HW 1: Algorithm Design - Sample Solution

Bart Massey

14 April 2014

Define the *subsequence reversal* operator $R_{ij} : \textbf{seq } S \rightarrow \textbf{seq } S$ to be the operator that reverses elements i through j of a sequence. That is, for a sequence

$$s = \langle s_1, s_2 \ldots s_{i-1}, s_i, s_{i+1} \ldots s_{j-1}, s_j, s_{j+1} \ldots s_{n-1}, s_n \rangle$$

we have

$$R_{ij}\ s = \langle s_1, s_2 \ldots s_{i-1}, s_j, s_{j-1} \ldots s_{i+1}, s_i, s_{j+1} \ldots s_{n-1}, s_n \rangle$$

Note that any subsequence reversal of a sequence is a permutation of that sequence.

Now let us define Subsequence Reversal Sorting in terms of R_{ij}.

Subsequence Reversal Sorting

Given: A sequence s of non-negative integers.

Find: A sequence r of n reversal operators on s such that

$$(r_n \circ r_{n-1} \circ \ldots \circ r_2 \circ r_1)\ s$$

yields a permutation of s that is in nondecreasing order.

We can adapt any number of sorting algorithms to work on reversals: let's do selection sort because it is easy to analyze.

Reversal Selection Sort

```
To reversal selection sort a sequence s of integers:
  for $i \leftarrow 1$ to $|s|$
    $m \leftarrow i$
    for $j \leftarrow i+1$ to $|s|$
      if $s[m] > s[j]$
        $m \leftarrow j$
    $s \leftarrow R[i,m]\ s$
  return $s$
```

To analyze this, notice that the comparison is in a doubly-nested loop, so (by Gauss's Sum) the algorithm requires $O(n^2)$ comparisons where n is the number of elements of the sequence to be sorted. The reversal is in a singly nested loop, so $O(n)$ reversals, which in the worst case requires reversing $O(n^2)$ elements.

Thus, this algorithm answers the first optional question. To answer the second, we will construct a family of sequences that require $O(n^2)$ element reversals for a sequence of n elements, and thus develop a lower bound $\Omega(n^2)$ on the number of elements to be reversed in reversal sorting.

Consider an even sequence

$$a(n) = \langle 1, 3, 5 \ldots n-3, n-1, 2, 4, 6 \ldots n-2, n \rangle$$

To sketch a proof that $a(n)$ requires $\Omega(n^2)$ elements to sort, we proceed by forward induction on n.

Base Case: When $n = 1$, the proposition is trivially true.

Inductive Case: Assume that the proposition is true for $n-2$. Now note that the $n-1$st element needs to move to the end, and that this cannot be done in less than $n/2$ element moves, and that none of these moves can be of assistance in the previous sorting which was, by hypothesis, already optimal. Thus we can see that the cost grows by $\Omega(n)$ in each step, and thus the bound is maintained.

I call this a sketch because I am dubious of the quality of this "proof": it may be wrong. If somebody finds an algorithm purporting to require less than $O(n^2)$ time, I'll be really interested.