T(\frac{n}{2^3}) + (\frac{n}{2^2})^2) \\
&= n^2 + 2n^2 + 8^3T(\frac{n}{2^3}) + 8^2(\frac{n^2}{4^2})) \\
&= n^2 + 2n^2 + 2^2n^2 + 8^3T(\frac{n}{2^3}) \\
&= \ldots \\
&= n^2 + 2n^2 + 2^2n^2 + 2^3n^2 + 2^4n^2 + \ldots
\end{aligned}
$$

  - Recursion depth: How long (how many iterations) it takes until the subproblem has constant size? i times where $\frac{n}{2^i} = 1 \Rightarrow i = \log n$
  - What is the last term? $8^iT(1) = 8^{\log n}$

$$
\begin{aligned}
T(n) &= n^2 + 2n^2 + 2^2n^2 + 2^3n^2 + 2^4n^2 + \ldots + 2^{\log n - 1}n^2 + 8^{\log n} \\
&= \sum_{k=0}^{\log n - 1} 2^k n^2 + 8^{\log n} \\
&= n^2 \sum_{k=0}^{\log n - 1} 2^k + (2^3)^{\log n}
\end{aligned}
$$

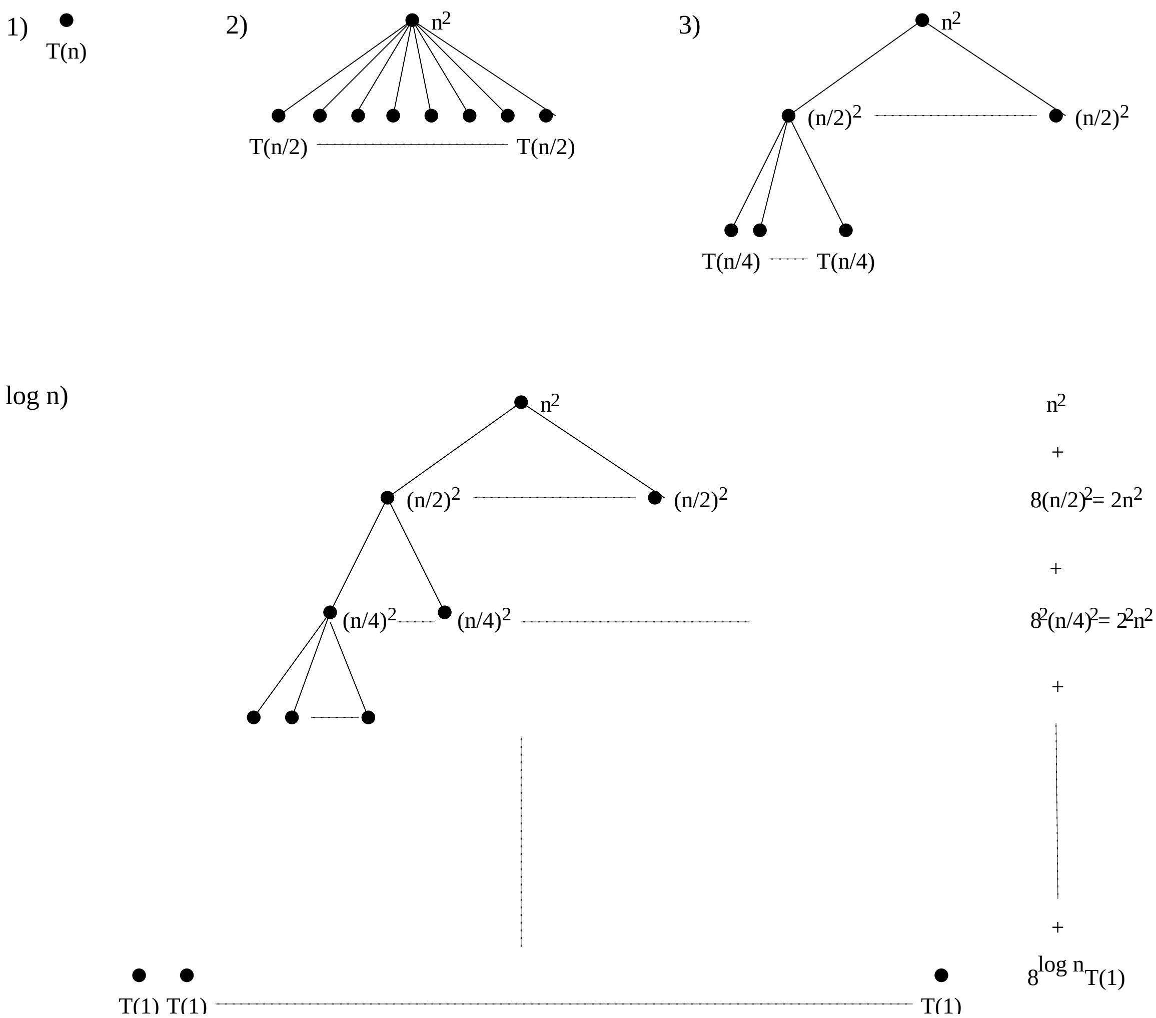
- Now $\sum_{k=0}^{\log n-1} 2^k$ is a geometric sum so we have $\sum_{k=0}^{\log n-1} 2^k = \Theta(2^{\log n-1}) = \Theta(n)$
- $(2^3)^{\log n} = (2^{\log n})^3 = n^3$

