CS483-07 Divide and Conquer

Instructor: Fei Li

Room 443 ST II

Office hours: Tue. & Thur. 1:30pm - 2:30pm or by appointments

`lifei@cs.gmu.edu` with subject: CS483

`http://www.cs.gmu.edu/~ lifei/teaching/cs483_fall07/`

This lecture note is based on notes by Anany Levitin and Jyh-Ming Lian.

Announcements

- October 9: no class. Oct. 8th is Columbus Day recess.
- Review class: October 11.
- Midterm is scheduled on October 16, 2007
- Today's lecture: Divide and Conquer (cont')
  1. Quicksort
  2. Binary search
  3. Binary tree traversal
  4. Strassen's matrix multiplication

General Divide-and-Conquer Recurrence

- Problem size: n. Divide the problems into b smaller instances; a of them need to be solved. $f(n)$ is the time spent on dividing and merging.

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

- **Master Theorem**: If $f(n) \in \Theta(n^d)$, where $d \geq 0$, then

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

Sorting Problem

- Given an array of n numbers, sort the elements in non-decreasing order.
- Input: An array $A[1, \ldots, n]$ of numbers
- Output: An array $A[1, \ldots, n]$ of sorted numbers

Mergesort - Algorithm

➢ Given an array of n numbers, sort the elements in non-decreasing order.

Algorithm 0.1: MERGESORT($A[1, \ldots n]$)

if $n = 1$

then return (A)

else $\begin{cases} B \leftarrow A[1 \cdots \lfloor \frac{n}{2} \rfloor] \\ C \leftarrow A[\lceil \frac{n}{2} \rceil \cdots n] \\ MergeSort(B) \\ MergeSort(C) \\ Merge(B, C, A) \end{cases}$

Mergesort - Algorithm

➢ Merge two sorted arrays, B and C and put the result in A

Algorithm 0.2: MERGE($B[1, \ldots p], C[1, \ldots q], A[1, \cdots p + q]$)

$i \leftarrow 1; j \leftarrow 1$

for $k \in \{1, 2, \ldots p + q - 1\}$

do $\begin{cases} \textbf{if } B[i] < C[j] \\ \quad \textbf{then } A[k] = B[i]; i \leftarrow i + 1 \\ \quad \textbf{else } A[k] = C[j]; j \leftarrow j + 1 \end{cases}$

Mergesort - Analysis

- All cases have same time efficiency: $\Theta(n \log_2 n)$
  $T_{\text{merge}}(n) = n - 1.$

$$T(n) = 2T(n/2) + n - 1, \quad \forall n > 1, \quad T(1) = 0$$

- Number of comparisons in the worst case is close to theoretical minimum for comparison-based sorting: $\lceil \log_2 n! \rceil \approx n \log_2 n - 1.44n$
- Space requirement: $\Theta(n)$ (not *in-place*) (In-place: The number are rearranged within the array.)
- Can be implemented without recursion?
- Is this algorithm `Mergesort` stable? (Stable: the output perserves the input order of equal elements.)

Quicksort - Algorithm

- Given an array of n numbers, sort the element in non-decreasing order.

Algorithm 0.3: QUICKSORT($A[1 \cdots n]$)

if $n = 1$
  then return (A)
  else
    Create two arrays B,C
    for $i \in \{2, 3, \ldots n\}$
      do
        if $A[i] < A[1]$
          then $B \leftarrow A[i]$
          else $C \leftarrow A[i]$
    $Quicksort(B)$
    $Quicksort(C)$
    $A \leftarrow (B, A[1], C)$

- $A[1]$ is chosen as the **pivot**. In general, any number can be the pivot.

- Example: 24, 11, 91, 10, 22, 32, 22, 3, 7, 99
- Is this algorithm `Quicksort` stable?

Quicksort - Algorithm

- `Quicksort` allows fast "in-place partition". Consider large files ($n \geq 10000$).

Algorithm 0.4: PARTITION($A[a \cdots b]$)

$p \leftarrow A[a]$

$i \leftarrow a + 1; j \leftarrow b$

repeat

$$\left\{\begin{array}{l} \textbf{while } A[i] < p \\ \quad \textbf{do } i \leftarrow i + 1 \\ \textbf{while } A[j] > p \\ \quad \textbf{do } j \leftarrow j - 1 \\ \textbf{if } i < j \\ \quad \textbf{then } \text{swap } (A[i], A[j]) \end{array}\right.$$

until $i \geq j$

swap $(A[a], A[j])$

Example: $5, 7, 3, 2, 8, 3, 6.$

Quicksort - Analysis

- Best case: split in the middle – $\Theta(n \log n)$.

$$T(n) = 2T(n/2) + \Theta(n).$$

- Worst case: sorted array! – $\Theta(n^2)$.

$$T(n) = T(n-1) + \Theta(n).$$

- Average case: random arrays – $\Theta(n \log n)$
- Improvements (these combine to $20 - 25\%$ improvement):
  1. Better pivot selection: median of three partitioning
  2. Switch to insertion sort on small subfiles.
  3. Elimination of recursion.

