G205 Fundamentals of Computer Engineering

CLASS 6, Wed. Sept. 24 2003

Stefano Basagni

Fall 2003

M-W, 9:50am-11:30am, 410 Ell

SORTING, 1

- As a computational problem:
  - INPUT: a sequence of *n* numbers

    <a1, a2, …, an>
  - OUTPUT: A permutation (reordering)

    <a'1, a'2, …, a'n> on the input sequence such that:

    $a'1 \le a'2 \le \ldots \le a'n$

SORTING, 2

- Input sequence: *<31,41,59,26,41,58>*
- Output sequence: *<26,31,41,41,58,59>*
- The input sequence is called an INSTANCE of the sorting problem
- One CP → many (sorting) algorithms

INSERTION SORT

```
Insertion-Sort(A,n)
    for j = 2 to n do
        key = A[j]
        i = j – 1
        while ( i > 0 ) and ( A[i] > key ) do
            A[ i + 1 ] = A[ i ]
            i = i – 1
        A[ i + 1 ] = key
```

IS Properties

- Sort a hand of playing cards
- Input is an array A[1…n]
- Sorting *in place*
- Running time
  - Worst case, number sorted in reverse order: $O(n^2)$
  - Best case, numbers already sorted: $O(n)$

BUBBLE SORT

```
Bubble-Sort(A,n)
  for i = 1 to n do
    for j = n downto i + 1 do
      if A[ j ] < A[ j – 1 ]
        then SWAP( A[ j ], A[ j - 1 ] )
```

BUBBLE SORT Correctness, 1

- Let A′ be the output of Bubble-Sort
- We need to prove:
  1. Termination (easy)
  2. $A'[1] \leq A'[2] \leq \ldots \leq A'[n]$
  3. Elements of A′ are a permutation of the elements of A

BS Correctness, 2

- **Loop invariant:** At the start of each iteration of the inner **for** loop

  $$A[j] = \min\{A[k] : j \leq k \leq n\}, \text{ and}$$

  A[j…n] is a permutation of the values that were in A[j…n] at the time that the loop started

BS Correctness, 3

- **Initialization:** Initially, j=n, and A[j…n] consists of the single element A[n]. The loop invariant trivially holds
- **Maintenance:** A[j] is the smallest value in A[j…n]. A[j] and A[j-1] are swapped if A[j] < A[j-1]. Since the only change to A[j-1…n] is this and since A[j…n] is a permutation of A[j…n] at the time that the loop started, then A[j-1…n] is a permutation of A[j-1…n]. Decrementing j for the next iteration maintains the invariant

BS Correctness, 4

- **Termination:** The loop terminates when j reaches i. By the statement of the loop invariant, $A[i] = \min\{A[k] : i \le k \le n\}$ and $A[i\ldots n]$ is a permutation of the values that were in $A[i\ldots n]$ at the time that the loop started
- EXERCISE: Finish proving the correctness (outermost **for**)

BUBBLE SORT, Running Time

- The BS running time depends on the number of iterations of **for** loops. For each i=1, 2,…, n the innermost loop makes n-i iterations

  $T_{BS}(n)$ = SUM(i=1,n) n-i

  = SUM(i=1,n) n – SUM(i=1,n) i =

  = $n^2/2 + n/2$

- $T_{BS}(n)$ is $O(n^2)$ in ***all*** cases (same as insertion sort in the worst case)

Algorithm Design

- Insertion-Sort and Bubble-Sort use an *incremental* approach to sorting
- *Divide and Conquer*:
  - Divide the problem into subproblems
  - Conquer the subproblems by solving them recursively
  - Combine the solutions to the subproblems to give a solution to the original problem

MERGE SORT, 1

- Follows the D&C approach
- To sort A[p…r]:
  - Divide the elements of A into two subarrays A[p…q] and A[q+1…r]
  - Conquer by recursively sorting the two subarrays
  - Combine by merging the two sorted subarrays to produce the sorted A[p…r]
- Recursion bottoms out when the subarray has just one element

