CS472 Module 10 Part B - Sorting and Computational Complexity

Athens State University

April 4, 2016

Recall: Computational Complexity

- Study of all possible algorithms that solve a given problem
- Determine a lower bound on efficiency of all algorithms for a given problem
- Problem analysis rather than algorithm analysis; consider the mathematical aspects of a problem that make it easy or difficult

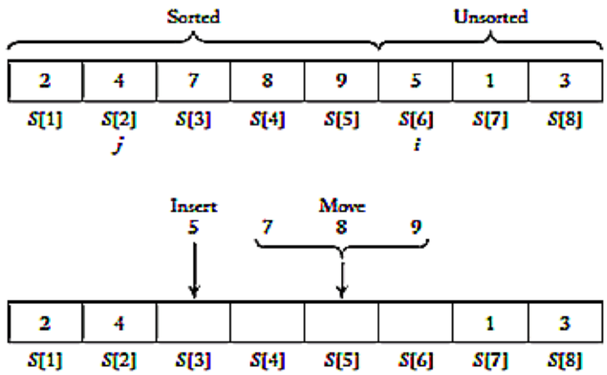
Consider: Matrix Multiplication

- It has been determined that a lower bound on the rate-of-growth for the **problem** is $\Omega(n^2)$
- Does not mean it is possible to create an algorithm that is $\Theta(n^2)$
- It means it is impossible to create an algorithm better than $\Theta(n^2)$
- So, either look for better algorithm or higher lower bound
- Best algorithm found to date is $\Theta(n^{2.38})$

A Return to Sorting

- The problem: re-arrange records according to a key field
- Algorithms that sort by comparison of keys can compare 2 keys to determine which is larger and can copy keys
  - Cannot do other operations on them
  - Algorithms: Exchange sort, Insertion sort, Selection sort

Example: Insertion Sort



Analysis Summary for Exchange, Insertion, and Selection Sorts

Algorithm	Comparison of Keys	Assignment of Records	Extra Space Use
Exchange Sort	$T(n) = \frac{n^2}{2}$	$W(n) = \frac{3n^2}{2}$ $A(n) = \frac{3n^2}{4}$	In-place
Insertion Sort	$W(n) = \frac{3n^2}{2}$ $A(n) = \frac{3n^2}{4}$	$W(n) = \frac{3n^2}{2}$ $A(n) = \frac{3n^2}{4}$	In-place
Selection Sort	$T(n) = \frac{n^2}{2}$	T(n)=3n	In-place

Permutation and Inversion

- For these three sorts, we can easily see that the worst case input of size n contains n distinct keys
  - And, as result, $n!$ different orderings
- **Permutation**: We denote the sequence $[k_1, k_2, \ldots, k_n]$ as a permutation of the first n integers
  - There are $n!$ possible permutations of the $n!$ integers
- An **inversion** in a permutation is a pair (k_i, k_j) s.t. $i < j$ and $k_i > k_j$

The Permutation Sorting Theorem

Theorem 1. *Any algorithm that sorts n distinct keys only by comparisons of keys and removes at most one inversion after each comparison must in the worst case do at least*

$$\frac{n(n-1)}{2}$$

comparison of keys and, on the average, do at least

$$\frac{n(n-1)}{4}$$

comparison of keys.

So what makes other sorts better?

- Algorithms such as the Mergesort, Quicksort, and Heapsort Algorithms remove more than one inversion
  - For example, Mergesort removes more than one inversion per step in its "merge" phase
- The downside is the space complexity: additional space is required by the book keeping these algorithms require

A Dynamic Programming Version of Mergesort

Algorithm 1: mergesort3: A DP version of Mergesort

```
Input: A array of keys S indexed from 1 to n
Output: The array S sorted in nondecreasing order
m ← power(2,floor(log(n)));
size ← 1;
for log(m) times do
    for low ← [1, ((m-2)*size - 1)] do
        mid ← low = size - 1;
        high = min(((low+2)*size - 1), n);
        merge3(low,mid,high, S);
size ← 2*size;
```

The $\Theta(n * lg(n))$ sorting algorithms

Algorithm	Comparison of Keys	Assignment of Records	Extra Space Use
Mergesort (naive)	$W(n) = n * \log(n)$ $A(n) = n * \log(n)$	$T(n) = 2 * n * \log(n)$	$\Theta(n)$ Records
Mergesort (naive)	$W(n) = n * \log(n)$ $A(n) = n * \log(n)$	$T(n) = 0$	$\Theta(n)$ links
Quicksort	$W(n) = \frac{n^2}{2}$ $A(n) = 1.38 * n * \log(n)$	$A(n) = 0.69 * n * \log(n)$	$\Theta(\log(n)$
Heapsort	$W(n) = 2 * n * \log(n)$ $A(n) = 2 * n * \log(n)$	$W(n) = n * \log(n)$ $A(n) = n * \log(n)$	In-place

Can we do better than $\Theta(n * \log(n))$?

- One can convert the sorting problem into a decision problem by building a *decision tree*
  - This is a binary tree where each node that compares the values of two elements from the list being sorted
  - Each edge is labeled with either 'YES' or 'NO'
- For every deterministic algorithm for sorting n distinct keys, there exists a pruned and valid decision tree containing exactly $n!$ leaves
- The worst case number of comparisons done in a decision tree is equal to its depth

What does this tell us about sorting?

Theorem 2. *Any deterministic algorithms that sorts n distinct keys only by comparisons of keys in the worst case do at least $\lceil \log(n!) \rceil$ comparison of keys*

Theorem 3. *For any positive integer n,* $\log(n!) \geq n * \log(n) - 1.45 * n$.