CMSC 351 - Introduction to Algorithms
Spring 2012
Lecture 9

Instructor: MohammadTaghi Hajiaghayi
Scribe: Rajesh Chitnis

1 Introduction

In this lecture we will look at Insertion Sort, Selection Sort and Merge Sort.

2 Notations

Recall the following notations that we will be using here : $\sum_{i=1}^{n}, \prod, \min, \max, \cup, \cap$

3 Sorting

Sorting is one of the most extensively studied problems in computer science. It is the basis for many algorithms and it consumes a large proportion of computing time for many typical applications. There are dozens of sorting algorithms but we cover only a few in this course.

The Problem: Given n numbers $x_1, x_2, \ldots, x_n$ we want to arrange them in increasing order. In other words, we want to find a sequence of distinct indices $1 \leq i_1, i_2, \ldots, i_n \leq n$ such that $x_{i_1} \leq x_{i_2} \leq \ldots \leq x_{i_n}$. For now we assume all numbers are distinct although all algorithms we discuss also work for non-distinct numbers as well.

Definition: A sorting algorithm is called in-place if no additional work space is used besides the initial array that holds the elements.

4 Insertion Sort

Insertion Sort is a sorting algorithm using induction. Suppose we know how to sort $n-1$ elements and we are given n numbers to sort. We can start with $n-1$ numbers and then put the n^{th} number in its correct place by scanning the $n-1$ sorted numbers until the correct place to insert is found.

The total number of comparisons for sorting n numbers may be as high as $1+2+\ldots+(n-1) = \frac{(n-1)n}{2} = O(n^2)$. Also for inserting, and thus moving, in the worst case we need $n-1$ elements to be moved and hence the total number of movements is also $O(n^2)$. We can have the elements in the array and use binary search on the sorted elements. Then the total number of comparisons is $\sum_{i=1}^{n} \lfloor \log i \rfloor = \theta(n \log n)$. However the number of movements is still $O(n^2)$.

5 Selection Sort

We can select the maximum number and put it at the end of the array by swapping it with the other elements. We recursively sort the rest of the elements. The advantage over insertion sort is that only $n-1$ data movements (swaps) are required versus $O(n^2)$ needed for insertion sort. However it takes $n-1$ comparisons to find the maximum element and so selection sort needs $O(n^2)$ comparisons as compared to $O(n \log n)$ comparisons needed for insertion sort. Using other data structures such as AVL trees or binary search trees we can do comparisons in $O(n \log n)$. We will cover binary search trees later in this course.

In Bubble Sort we swap in the unsorted part of the array (a bit of waste). If $A[i] < A[i-1]$ then it swaps $A[i]$ and $A[i-1]$. Recall in selection sort we only keep the index of the maximum element.

6 Merge Sort

Merging Operation: Denote the first set by $a_1, a_2, \ldots, a_n$ and the second set by $b_1, b_2, \ldots, b_m$. We assume both are sorted in increasing order. Scan the first set until the right place to insert b_1 is found and insert it. Then continue the scan from that place until the right place to insert b_2 is found, and so on. Since the $b_i's$ are sorted we never need to go back. The total number of movements is $O(n+m)$.

Data movement is inefficient if we insert the $b_i's$, however if we use a temporary array where each element is copied exactly once then the overall time is $O(n+m)$. It is not an in-place sorting algorithm.

Merge Sort is a divide-and-conquer (recursive) algorithm as follows:

1. Divide the sequence into parts of close-to-equal size.
2. Sort each part separately recursively.
3. Merge the two parts into one sorted array.

Let $T(n)$ denote the time complexity for sorting n numbers. Then the recurrence is

$$T(2n) = 2T(n) + O(n)\ ;\ T(2) = 1$$

As we have seen in Chapter 3 (Master Theorem) of the book [1], it is $O(n \log n)$ which is much better than insertion or selection sorts. However it requires additional storage to copy the merged sets and is not an in-place sorting algorithm.

References

[1] Udi Manber, *Introduction to Algorithms - A Creative Approach*