Tutorial 5 Exercise Solutions

COMP526: Efficient Algorithms

3–4 November, 2024

Exercise 1. Suppose an array a is "almost sorted" in the sense that if a stores the values $c_0 \leq c_0 \leq c_3 \leq \cdots \leq c_{n-1}$, then $c_i = a[j]$ where $\left|j - i\right| \leq k$. That is, the final (sorted) index of each value in a is no more than k from its initial index in a. Argue that on input a, the InsertionSort algorithm will terminate after at most $O(nk)$ steps.

```
procedure InsertionSort(a, n)
    for i = 1, 2, ..., n − 1 do
        j ← i
        while j > 0 and a[j] < a[j − 1] do
            Swap(a, j, j − 1)
            j ← j − 1
        end while
    end for
end procedure
```

Solution. Consider the case where a satisfies the condition stated in the exercise description: every element in a is within distance k of its correct sorted position. In particular, this means that the k smallest elements in a are initially stored in $a[1 \ldots 2k]$. After $2k$ iterations of the outer for loop of InsertionSort, the first k elements of a are sorted using $O(k^2)$ operations. Similarly, after k more iterations, the next k elements are sorted (using another $O(k^2)$ operations). Arguing in this way, we find that every k iterations of the outer loop, the next k elements are sorted using $O(k^2)$ operations. Thus all elements are sorted after n/k "rounds," each consisting of $O(k^2)$ operations. Therefore, the total number of operations performed by InsertionSort is $(n/k)O(k^2) = O(kn)$. □

Exercise 2. Suppose we are given two arrays a and b of size n that store two distinct permutations of $\{1, 2, \ldots, n\}$. That is, both a and b store each of the numbers from 1 to n, but the two arrays differ in their values at at least one index. Consider a sequence of swap operations $S_1, S_2, \ldots, S_m$ that are applied to both a and b, where each S_i swaps the values at two indices of the array it is applied to. Argue that after performing the swap operations, a and b are still distinct. In particular, the same sequence of swaps cannot sort both arrays.

Solution. We claim that after each swap operation, we have $a \neq b$. That is, there exists some index j_i such that $a[j_i] \neq b[j_i]$. We argue by induction on m, the number of swaps applied.

For the base case, $m = 0$, no swaps are applied, so let j_0 be an index where $a[j_0] \neq b[j_0]$. The index j_0 exists by the assumption that a and b are distinct.

For the inductive step, suppose that a and b are distinct after performing that m swaps, and they differ at index j_m. Consider their state after performing another swap S_{m+1}. If S_{m+1} does not swap the value at index j_m with another value, then after performing S_{m+1}, we still have $a[j_m] \neq b[j_m]$, as these values did not change. Thus in this case, we can take $j_{m+1} = j_m$. On the other hand, suppose S_{m+1} swaps the values of a at indices j_m and another index i_m. Then after the swap, we will have $a[i_m] \neq b[i_m]$. Thus we can take $j_{m+1} = i_m$ in this case. As a and b are distinct after applying S_{m+1} in either case, the claim follows by induction. □

Exercise 3. Suppose we apply RADIXSORT to an array a of size n that stores n distinct numerical values. Explain why in this scenario the running time of RADIXSORT is $\Omega(n \log n)$.

Solution. Recall that the running time of RADIXSORT is $\Theta(Bn)$, where B is the number of bits used to represent each value. Thus, it suffices to show that $B = \Omega(\log n)$ in this scenario. The number of *distinct* values that can be represented with B bits is 2^B. Since a stores n distinct values, we have $2^B \geq n$. Taking the log base two of both sides of this expression gives $B \geq \log n$, which gives the desired result. □

Exercise 4. Suppose a function T satisfies the recursion relation

$$T(n) = T(cn) + O(n) \quad \text{for some } c \text{ satisfying } \frac{1}{2} \leq c < 1.$$

for $n \geq 1$, and $T(1) = O(1)$. Argue that $T(n) = O(n)$. You may find the following fact useful: for any value of $a < 1$, we have

$$\sum_{i=0}^{k} a^i = 1 + a + a^2 + \cdots, + a^k = \frac{1 - a^{-k-1}}{1 - a} < \frac{1}{1 - a}. \tag{1}$$

Solution. To start, it is helpful to rewrite the hypothesis as $T(n) \leq T(cn) + bn$ for some constant b. Since this formula holds for all n, we can apply the formula recursively:

$$\begin{aligned}
T(n) &\leq T(cn) + bn \\
&\leq (T(c^2 n) + cbn) + bn \\
&= T(c^2 n) + (1 + c)bn \\
&\leq (T(c^3 n) + c^2 bn) + (1 + c)bn \\
&= T(c^3 n) + (1 + c + c^2)bn.
\end{aligned}$$

Continuing in this way, we find that after applying the recursive bound k times, we obtain

$$T(n) \leq T(c^k n) + bn \sum_{i=0}^{k} c^i. \tag{2}$$

More formally, we can prove () by induction.

In order to apply the base case, we should have $c^k n = 1$, or equivalently, $n = c^{-k}$. Taking the $\log_c$ of both sides gives $k = -\log_c(n) = \log(n)/\log(1/c)$. Using $k = \log(n)/\log(1/c)$. Using this value of k and applying (1), we find that

$$T(n) < T(0) + bn\frac{1}{1-c} = O(n)$$

which is the desired result. □