Topic 2: Recurrences and Strassen’s Algorithm
(CLRS 4.0—4.4, 28.1–28.2)

CPS 230, Fall 2001

1 Recurrences

- As we saw previously with divide-and-conquer algorithms, the analysis of recursive algorithms leads to recurrence relations.
- Merge sort leads to the recurrence $T(n) = 2T(n/2) + \Theta(n)$
  
- 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 $k$).

1.1 Substitution method

- Idea: Make good guess and prove by induction.
- Let’s solve $T(n) = 2T(n/2) + n$, $T(1) = 1$ using substitution

  - Guess $T(n) \leq cn\log n$ for some constant $c$, for $n \geq 2$ (that is, $T(n) = O(n\log n)$)
  - Proof:
    * Basis: Function constant for small constant $n$ (e.g., $T(2) = 4 \leq cn\log n$ if $c \geq 2$.
    * Induction:
      Assume holds for $n/2$: $T(n/2) \leq c\frac{n}{2}\log\frac{n}{2}$
      Show holds for $n$: $T(n) \leq cn\log n$
      Proof:

      $T(n) = 2T(n/2) + n$
      \begin{align*}
      \leq & \quad 2 \left( c\frac{n}{2}\log\frac{n}{2} \right) + n \\
      = & \quad cn\log\frac{n}{2} + n \\
      = & \quad cn\log n - cn\log 2 + n \\
      = & \quad cn\log n - cn + n
      \end{align*}$

      So all is fine if $c \geq 1$, since the right-hand side will be at most $cn\log n$.
- $T(n) = \Omega(n\log n)$ can be proved similarly.
- How do we make a good guess?
- Something of an art!
- Try different bounds (e.g. $\Omega(n)$ easy, show $O(n^2) \implies$ guess $O(n \log n)$)

- **Note:** *changing variables* can sometimes help

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

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

    Let $S(m) = T(2^m)$
    
    $T(2^m) = 2T(2^{m/2}) + \log n \implies S(m) = 2S(m/2) + \log n$
    
    $\implies S(m) = O(m \log m)$
    
    $\implies T(n) = T(2^m) = S(m) = O(m \log m) = O(\log n \log \log n)$

### 1.2 Iteration 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 $T(n) = 8T(n/2) + n^2$ ($T(1) = 1$)

    $T(n) = n^2 + 8T(n/2)$
    
    $= n^2 + 8 \left( 8T(\frac{n}{2^2}) + \left( \frac{n}{2} \right)^2 \right)$
    
    $= n^2 + 8^2T \left( \frac{n}{2^2} \right) + 8 \left( \frac{n^2}{4} \right)$
    
    $= n^2 + 2n^2 + 8^2T \left( \frac{n}{2^2} \right)$
    
    $= n^2 + 2n^2 + 8^2 \left( 8T \left( \frac{n}{2^3} \right) + \left( \frac{n}{2^2} \right)^2 \right)$
    
    $= n^2 + 2n^2 + 8^3T \left( \frac{n}{2^3} \right) + 8^2 \left( \frac{n^2}{4^2} \right)$
    
    $= n^2 + 2n^2 + 2^2n^2 + 8^3T \left( \frac{n}{2^3} \right)$
    
    $= \ldots$
    
    $= n^2 + 2n^2 + 2^2n^2 + 2^3n^2 + 2^4n^2 + \ldots$

  - How long does it continue? $i$ times where $\frac{n}{2^i} = 1 \implies i = \log n$

  - What is the last term? $8^i T(1) = 8^{\log n}$

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

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

\[
(2^3)^{\log n} = (2^{\log n})^3 = n^3
\]

\[
T(n) = n^2 \cdot \Theta(n) + n^3 = \Theta(n^3)
\]

2 Matrix Multiplication

- Let \( X \) and \( Y \) be \( n \times n \) matrices

\[
X = \begin{pmatrix}
    x_{11} & x_{12} & \cdots & x_{1n} \\
    x_{21} & x_{22} & \cdots & x_{2n} \\
    \vdots & \vdots & \ddots & \vdots \\
    x_{n1} & x_{n2} & \cdots & x_{nn}
\end{pmatrix}
\]

- We want to compute \( Z = X \cdot Y \)

\[
z_{ij} = \sum_{k=1}^{n} X_{ik} \cdot Y_{kj}
\]

- Naive method uses \( \longrightarrow n^2 \cdot n = \Theta(n^3) \) operations

- Divide-and-conquer solution:

\[
Z = \begin{pmatrix}
    A & B \\
    C & D
\end{pmatrix}
\cdot
\begin{pmatrix}
    E & F \\
    G & H
\end{pmatrix}
= \begin{pmatrix}
    (A \cdot E + B \cdot G) & (A \cdot F + B \cdot H) \\
    (C \cdot E + D \cdot G) & (C \cdot F + D \cdot H)
\end{pmatrix}
\]

- 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).