$$
\begin{aligned}
T(n) &= n^2 \cdot \Theta(n) + n^3 \\
&= \Theta(n^3)
\end{aligned}
$$

2.1 Recursion tree

A different way to look at the iteration method: is the recursion-tree, discussed in the book (4.2).

- we draw out the recursion tree with cost of single call in each node—running time is sum of costs in all nodes
- if you are careful drawing the recursion tree and summing up the costs, the recursion tree is a direct proof for the solution of the recurrence, just like iteration and substitution
- Example: $T(n) = 8T(n/2) + n^2 \ \ (T(1) = 1)$



3 Matrix Multiplication

- Let X and Y be $n \times n$ matrices

$$X = \left\{ \begin{array}{llll} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{1n} \\ x_{31} & x_{32} & \cdots & x_{1n} \\ \cdots & \cdots & \cdots & \cdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{array} \right\}$$

- We want to compute $Z = X \cdot Y$
  - $z_{ij} = \sum_{k=1}^{n} X_{ik} \cdot Y_{kj}$
- Naive method uses $\Rightarrow n^2 \cdot n = \Theta(n^3)$ operations
- Divide-and-conquer solution:

$$Z = \left\{ \begin{array}{cc} A & B \\ C & D \end{array} \right\} \cdot \left\{ \begin{array}{cc} E & F \\ G & H \end{array} \right\} = \left\{ \begin{array}{cc} (A \cdot E + B \cdot G) & (A \cdot F + B \cdot H) \\ (C \cdot E + D \cdot G) & (C \cdot F + D \cdot H) \end{array} \right\}$$

  - The above naturally leads to divide-and-conquer solution:
    * Divide X and Y into 8 sub-matrices A, B, C, and D.
    * Do 8 matrix multiplications recursively.
    * Compute Z by combining results (doing 4 matrix additions).
  - Lets assume $n = 2^c$ for some constant c and let A, B, C and D be $n/2 \times n/2$ matrices
    * Running time of algorithm is $T(n) = 8T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^3)$
  - But we already discussed a (simpler/naive) $O(n^3)$ algorithm! Can we do better?

3.1 Strassen's Algorithm

- Strassen observed the following:

$$Z = \left\{ \begin{array}{cc} A & B \\ C & D \end{array} \right\} \cdot \left\{ \begin{array}{cc} E & F \\ G & H \end{array} \right\} = \left\{ \begin{array}{cc} (S_1 + S_2 - S_4 + S_6) & (S_4 + S_5) \\ (S_6 + S_7) & (S_2 + S_3 + S_5 - S_7) \end{array} \right\}$$

where

$$\begin{aligned} S_1 &= (B - D) \cdot (G + H) \\ S_2 &= (A + D) \cdot (E + H) \\ S_3 &= (A - C) \cdot (E + F) \\ S_4 &= (A + B) \cdot H \\ S_5 &= A \cdot (F - H) \\ S_6 &= D \cdot (G - E) \\ S_7 &= (C + D) \cdot E \end{aligned}$$

– Lets test that $S_6 + S_7$ is really $C \cdot E + D \cdot G$

$$\begin{aligned} S_6 + S_7 &= D \cdot (G - E) + (C + D) \cdot E \\ &= DG - DE + CE + DE \\ &= DG + CE \end{aligned}$$

