SPRING 2017: **COT 5407** Intro. to Algorithms

Sample Solutions

Honesty statement:

I have adhered to the collaboration policy for this class. In other words, everything written down in this submission is my own work. For problems where I received any help, I have cited the source, and/or named the collaborator.

Signed and Acknowledged, Johny B Good

1. (**Regular**) We discussed the **invariant** for SelectionSort in class. You can find the pseudocode and invariants for InsertionSort and MergeSort in Chapter 2 of your text (Cormen, et al.) [CLRS]. Write down precise **invariants** for BubbleSort (see p40 of text) and Merge (not MergeSort) (see p31 of text).

   Solution 1: Invariant for Merge can be found on p32 of [CLRS]. It states:

   > At the start of each iteration of the for-loop of lines 12 through 17, the subarray $A[p..(k-1)]$ contains the $k-p$ smallest elements of $L[1..(n_1+1)]$ and $R[1..(n_2+1)]$ in sorted order. Moreover, $L[i]$ and $R[j]$ are the smallest elements of their arrays that have not been copied back into A.

   The proof (not asked for in the problem) is in the text p32-34.

   Invariant for BubbleSort (pseudocode on p40 of [CLRS]) is:

   > At the start of each iteration of the for-loop for i (lines 1 through 4), the subarray $A[1..(i-1)]$ contains the $i-1$ smallest elements of A in sorted order.

   Proof is not required for this problem. Proof requires you to show **Initialization** at start of iteration $i = 1$ (trivial!) and **Maintenance**, i.e., at start of arbitrary iteration i. A more detailed invariant is possible by adding something about the for-loop for j on lines 2-4.

Always start a fresh problem on a fresh page. Since you do not need to print out your solutions, we will not be killing any trees in the process.

3. (**Regular**) Most sorting algorithms work correctly even if all items are not unique, i.e., there are repeats. A sorting algorithm is called **stable** if numbers with the same value appear in the output array in the same order as they do in the input array. Which of the algorithms SelectionSort, InsertionSort and BubbleSort are stable? If they are not stable give a small example with at most 4 items to prove your answer.

   Solution 3: Consider an input array A with 3 items $[2_1, 2_2, 1]$ in that order. The array has two copies of the number 2. In order to distinguish them we label them as 2_1 and 2_2, where 2_1 appears before 2_2 in the input array. If you apply algorithm XxxSort (refer to pseudocode on page XX of [CLRS]) then because of $\cdots$ and line `mm-nn` in pseudocode, the output would be $\cdots$ proving that XxxSort is not a stable sorting algorithm.

X. (**Regular**) Design an algorithm to $\cdots$

Solution X:

Basic Idea: The basic idea behind my solution is the following observation. We know that $\cdots$. Thus $\cdots$. However, $\cdots$. In order to take care of this problem, we do the following $\cdots$. The following pseudocode for FictionFlight provides the details of the idea.

Pseudocode:

Algorithm 1 FictionFlight

```
Assume bird starts at (r_R, c_R) of grid
s ← length of command seq S
D[0, 0, r_R, c_R] = TRUE
(prevR, prevC) = (r_R, c_R)
for t = 1 to s do
    D[0, t, Move(s[t], prevR, prevC)] = TRUE
    (prevR, prevC) = MOVE(s[t], prevR, prevC)        ▷ This trick is explained above
end for
for e = 1 to numRows × numCols do
    for t = 1 to s do
        for r = 1 to numRows do
            for c = 1 to numCols do
                D[e, t, r, c] = D[e − 1, t − 1, r, c] or D[e − 1, t − 1, NGBR(r, c)]
            end for
        end for
    end for
    if (D[e, s, r_E, c_E) = TRUE) then
        Return e
    end if
end for
```

Correctness: This is not a formal proof of correctness. However, the algorithm must be correct because it uses the Xx algorithm from the book (page Zz) and algorithm MOVE (discussed in Lec W of class) with a minor modification, and hence must be correct.

Time Complexity Analysis: The pseudocode suggests that the time complexity can be written down using the following recurrence:

$$T(n) = 2T(n/2) + O(n)$$

Solving this recurrence using case 2 of the Master Theorem gives us

$$T(n) = \Theta(n \log n)$$