

Analysis of Algorithms CS 3343 Lecture Fifteen

Prof. William Winsborough
November 13, 2008

Business

- Turn in Homework 8
- Recall:
 - Homework 9 due Tuesday 11/18
 - 8.1-1, 8.2-1, 8.2-2, 8.2-3, 8.2-4, 8.3-1, 8.3-2, 8.3-3, 8.3-4
 - Midterm 2 will be Thursday November 20
 - Tuesday November 18 will be devoted to review
- Today's slides are only slightly different from lecture 14 slides
 - We'll review the last part of the lower bound for comparison sorts
- Questions?

13 November 2008

Winsborough CS 3343 Lecture 15

2

Comparison Sorting Algorithms

- Comparison sorting algorithms work by comparing elements of the input
 - All the sorting algorithms we have studied so far are comparison sorts
- The lower bound on time required by any comparison sorting algorithm in the worst case: $\Omega(n \lg n)$
 - See next two slides for proof
 - So merge sort and heap sort achieve the best asymptotic runtime possible for a comparison sort
 - In chapter 8, we study some sorting algorithms that use operations other than comparisons

13 November 2008

Winsborough CS 3343 Lecture 15

3

Decision Trees

- Internal nodes give comparisons performed by any given algorithm
 - All other aspects of the algorithm are "abstracted" away
- Leaves give permutations of input that correspond to sorted order
 - $\langle \pi(1), \pi(2), \dots, \pi(n) \rangle$
 - Both comparisons and permutations are expressed in terms of the indices of the input elements
- Example: figure 8.1, p.166: insertion sort of [6,8,5]
- For each internal node, $a_i \leq a_j$, the left subtree gives subsequent comparisons to be performed when $a_i \leq a_j$, and the right, when $a_i > a_j$
 - Each path from root to leaf gives the comparisons required to determine that the permutation π labeling the leaf is correct

13 November 2008

Winsborough CS 3343 Lecture 15

4

Comparison Sorting Lower Bound

- Any sorting algorithm has to be able to return any permutation of its input
 - The only basis for distinguishing which permutation is correct is by comparing elements of the input
 - The worst case number of comparisons that must be performed is the length of the longest path from the root to a leaf
- There are $n!$ different permutations
 - Therefore the number of leaves k satisfies $k \geq n!$

13 November 2008

Winsborough CS 3343 Lecture 15

5

Counting Sort

- Requirement: each of n inputs is an integer in the range $0..k$ for some integer k
- Idea: for each integer $i \in \{0..k\}$, determine the number of input elements $x \leq i$
 - Use this to place x in the correct position
- Variables:
 - $A[1..n]$: input
 - $B[1..n]$: output
 - $C[0..k]$: temporary working storage

13 November 2008

Winsborough CS 3343 Lecture 15

6

Counting Sort Algorithm

- Counting-Sort(A, B, k)
 - for $i \leftarrow 0$ to k
 - do $C[i] \leftarrow 0$
 - for $j \leftarrow 1$ to $\text{length}[A]$
 - do $C[A[j]] \leftarrow C[A[j]] + 1$
 - % $C[i]$ now contains the number of elements equal to i
 - for $i \leftarrow 1$ to k
 - do $C[i] \leftarrow C[i] + C[i-1]$
 - % $C[i]$ now contains number of elements less than or equal to i
 - for $j \leftarrow \text{length}[A]$ downto 1
 - do $B[C[A[j]]] \leftarrow A[j]$
 - $C[A[j]] \leftarrow C[A[j]] - 1$
- Example p.169

13 November 2008

Winsborough CS 3343 Lecture 15

7

Time Complexity of Counting Sort

- for loop 1-2: $O(k)$
- for loop 3-4: $O(n)$
- for loop 6-7: $O(k)$
- for loop 9-11: $O(n)$
- Total: $O(n+k)$
 - Usually use counting sort when $k = O(n)$, in which case running time is $\Theta(n)$
- No comparisons

13 November 2008

Winsborough CS 3343 Lecture 15

8

Property of Sorting Algorithms: Stability

- An algorithm is stable if multiple copies of the same value are ordered as they were in the original input
 - This is usually most relevant when there is satellite data and we are sorting keys
- Counting sort is stable
 - This is important when it is used as a subroutine of radix sort

13 November 2008

Winsborough CS 3343 Lecture 15

9

Radix Sort

- Used by antique computer punch-card sorting machines
 - d -digit number occupies a field of d columns
 - Sort first by least-significant digit
 - Iterate up to most-significant
- Sometimes used to sort by multiple fields
 - Dates, check number, etc.

13 November 2008

Winsborough CS 3343 Lecture 15

10

Radix Sort

- Radix-Sort(A, d)
 - for $i \leftarrow 1$ to d
 - do use a stable sort to sort A on digit i
- Digit 1 is least significant, d is most
- Assuming each digit can take on up to k values, Radix-Sort correctly sorts in $\Theta(d(n+k))$ time
 - Induction on the column being sorted
 - Complexity relies on using $\Theta(n+k)$ stable sort, such as counting sort

13 November 2008

Winsborough CS 3343 Lecture 15

11

Using Radix Sort by Breaking Up Keys

- Given n b -bit numbers and any positive integer $r \leq b$, Radix-Sort correctly sorts these numbers in $\Theta((b/r)(n+2^r))$ time
 - View each key as having $d = \lceil b/r \rceil$ digits of r bits each
 - Each digit is in the range $0..2^r - 1$
 - Use counting sort with $k = 2^r - 1$
 - Each pass takes time $\Theta(n + k) = \Theta(n + 2^r)$
 - d passes gives a total time of $\Theta(d(n + 2^r)) = \Theta((b/r)(n+2^r))$

13 November 2008

Winsborough CS 3343 Lecture 15

12