Exercises

1. **Recursive Insertion Sort!** In this exercise, you’ll write a pseudocode algorithm for a recursive version of Insertion sort, a different way of expressing the same underlying algorithmic idea as the iterative version from class.

   We’ll do this in two parts, ending up with an algorithm to sort LLists of numbers (i.e., using the LList data structure from class for the sequence to be sorted). For this exercise, sorting is taken to mean in non-decreasing order.

   (a) Write a recursive `LLInsert` algorithm that inserts a number `x` in the proper location in a sorted LList `L`.

   ```
   # Input: Number x and sorted LList L = [a_0, a_1, ..., a_n],
   # where a_0 ≤ a_1 ≤ ... ≤ a_n
   # Output: List L' = [b_0, b_1, ..., b_{n+1}] containing input x and the
   # n + 1 elements of L, in sorted order
   # b_0 ≤ b_1 ≤ ... ≤ b_{n+1}
   ```

   (b) Using the `LLInsert` function, write a recursive `LLInsertionSort` algorithm that takes an LList `L` of numbers, possibly unsorted, and returns a sorted LList `L'` with the same elements as `L` in sorted order, consistent with the specification of the sorting problem.

   ```
   # Input: LList L = [a_0, a_1, ..., a_n]
   # Output: List L' = [b_0, b_1, ..., b_n] containing exactly the
   # elements of L, in sorted order b_0 ≤ b_1 ≤ ... ≤ b_n
   ```

   As usual, explain your algorithms and give correctness arguments.
2. Consider this pseudocode algorithm for the sorting method *Selection Sort*. (The specification for this sorting algorithm is the same given in class for Insertion Sort, and for Bubble Sort.)

\begin{verbatim}
SELECTIONSORT(A[1...n])
    for i = 1 to length[A] - 1
        min = i
        for j = i + 1 to length[A]
                min = j
        // the next 3 lines swap A[i] and A[min], using a temporary variable
        temp = A[i]
        A[min] = temp
\end{verbatim}

Given the following proposed loop invariant for the outer for loop of `SELECTIONSORT`, give a correctness proof for the algorithm (including a proof of the validity of the invariant).

**Proposed loop invariant for `SELECTIONSORT`:**

Subarray `A[1..i-1]` contains the `i-1` smallest elements of `A` in sorted order, and `A[i..n]` consists of the remaining values of `A` (no constraint on order).