Binary Search

- Imagine that you are placed in an unknown building and you are given a room number (say STII, 443), you need to find your CS 483 instructor. What will you do?
- **Binary Search**:
  - Very efficient algorithm for searching in **sorted array**

    Example: find 70 in {3, 14, 27, 31, 39, 42, 55, 70, 74, 81, 85, 93, 98}

Binary Search - Algorithm

- Given a sorted array A of n numbers, find a key K in A

Algorithm 0.5: BINARYSEARCH($A[1 \cdots n], K$)

$a \leftarrow 1; b \leftarrow n$

while $a < b$

do $\begin{cases} m \leftarrow \lfloor \frac{a+b}{2} \rfloor \\ \textbf{if } K = A[m] \\ \textbf{return } (m) \\ \quad \textbf{else if } K < A[m] \\ \qquad b \leftarrow m - 1 \\ \quad \textbf{else } a \leftarrow m + 1 \end{cases}$

return (-1)

Binary Search - Analysis

- $T_{worst}(n)$

  $T_{\text{worst}}(n) = T_{\text{worst}}(\lfloor n/2 \rfloor) + 1 = \Theta(\log_2 n)$, for $n > 1, T_{\text{worst}}(1) = 1$.

- $T_{best}(n)$

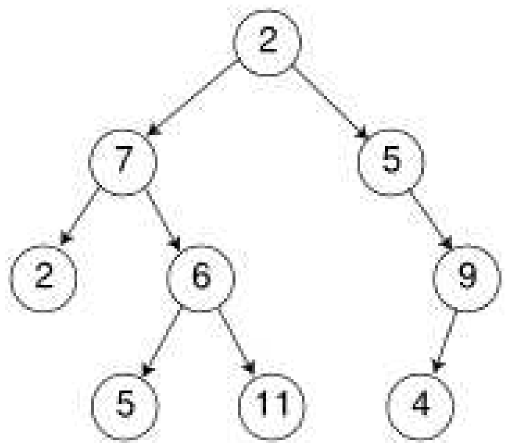
  1

- $T_{avg}(n)$

  $\Theta(\log_2 n)$.

Binary Tree

- In a binary tree, each node has zero or two nodes.



- Compute the height of a given binary tree T

Algorithm 0.6: HEIGHT(T)

if $T = \emptyset$ **return** (-1)

else return $(\max\{Height(T_L), Height(T_R)\} + 1)$

Binary Tree Traversals

- 3 classical traversals
  - **Preorder traversals**: root $\longrightarrow$ left subtree $\longrightarrow$ right subtree
  - **Inorder traversals**: left subtree $\longrightarrow$ root $\longrightarrow$ right subtree
  - **Postorder traversals**: left subtree $\longrightarrow$ right subtree $\longrightarrow$ root
- Example: page 142.

Integer Multiplication

- What is the time complexity of multiplying two integers using the algorithms we learned in elementary schools?

  Example: how do you compute this: 12345×67890?

  n^2 digit multiplication $+\, n$ addition

- Is there a better way of multiplying two integers, in terms of reducing the number of multiplication?

  Carl Friedrich Gauss (1777-1855) discovered that $AB = (a10^{\frac{n}{2}} + b)(c10^{\frac{n}{2}} + d) = K_2 10^n + K_1 10^{\frac{n}{2}} + K_0$, where $K_2 = ac$, $K_0 = bd$, $K_1 = (a+b)(c+d) - (K_0 + K_2)$.

  Example: how do you compute this: 12345×67890?

Integer Multiplication

- Divide-and-conquer integer multiplication

Algorithm 0.7: M($A[1 \cdots n], B[1 \cdots n]$)

if $n = 1$

then return $(A[1]B[1])$

$$\textbf{else} \begin{cases} a \leftarrow A[1 \cdots \frac{n}{2}], b \leftarrow A[\frac{n}{2}+1 \cdots n] \\ c \leftarrow B[1 \cdots \frac{n}{2}], d \leftarrow B[\frac{n}{2}+1 \cdots n] \\ K_2 \leftarrow M(a, c) \\ K_0 \leftarrow M(b, d) \\ K_1 \leftarrow M(a+b, c+d) - (K_0 + K_2) \\ \textbf{return } (K_2 10^n + K_1 10^{\frac{n}{2}} + K_0) \end{cases}$$

Integer Multiplication

- What is the time complexity?

  First we formulate the time complexity as: $T(n) = 3T(\frac{n}{2}) + O(n)$.

  Using Master Theorem, we have $a = 3$, $b = 2$ and $d = 1$. So, $T(n) = \Theta(n^{\log_2 3}) \approx \Theta(n^{1.6})$

Matrix Multiplication

Strassen's Matrix Multiplication:

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & A_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$= \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{bmatrix}$$

- $m_1 = (A_{11} + A_{22})(B_{11} + B_{22})$
- $m_2 = (A_{21} + A_{22})B_{11}$
- $m_3 = A_{11}(B_{12} - B_{22})$
- $m_4 = A_{22}(B_{21} - B_{11})$
- $m_5 = (A_{11} + A_{12})B_{22}$
- $m_6 = (A_{21} - A_{11})(B_{11} + B_{12})$
- $m_7 = (A_{12} - A_{22})(B_{21} + B_{22})$

Matrix Multiplication

- What is the time complexity?

  $T(n) = 7T(\frac{n}{2}) + O(n)$

  Using Master Theorem, we have $a = 7$, $b = 4$ and $d = 1$.

  So, $T(n) = \Theta(n^{\log_2 7}) \approx \Theta(n^{2.8})$

- Do you still remember what the time complexity of the brute-force algorithm is?

  $O(n^3)$

Summary of Sorting Algorithms

Algorithm	Time	Notes
selection-sort	$O(n^2)$	in-place. slow (good for small inputs)
insertion-sort	$O(n^2)$	in-place. slow (good for small inputs)
quick-sort	expected $O(n \log n)$	in-place, randomized, fastest (good for large inputs)
merge-sort	$O(n \log n)$	sequential data access. fast (good for huge inputs)