G205
Fundamentals of Computer Engineering

CLASS 7, Mon. Sept. 29 2003
Stefano Basagni
Fall 2003
M-W, 9:50am-11:30am, 410 Ell
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
then q = int((p+r)/2)
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)
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 \(T(n/b)\) \(\rightarrow\) all cost \(aT(n/b)\)
  - Let \(C(n)\) be the time to combine solutions
Recurrence for D&C

\[ T_{D&C}(n) = \Theta(1) \quad \text{if } n \leq c \]

\[ T_{D&C}(n) = aT_{D&C}(n/b) + D(n) + C(n) \quad \text{otherwise} \]
Analyzing Merge-Sort

- **Base case:** \( n=1 \ (p \geq r) \) \( \Rightarrow \) \( T(1) \) in \( \Theta(1) \)

- **When** \( n \geq 2 \):
  - **Divide:** Compute \( q \) as the average of \( p \) and \( r \) \( \Rightarrow \) \( D(n) \) in \( \Theta(1) \)
  - **Conquer:** Recursively solve two \( n/2 \)-size subproblems \( \Rightarrow 2T(n/2) \)
  - **Combine:** Merge on a \( n \)-element subarray takes \( \Theta(n) \) \( \Rightarrow \) \( C(n) \) in \( \Theta(n) \)
Recurrence for Merge-Sort

\[ T_{MS}(n) = \Theta(1) \quad \text{if } n = 1 \]
\[ T_{MS}(n) = 2T_{MS}(n/2) + \Theta(n) \quad \text{if } n > 1 \]

\( \blacklozenge \) By the MASTER THEOREM:

\[ T_{MS}(n) \text{ is in } \Theta(n \log n) \]

\( \blacklozenge \) Faster than IS and BS
Merge-Sort Recurrence

Without the Master Theorem

Rewrite the recurrence:

- $T_{MS}(n) = c$ if $n = 1$
- $T_{MS}(n) = 2T_{MS}(n/2) + c$ if $n > 1$

Recursion Tree = successive expansion of the recurrence
Merge-Sort Recursion Tree

- Each level of the tree has cost $cn$
- There are $\log n + 1$ levels
  - Prove it by induction
- Total cost is $cn(\log n + 1)$ $cn \log n + cn$
- $T_{MS}(n)$ is in $\Theta(n\log n)$ "<" $O(n^2)$

QUESTION:
HOW FAST CAN WE SORT?
Lower Bounds for Sorting

- Lower bound: A function or growth rate below which solving a problem is impossible
- A measure of how much has to be spent
- Natural lower bound for sorting: All elements must at least be read $\Omega(n)$
Comparison-based Sorting

- The only operation that may be used to gain order information about a sequence is comparison of pairs of elements.
- All sorts seen so far are comparison sorts: insertion sort, bubble sort, merge sort.
- Other famous sorting algorithms are too: quicksort, heapsort, treesort.
Decision Tree, 1

- Abstraction of any comparison sort
- Represents comparisons made by
  - a specific sorting algorithm
  - on inputs of a given size
- Abstracts away everything else: control and data movement
- We are counting *only* comparisons
Decision Tree, 2

For any comparison-based sorting:
- One tree for each $n$
- The algorithm splits in two at each node, based on the information it has up to that point
- The tree models all possible execution traces

The length $h$ of the longest root-leaf path:
- Depends on the algorithm
  - Insertion sort: $\Theta(n^2)$
  - Merge sort: $\Theta(n \log n)$
Decision Tree, 3

- Lemma: Any binary tree of height $h$ has $l \leq 2^h$ leaves (by induction)
- Theorem: *Any* decision tree that sorts $n$ elements has height $\Omega(n \log n)$
- Proof
  - Every decision tree has $l \geq n!$ leaves (every permutation appears at least once)
  - By lemma, $n! \leq l \leq 2^h$ or $2^h \geq n! \Rightarrow h \geq \log n!$
  - Stirling approximation: $n! \geq (n/e)^n \Rightarrow h \in \Omega(n \log n)$
Lower Bound for Comparison-based Sorting

