Complexity of Sorting

COMP 215 Lecture 12

Complexity

- The rest of this course will be concerned with complexity:
  - Evaluating the difficulty of problems apart from any particular algorithm.
- At least two ways to go about this:
  - Analyze the problem itself to find a lower bound on the complexity of finding a solution.
    - We will do this for sorting (this week and next) and searching (8th week).
    - Goal is to find the tightest (highest) lower bound possible.
  - If we *can't* get a good lower bound, reduce the problem to another problem that is “well understood”. (weeks 9 and 10.)

Review of Simple Sorts

- Insertion Sort:

```
void insertion_sort (int n, keytype S[]){
   keytype x;
   for (int i = 2; i<=n; i++) {
      x = S[i];
      j = i – 1;
      while (j > 0 && S[j] > x) {
         S[j+1] = S[j];
         j--;
      }
      S[j+1] = x;
   }
}
```

Review of Simple Sorts

- Selection Sort

```
void selection_sort (int n, keytype S[]){
   index smallest;
   for (int i = 1; i <= n-1; i++) {
      smallest = i;
      for (int j = i+1; j <= n; j++) {
         if (S[j] < S[smallest])
            smallest = j;
      }
      swap S[i] and S[smallest];
   }
}
```

Review of Simple Sorts

- Exchange Sort

```
void selection_sort (int n, keytype S[]){
   index smallest;
   for (int i = 1; i <= n-1; i++) {
      smallest = i;
      for (int j = i+1; j <= n; j++) {
         if (S[j] < S[i])
            swap S[i] and S[smallest];
      }
   }
}
```

Analysis

- Easy to see that all three are in place and worst case order n^2.
- Our book also gives some exact bounds and average case analyses.

Analyzing the Sorting Problem

- Inversions are central to analyzing the sorting problem.
- An inversion is a pair of keys that is out of order.
  - In the list [1, 3, 4, 2] there are exactly two inversions, (3, 2) and (4, 2).
  - There are no inversions in a sorted list.
- We are most interested in analyzing sorting when all keys are distinct – this leads to the worst worst cases.
- If we assume all keys are distinct, then we can, without loss of generality, assume that all keys are in the range 1..*n*.
- Note: we are confining ourself to sorts that sort only by comparison of keys.

How Many Inversions?

- What is the maximum number of inversions possible in an list of n items?
- When does it happen?

How Many Inversions?

- What is the maximum number of inversions possible in an list of n items?

$$(n-1)+(n-2)+\ldots+1=$$

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

- What is the average number of inversions?

Average # of Inversions

- Notice that every permutation of keys can be paired with its transpose.
  - E.g. [1, 3, 4, 2] and [2, 4, 3, 1].
  - A given pair of keys is an inversion in *either* the original permutation or the transpose, not both.
  - E.g. in the original (4,2) is an inversion, in the transpose (2,4) is not.
  - So each permutation-transpose pair has exactly $\frac{n(n-1)}{2}$ inversions. (Why?)
  - Between them, they average $\frac{n(n-1)}{4}$ inversions.
  - Since every permutation is equally likely we can conclude that this is the average number of inversions overall.

A Lower Bound for Some Sorts

- We can immediately determine a lower bound for sorts that remove at most one inversion per comparison.

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

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

- It is easy to see that insertion sort is in this category.
- Less easy to see that selection and exchange sort are as well.