MERGE SORT, 2

```
Merge-Sort(A, p, r)
  if p < r                              check the base case
    then q = int((p+r)/2)               divide
         Merge-Sort(A, p, q)            conquer
         Merge-Sort(A, q+1, r)          conquer
         Merge(A, p, q, r)              combine
```

- Initial Call: Merge-Sort(A,1,n)

Merging in Merge-Sort

- Input: Array A and indices p, q, and r:
  - $p \leq q < r$
  - Subarrays A[p…q] and A[q+1…r] are sorted (neither is empty)
- Output: The two subarrays are merged into a single sorted subarray A[p…r]
- We want it to be $\Theta(n)$, where $n=r-p+1$

(Note: n is now the size of a subproblem)

Idea of Linear Time Merging

- Think of two piles of cards
  1. Each pile is sorted, face up (small on top)
  2. Merge them into one sorted pile, face down. Basic step:
     - Choose the smallest of the top cards
     - Remove it from its pile
     - Place it face down onto the output pile
  3. Repeat basic step till one pile is empty
  4. Place the remaining input pile face down onto the output pile

Why it is Linear

- Each basic step takes constant time
- There are $\leq$ n basic steps, since each step removes one card, and we started with n cards
- The whole procedure takes $\Theta(n)$ time

The Procedure Merge

```
Merge(A, p, q, r)
  n1 = q-p+1
  n2 = r-q
  create arrays L[1…n1+1], R[1…n2+1]
  for i = 1 to n1 do L[i] = A[p+i-1]
  for j = 1 to n2 do R[j] = A[q+j]
  L[n1+1] = R[n2+1] = BIG                    "sentinel"
  i = j = 1
  for k = p to r do
    if L[i] <= R[j] then A[k] = L[i]
                         i = i+1
                    else  A[k] = R[j]
                         j = j+1
```

Correctness of Merge(A,p,q,r)

- Loop invariant (textbook):

  At the start of each iteration of the "for k" the subarray A[p…k-1] contains the k-p smallest elements of L[1…n1+1] and R[1…n2+1] in sorted order. Moreover, L[i] and R[j] are the smallest elements of their arrays that have not been copied into back into A

- We see an example ;-)

Running Time of Merge

- The first two for loops take
  - $\Theta(n1+n2) = \Theta(n)$
- Last for makes n iterations, each taking constant time → $\Theta(n)$
- Total running time for Merge: $\Theta(n)$

Analyzing D&C Algorithms

- We use recurrence equations
- Base case: problem size is small enough ($n \leq c$). Costs constant time $\Theta(1)$
- Recursive case:
  - Divide the problem into a subproblems each 1/b the size of the original
  - Let D(n) be the time to divide a n-size problem
  - Each subproblem costs T(n/b) → all cost aT(n/b)
  - Let C(n) be the time to combine solutions

Recurrence for D&C

$T_{D\&C}(n)=\Theta(1)$ if $n \le c$

$T_{D\&C}(n)=aT_{D\&C}(n/b)+D(n)+C(n)$

otherwise

Analyzing Merge-Sort

- Base case: $n=1$ ($p \geq r$) → $T(1)$ in $\Theta(1)$
- When $n \geq 2$:
  - Divide: Compute q as the average of p and r → $D(n)$ in $\Theta(1)$
  - Conquer: Recursively solve two n/2-size subproblems → $2T(n/2)$
  - Combine: Merge on a n-element subarray takes $\Theta(n)$ → $C(n)$ in $\Theta(n)$

Recurrence for Merge-Sort

$T_{MS}(n)=\Theta(1)$ if $n = 1$

$T_{MS}(n)=2T_{MS}(n/2)+ \Theta(n)$ if $n > 1$

- By the MASTER THEOREM:

$$T_{MS}(n)=\Theta(n\log n)$$

- Faster than IS and BS

Assignments

- Textbook, pages 27—38
- Updated information on the class web page:

  www.ece.neu.edu/courses/eceg205/2003fa