- The height of a decision tree indicates how many comparison at least have to be made to sort a sequence of n elements → lower bound for sorting
- Comparison-based sorting is in \( \Omega(n \log n) \)
- Merge-Sort is as good as it gets (asymptotically optimal)
Sorting in Linear Time

- We cannot go faster than $\Omega(n)$
- Must be a non-comparison sorting
- Works when assumptions on the number to be sorted are made
  - Counting sort $\rightarrow$ numbers in $\{0, 1, \ldots, k\}$
  - Radix sort $\rightarrow$ numbers with a constant number of digits
  - Bucket sort $\rightarrow$ numbers drawn from a uniform distribution
Counting Sort, 1

Numbers are integers in \{0,1,...,k\}

INPUT: A[1...n], A[j] ∈ \{0,1,...,k\} for all j=1,2,...,n. Array A and values n and k are given as parameters

OUTPUT: B[1...n], sorted. B is assumed to be already allocated and is given as a parameter

Auxiliary storage: C[0...k]
Counting Sort, 2

Counting-Sort(A, B, n, k)
  for i = 0 to k do C[i] = 0
  for j = 1 to n do C[A[j]] = C[A[j]] + 1
  for i = 1 to k do C[i] = C[i] + C[i-1]
  for j = n downto 1 do
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
Counting Sort, Example

INPUT: A = 2₁, 5₁, 3₁, 0₁, 2₂, 3₂, 0₂, 3₃

OUTPUT: B = 0₁, 0₂, 2₁, 2₂, 3₁, 3₂, 3₃, 5₁

Counting-Sort is STABLE: keys with same value appear in same order in output as they did in input (because of how the last loop works)

Analysis: $\Theta(n+k)$, which is $\Theta(n)$ if $k$ is in $O(n)$
Radix Sort

Key idea: Sort least significant digit of each number first

To sort d digits:

Radix-Sort(A,d)

for i = 1 to d do
  use a stable sorting to sort array A on digit i
Radix Sort, Example

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>329</td>
<td>720</td>
<td>720</td>
<td>329</td>
<td></td>
</tr>
<tr>
<td>457</td>
<td>355</td>
<td>329</td>
<td>355</td>
<td></td>
</tr>
<tr>
<td>657</td>
<td>436</td>
<td>436</td>
<td>436</td>
<td></td>
</tr>
<tr>
<td>839</td>
<td>457</td>
<td>839</td>
<td>457</td>
<td></td>
</tr>
<tr>
<td>436</td>
<td>657</td>
<td>355</td>
<td>657</td>
<td></td>
</tr>
<tr>
<td>720</td>
<td>329</td>
<td>457</td>
<td>720</td>
<td></td>
</tr>
<tr>
<td>355</td>
<td>839</td>
<td>657</td>
<td>839</td>
<td></td>
</tr>
</tbody>
</table>
Radix Sort: Correctness

- Induction on number of passes (i in pseudo-code)
- Assume digits 1, 2, ..., i-1 are sorted
- Show that a stable sort on digit i leaves digits 1, 2, ..., i sorted
  - If two digits in position i are different ordering by position i is correct (other digits are irrelevant)
  - If the digits are the same, numbers are already in the right order (ind. hyp.)
Radix Sort, Analysis

- Use Counting Sort as stable sorting
- $\Theta(n+k)$ per pass
- $d$ passes
- $\Theta(d(n+k))$ total
- If $k$ is in $O(n)$ the $T_{RS}(n)$ is in $\Theta(dn)$
- When $d$ is $\Theta(1)$ Radix Sort is linear time
How to break a number into digits

- $n$ b-bits numbers
- Break into $r$-bits digits, have $d = \text{ceil}(b/r)$
- Use Counting Sort $k = 2^r - 1$
- $T_{RS}(n)$ is in $\Theta((b/r)(n+2^r))$
- Exercise: Choose $r$ and compare Radix Sort and Merge-Sort
Assignments

- Textbook, pages 165—173
- Updated information on the class web page:
  www.ece.neu.edu/courses/eceg205/2003fa