- Let’s 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 \( T(n) = 8T(n/2) + \Theta(n^2) \iff T(n) = \Theta(n^3) \)
  * But we already discussed a (simpler/naive) \( O(n^3) \) algorithm! Can we do better?

2.1 Strassen’s Algorithm

- Strassen observed the following:

\[
Z = \begin{pmatrix}
    A & B \\
    C & D
\end{pmatrix}
\cdot
\begin{pmatrix}
    E & F \\
    G & H
\end{pmatrix}
= \begin{pmatrix}
    (S_1 + S_2 - S_4 + S_6) & (S_4 + S_5) \\
    (S_0 + S_7) & (S_2 + S_3 + S_5 - S_7)
\end{pmatrix}
\]

where

\[
S_1 = (B - D) \cdot (G + H)
\]
\[
S_2 = (A + D) \cdot (E + H)
\]
\[
S_3 = (A - C) \cdot (E + F)
\]
\[
S_4 = (A + B) \cdot H
\]
\[
S_5 = A \cdot (F - H)
\]
\[
S_6 = D \cdot (G - E)
\]
\[
S_7 = (C + D) \cdot E
\]
- Let's test that $S_6 + S_7$ is really $C \cdot E + D \cdot G$

$$S_6 + S_7 = D \cdot (G - E) + (C + D) \cdot E$$
$$= DG - DE + CE + DE$$
$$= DG + CE$$

- 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.

- Let's solve the recurrence using the iteration method

$$T(n) = 7T(n/2) + n^2$$
$$= n^2 + 7 \left( T\left( \frac{n}{2^2} \right) + \left( \frac{n}{2} \right)^2 \right)$$
$$= n^2 + \left( \frac{7}{2^2} \right) n^2 + 7^2 T\left( \frac{n}{2^3} \right)$$
$$= n^2 + \left( \frac{7}{2^2} \right) n^2 + \left( \frac{7}{2^2} \right)^2 n^2 + 7^3 T\left( \frac{n}{2^5} \right)$$
$$= n^2 + \left( \frac{7}{2^2} \right) n^2 + \left( \frac{7}{2^2} \right)^2 n^2 + \left( \frac{7}{2^2} \right)^3 n^2 \ldots + \left( \frac{7}{2^2} \right)^{\log n - 1} n^2 + \gamma^{\log n}$$
$$= \sum_{i=0}^{\log n - 1} \left( \frac{7}{2^2} \right)^i n^2 + \gamma^{\log n}$$
$$= n^2 \cdot \Theta \left( \left( \frac{7}{2^2} \right)^{\log n - 1} \right) + \gamma^{\log n}$$
$$= n^2 \cdot \Theta \left( \frac{\gamma^{\log n}}{(2^2)^{\log n}} \right) + \gamma^{\log n}$$
$$= n^2 \cdot \Theta \left( \frac{\gamma^{\log n}}{n^2} \right) + \gamma^{\log n}$$
$$= \Theta(\gamma^{\log n})$$

- Now we have the following:

$$\gamma^{\log n} = \left( \gamma^{\log_7 n} \right)^{\log_2 7}$$
$$= \left( n^{1/\log_7 2} \right)^{\log_2 7}$$
$$= n^{\log_7 2}$$
$$= n^{\log 2}$$
$$= n$$

- Or in general: $a^{\log_b n} = \log_b a$

So the solution is $T(n) = \Theta(n^{\log 7}) = \Theta(n^{2.81\ldots})$
• 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) \( \Omega(n^2) \).
  – Book present Strassen’s algorithm in a somewhat strange way.

3 Master Method

• It be nice to have a general solution to \( T(n) = aT(n/b) + n^c, \ T(1) = 1. \)
  — we do!

\[
T(n) = aT\left(\frac{n}{b}\right) + n^c \quad a \geq 1, b \geq 1, c \geq 0
\]

\[
\implies T(n) = \begin{cases} 
  \Theta(n^{\log_b a}) & a > b^c \\
  n^c \log_b n + \text{smaller-order terms} & a = b^c \\
  n^c \cdot \frac{1}{1-b^c} + \text{smaller-order terms} & a < b^c 
\end{cases}
\]