- This leads to a divide-and-conquer algorithm with running time $T(n) = 7T(n/2) + \Theta(n^2)$
  – We only need to perform 7 multiplications recursively.
  – Division/Combination can still be performed in $\Theta(n^2)$ time.
- Lets solve the recurrence using the iteration method

$$\begin{aligned} T(n) &= 7T(n/2) + n^2 \\ &= n^2 + 7(7T(\frac{n}{2^2}) + (\frac{n}{2})^2) \\ &= n^2 + (\frac{7}{2^2})n^2 + 7^2T(\frac{n}{2^2}) \\ &= n^2 + (\frac{7}{2^2})n^2 + 7^2(7T(\frac{n}{2^3}) + (\frac{n}{2^2})^2) \\ &= n^2 + (\frac{7}{2^2})n^2 + (\frac{7}{2^2})^2 \cdot n^2 + 7^3T(\frac{n}{2^3}) \\ &= n^2 + (\frac{7}{2^2})n^2 + (\frac{7}{2^2})^2n^2 + (\frac{7}{2^2})^3n^2 .... + (\frac{7}{2^2})^{\log n - 1}n^2 + 7^{\log n} \\ &= \sum_{i=0}^{\log n - 1} (\frac{7}{2^2})^i n^2 + 7^{\log n} \\ &= n^2 \cdot \Theta((\frac{7}{2^2})^{\log n - 1}) + 7^{\log n} \\ &= n^2 \cdot \Theta(\frac{7^{\log n}}{(2^2)^{\log n}}) + 7^{\log n} \\ &= n^2 \cdot \Theta(\frac{7^{\log n}}{n^2}) + 7^{\log n} \\ &= \Theta(7^{\log n}) \end{aligned}$$

  – Now we have the following:

$$\begin{aligned} 7^{\log n} &= 7^{\frac{\log_7 n}{\log_7 2}} \\ &= (7^{\log_7 n})^{(1/\log_7 2)} \\ &= n^{(1/\log_7 2)} \\ &= n^{\frac{\log_2 7}{\log_2 2}} \\ &= n^{\log 7} \end{aligned}$$

  – Or in general: $a^{\log_k n} = n^{\log_k a}$

So the solution is $T(n) = \Theta(n^{\log 7}) = \Theta(n^{2.81\dots})$

- Note:
  - We are 'hiding' a much bigger constant in $\Theta()$ than before.
  - Currently best known bound is $O(n^{2.376..})$ (another method).
  - Lower bound is (trivially) $\Omega(n^2)$.

4 Master Method

- We have solved several recurrences using *substitution* and *iteration*.
- we solved several recurrences of the form $T(n) = aT(n/b) + n^c \quad (T(1) = 1)$.
  - Strassen's algorithm $\Rightarrow T(n) = 7T(n/2) + n^2$ ($a = 7, b = 2$, and $c = 2$)
  - Merge-sort $\Rightarrow T(n) = 2T(n/2) + n$ ($a = 2, b = 2$, and $c = 1$).
- It would be nice to have a general solution to the recurrence $T(n) = aT(n/b) + n^c$.
- We do!

$$\boxed{\begin{array}{l} T(n) = aT\left(\frac{n}{b}\right) + n^c \quad a \geq 1, b \geq 1, c > 0 \\ \Downarrow \\ T(n) = \begin{cases} \Theta(n^{\log_b a}) & a > b^c \\ \Theta(n^c \log_b n) & a = b^c \\ \Theta(n^c) & a < b^c \end{cases} \end{array}}$$

Proof (Iteration method)

$$\begin{aligned}
T(n) &= aT\left(\tfrac{n}{b}\right) + n^c \\
&= n^c + a\left(\left(\tfrac{n}{b}\right)^c + aT\left(\tfrac{n}{b^2}\right)\right) \\
&= n^c + \left(\tfrac{a}{b^c}\right)n^c + a^2T\left(\tfrac{n}{b^2}\right) \\
&= n^c + \left(\tfrac{a}{b^c}\right)n^c + a^2\left(\left(\tfrac{n}{b^2}\right)^c + aT\left(\tfrac{n}{b^3}\right)\right) \\
&= n^c + \left(\tfrac{a}{b^c}\right)n^c + \left(\tfrac{a}{b^c}\right)^2 n^c + a^3T\left(\tfrac{n}{b^3}\right) \\
&= \dots \\
&= n^c + \left(\tfrac{a}{b^c}\right)n^c + \left(\tfrac{a}{b^c}\right)^2 n^c + \left(\tfrac{a}{b^c}\right)^3 n^c + \left(\tfrac{a}{b^c}\right)^4 n^c + \dots + \left(\tfrac{a}{b^c}\right)^{\log_b n - 1} n^c + a^{\log_b n}T(1) \\
&= n^c \textstyle\sum_{k=0}^{\log_b n - 1} \left(\tfrac{a}{b^c}\right)^k + a^{\log_b n} \\
&= n^c \textstyle\sum_{k=0}^{\log_b n - 1} \left(\tfrac{a}{b^c}\right)^k + n^{\log_b a}
\end{aligned}$$

