Lecture 4: Recurrences and Strassen's Algorithm

(CLRS 4.1-4.2, 28.1+28.2)

May 21st, 2002

1 Summation review

- Last time we computed a number of sum's using:
  - Splitting and bounding terms ideas
  - Induction (!)

- Arithmetic sum:
  - $\sum_{k=1}^{n} k = \frac{n(n+1)}{2} = \Theta(n^2)$
  - $\sum_{k=1}^{n} k^d = \Theta(n^{d+1})$
- Geometric sum:
  - $\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1} = \Theta(x^n)$
- Harmonic sum:
  - $\sum_{i=1}^{n} \frac{1}{k} = \Theta(\log n)$

2 Recurrences

- Last time we started discussing how to solve recurrences.
  - Recurrences often needs to be solved in order to analyze divide-and-conquer algorithms.
- We saw how to solve the recurrence $T(n) = 2T(n/2) + n$ using the substitution method
  - Idea in substitution method is to make good guess and prove by induction.

2.1 Substitution method

- Solution to $T(n) = 2T(n/2) + n$ using substitution
  - Guess $T(n) \leq cn \log n$ for some constant c (that is, $T(n) = O(n \log n)$)
  - Proof:
    * Basis: Function constant for small constant n

* 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:

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

  So ok if $c \geq 1$

- The hard part of the substitution method is often to make a good guess.

2.2 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 $T(n) = 8T(n/2) + n^2 \quad (T(1) = 1)$

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

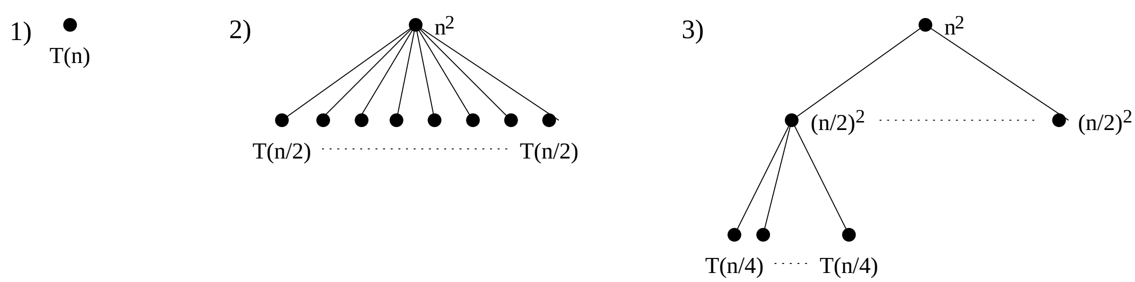
  - How long does it continue? i times where $\frac{n}{2^i} = 1 \Rightarrow i = \log n$
  - What is the last term? $8^iT(1) = 8^{\log n}$

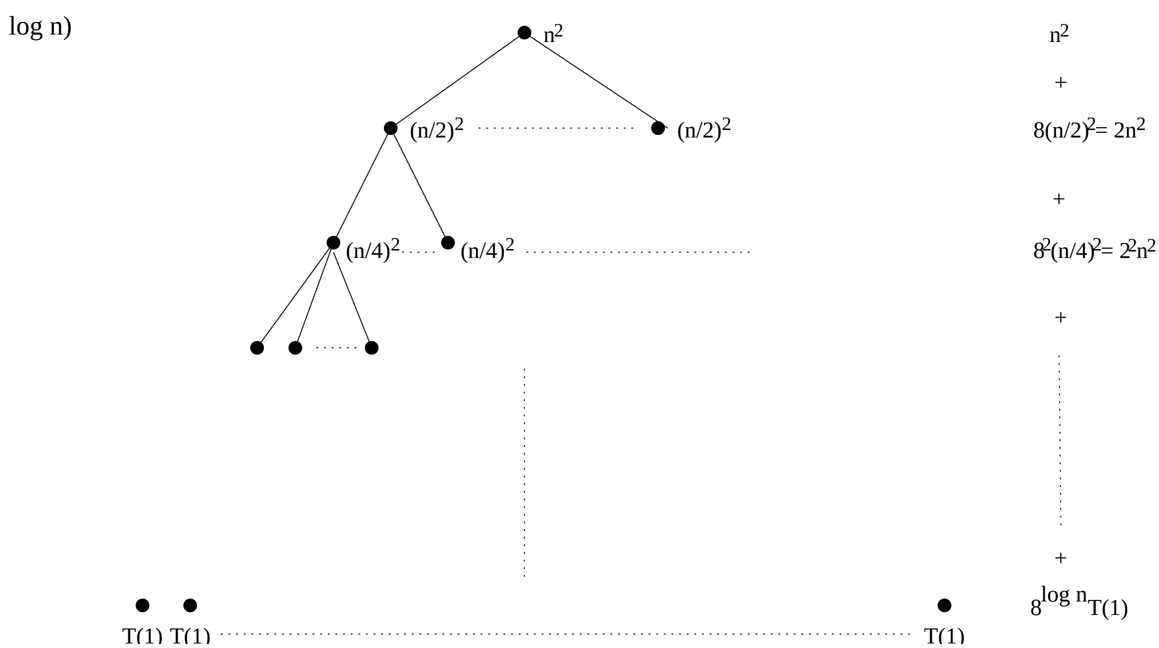
$$
\begin{aligned}
T(n) &= n^2 + 2n^2 + 2^2n^2 + 2^3n^2 + 2^4n^2 + \ldots + 2^{\log n - 1}n^2 + 8^{\log n} \\
&= \sum_{k=0}^{\log n - 1} 2^k n^2 + 8^{\log n} \\
&= n^2 \sum_{k=0}^{\log n - 1} 2^k + (2^3)^{\log n}
\end{aligned}
$$

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

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

- The book discuss a different way of looking at the iteration method: the recursion-tree method
  - we draw out the recursion tree with cost of single call in each node—running time is sum of costs in all nodes (like we discussed when analyzing merge-sort).
  - really the same as iterating.
  - Example: $T(n) = 8T(n/2) + n^2 \quad (T(1) = 1)$





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

3 Matrix Multiplication

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

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

- 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

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

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

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

- 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 $T(n) = \Theta(n^{\log 7}) = \Theta(n^{2.81...})$

- 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.