Practice Problems Set 5

Fall 25

1. Partly Sorted (attributed to CLRS Exercise 8.1-4)

   You are given an n-element input sequence, and you know in advance that it is partly sorted in the following sense. Each element initially in position i such that $i \mod 4 = 0$ is either already in its correct position, or it is one place away from its correct position. For example, you know that after sorting, the element initially in position 12 belongs in position 11, 12, or 13. You have no advance information about the other elements, in positions i where $i \mod 4 \neq 0$. Show that an $\Omega(n \lg n)$ lower bound on comparison-based sorting still holds in this case.

Solution:

To get a permutation, place each of the $i \mod 4 = 0$ elements; there are $3^{n/4}$ ways to do so. Now you can place each of the remaining $3n/4$ items in any order in the remaining places, so that there are $3^{n/4}(3n/4)!$ possible sorted orders and $3^{n/4}(3n/4)!$ leaves in the decision tree. The height of this decision tree is at least $\lg(3^{n/4}(3n/4)!)$, which is $\Omega(n \lg n)$.

2. Sorting with a Range (attributed to CLRS Exercise 8.3-5)

   Show how to sort n integers in the range 0 to $n^3 - 1$ in $O(n)$ time.

Solution:

Treat the numbers as 3-digit numbers in radix n. Each digit ranges from 0 to $n - 1$. Sort these 3-digit numbers with radix sort.

There are 3 calls to counting sort, each taking $\Theta(n+n) = \Theta(n)$ time, so that the total time is $\Theta(n)$.

3. Worst Case for Bucket Sort (attributed to CLRS Exercise 8.4-2)

   Explain why the worst-case running time for bucket sort is $\Theta(n^2)$. What simple change to the algorithm preserves its linear average-case running time and makes its worst-case running time $O(n \lg n)$?

Solution:

The worst-case running time for the bucket-sort algorithm occurs when the assumption of uniformly distributed input does not hold. If, for example, all the input ends up in the first bucket, then in the insertion sort phase it needs to sort all the input, which takes $\Theta(n^2)$ time in the worst case.

A simple change that will preserve the linear expected running time and make the worst-case running time $O(n \lg n)$ is to use a worst-case $O(n \lg n)$-time algorithm, such as merge sort or heapsort, instead of insertion sort when sorting the buckets.