Recall geometric sum $\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1} = \Theta(x^n)$

- $\boxed{a < b^c}$

  $a < b^c \Leftrightarrow \frac{a}{b^c} < 1 \Rightarrow \sum_{k=0}^{\log_b n-1} \left(\frac{a}{b^c}\right)^k \leq \sum_{k=0}^{+\infty} \left(\frac{a}{b^c}\right)^k = \frac{1}{1-\left(\frac{a}{b^c}\right)} = \Theta(1)$

  $a < b^c \Leftrightarrow \log_b a < \log_b b^c = c$

  $$\begin{aligned} T(n) &= n^c \sum_{k=0}^{\log_b n-1} \left(\frac{a}{b^c}\right)^k + n^{\log_b a} \\ &= n^c \cdot \Theta(1) + n^{\log_b a} \\ &= \Theta(n^c) \end{aligned}$$

- $\boxed{a = b^c}$

  $a = b^c \Leftrightarrow \frac{a}{b^c} = 1 \Rightarrow \sum_{k=0}^{\log_b n-1} \left(\frac{a}{b^c}\right)^k = \sum_{k=0}^{\log_b n-1} 1 = \Theta(\log_b n)$

  $a = b^c \Leftrightarrow \log_b a = \log_b b^c = c$

  $$\begin{aligned} T(n) &= \sum_{k=0}^{\log_b n-1} \left(\frac{a}{b^c}\right)^k + n^{\log_b a} \\ &= n^c \Theta(\log_b n) + n^{\log_b a} \\ &= \Theta(n^c \log_b n) \end{aligned}$$

- $\boxed{a > b^c}$

  $a > b^c \Leftrightarrow \frac{a}{b^c} > 1 \Rightarrow \sum_{k=0}^{\log_b n-1} \left(\frac{a}{b^c}\right)^k = \Theta\left(\left(\frac{a}{b^c}\right)^{\log_b n}\right) = \Theta\left(\frac{a^{\log_b n}}{(b^c)^{\log_b n}}\right) = \Theta\left(\frac{a^{\log_b n}}{n^c}\right)$

  $$\begin{aligned} T(n) &= n^c \cdot \Theta\left(\frac{a^{\log_b n}}{n^c}\right) + n^{\log_b a} \\ &= \Theta(n^{\log_b a}) + n^{\log_b a} \\ &= \Theta(n^{\log_b a}) \end{aligned}$$

- Note: Book states and proves the result slightly differently (don't read it).

5 Changing variables

Sometimes reucurrences can be reduced to simpler ones by *changing variables*

- Example: Solve $T(n) = 2T(\sqrt{n}) + \log n$

  Let $m = \log n \Rightarrow 2^m = n \Rightarrow \sqrt{n} = 2^{m/2}$

  $T(n) = 2T(\sqrt{n}) + \log n \Rightarrow T(2^m) = 2T(2^{m/2}) + m$

  Let $S(m) = T(2^m)$

  $$\begin{aligned} T(2^m) = 2T(2^{m/2}) + m &\Rightarrow S(m) = 2S(m/2) + m \\ &\Rightarrow S(m) = O(m \log m) \\ &\Rightarrow T(n) = T(2^m) = S(m) = O(m \log m) = O(\log n \log \log n) \end{aligned}$$

6 Other recurrences

Some important/typical bounds on recurrences not covered by master method:

- Logarithmic: $\Theta(\log n)$
  - Recurrence: $T(n) = 1 + T(n/2)$
  - Typical example: Recurse on half the input (and throw half away)
  - Variations: $T(n) = 1 + T(99n/100)$
- Linear: $\Theta(N)$
  - Recurrence: $T(n) = 1 + T(n-1)$
  - Typical example: Single loop
  - Variations: $T(n) = 1 + 2T(n/2), T(n) = n + T(n/2), T(n) = T(n/5) + T(7n/10 + 6) + n$
- Quadratic: $\Theta(n^2)$
  - Recurrence: $T(n) = n + T(n-1)$
  - Typical example: Nested loops
- Exponential: $\Theta(2^n)$
  - Recurrence: $T(n) = 2T(n-1)$