• Note:
  – In Strassen’s algorithm we had \( a = 7, b = 2, \) and \( c = 2 \)
    \( \implies b^c = 2^2 = 4 < 7 = a \implies T(n) = \Theta(n^{\log_2 7}) \)
  – In merge-sort we had \( a = 2, b = 2, \) and \( c = 1 \)
    \( \implies b^c = 2^1 = 2 = a \implies T(n) = \Theta(n^{1+\log_2 n}) = \Theta(n \log n) \)

Proof (by the iteration method)

\[
T(n) = aT\left(\frac{n}{b}\right) + n^c \\
= n^c + a \left( \left( \frac{n}{b}\right)^c + aT\left(\frac{n}{b^2}\right) \right) \\
= n^c + \left( \frac{a}{b^c} \right) n^c + a^2 T\left(\frac{n}{b^2}\right) \\
= n^c + \left( \frac{a}{b^c} \right) n^c + a^2 \left( \left( \frac{n}{b^2}\right)^c + aT\left(\frac{n}{b^4}\right) \right) \\
= n^c + \left( \frac{a}{b^c} \right) n^c + \left( \frac{a}{b^c} \right)^2 n^c + a^3 T\left(\frac{n}{b^4}\right) \\
= \ldots \\
= n^c + \left( \frac{a}{b^c} \right) n^c + \left( \frac{a}{b^c} \right)^2 n^c + \left( \frac{a}{b^c} \right)^3 n^c + \ldots + \left( \frac{a}{b^c} \right)^{\log_b n-1} n^c + a^{\log_b n} n T(1) \\
= n^c \sum_{k=0}^{\log_b n-1} \left( \frac{a}{b^c} \right)^k + a^{\log_b n} n \\
= n^c \sum_{k=0}^{\log_b n-1} \left( \frac{a}{b^c} \right)^k + n^{\log_b a} \quad \text{by the logarithm identity}
\]

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

• \( a < b^c \)

\[
a < b^c \iff \frac{a}{b^c} = d < 1 \implies \sum_{k=0}^{\log_b n-1} d^k = \frac{1-d^{\log_b n}}{1-d} = \frac{1-n^{\log_b d}}{1-d} \sim \frac{1}{1-d} = \frac{1}{1-\frac{1}{b^c}}
\]

\[
T(n) = n^c \sum_{k=0}^{\log_b n-1} \left( \frac{a}{b^c} \right)^k + n^{\log_b a} \\
\sim n^c \cdot \frac{1}{1-\frac{a}{b^c}} + n^{\log_b a} \\
\sim n^c \cdot \frac{1}{1-\frac{1}{b^c}} = \Theta(n^c) \quad \text{since } \log_b a < c
\]

• \( a = b^c \)
\[ a = b^c \iff \frac{a}{b} = 1 \implies \sum_{k=0}^{\log_b n-1} \left( \frac{a}{b} \right)^k = \sum_{k=0}^{\log_b n-1} 1 = \log_b n \]

\[
T(n) = \sum_{k=0}^{\log_b n-1} \left( \frac{a}{b} \right)^k + n^{\log_b a} \\
= n^c \log_b n + n^{\log_b a} \\
= n^c \log_b n + n^c \\
\sim n^c \log_b n
\]

• \[ a > b^c \iff \frac{a}{b} > 1 \implies \sum_{k=0}^{\log_b n-1} \left( \frac{a}{b} \right)^k = \Theta \left( \left( \frac{a}{b} \right)^{\log_b n} \right) = \Theta \left( \frac{n^{\log_b a}}{n^c} \right) \]

\[
T(n) = n^c \cdot \Theta \left( \frac{n^{\log_b a}}{n^c} \right) + n^{\log_b a} \\
= \Theta (n^{\log_b a}) + n^{\log_b a} \\
= \Theta (n^{\log_b a})
\]

• Note: Book states and proves the result slightly differently. (No need to read it).

3.1 Other types of recurrences

Some important/typical bounds on recurrences:

• Logarithmic (special case of Master Method): \( \Theta (\log n) \)
  
  − 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
  
  − Can actually use a form of the Master Method to solve linear recurrences (even though it is trivial to solve it directly): Let’s define the term \( S(2^{n-1}) = T(n) \), so that \( S(1) = T(1) = 1 \). If we define \( m = 2^{n-1} \), then we have
    
    \[
    S(m) = 1 + S(m/2),
    \]
    
    which by a more careful form of the Master Method implies that \( S(m) = 1 + \log m \). Therefore, \( T(n) = 1 + \log m = 1 + \log 2^{n-1} = n \).
  
  − The point is that the Master Method applies in many situations where you might not think so at first.
  
  − 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: \( T(n) = 2T(n - 1) \)