1. As all for loop algorithms, Algorithm 1.2 (p.7) can be written as a while loop. Write such a loop \(L\): \(\text{while } C \text{ do } B\) and show an invariant \(I\) useful in proving correctness of the algorithm: \(\langle P \rangle L \langle Q \rangle\), where \(P\) is "\(i = 0 \land \text{currentMax} = A[0]\)" and \(Q\) is "currentMax is the maximum value stored in \(A\)". Prove the correctness of the while loop by proving the following three conditions:

   (i) \(P \Rightarrow I\)
   (ii) \(\langle C \land I \rangle B \langle I \rangle\)
   (iii) \((\neg C \land I) \Rightarrow Q\)

   where \(\Rightarrow\) is logical implication, \(\land\) is conjunction ("and"), and \(\neg\) is negation ("not").

2. Given an array \(p[1..n]\), the following algorithm computes an array \(h[1..n]\) of values defined as \(h[i] = \max\{j: 0 < j < i \text{ and } p[j] > p[i]\}\):

   \[
i \leftarrow 1
   \text{while } i \leq n \text{ do }
   \quad j \leftarrow i - 1
   \quad \text{while } j > 0 \text{ and } p[j] \leq p[i] \text{ do }
   \quad \quad j \leftarrow h[j]
   \quad \text{end while}
   \quad h[i] \leftarrow j
   \quad i \leftarrow i + 1
   \text{end while}
\]

   (a) Prove the algorithm’s correctness by stating invariants of the inner and outer loops and showing their properties (i)-(iii).

   (b) Show that the worst case of complexity of inner loop is \(\Theta(n)\).

   (HINT: To prove the upper bound, explain why the loop could never require more than a linear number of steps. To prove the lower bound, describe a worst-case input that requires \(\Omega(n)\) steps.)

   (c) Use the credit invariant method to prove that the total running time of the algorithm is \(O(n)\).

   (HINT: In the \(i\)th iteration of the outer loop, assign one or more credits to the array position \(h[i]\). Think about how those credits will be spent, and then prove that those credits will be available when you need them.)
3. Consider the following implementation of a Queue (FIFO) by two Stacks (LIFO), First and Second:

**Algorithm 1** enqueue\(x\): adds item \(x\) to the Queue
\[
\text{First.push}(x)
\]

**Algorithm 2** dequeue(): remove and return oldest item in the Queue
\[
\text{if } \neg \text{Second.isEmpty()} \text{ then} \\
\quad \text{return Second.pop()} \\
\text{else} \\
\quad \text{while } \neg \text{First.isEmpty()} \text{ do} \\
\quad \quad \text{Second.push(First.pop())} \\
\quad \text{end while} \\
\quad \text{return Second.pop()} \\
\text{end if}
\]

(a) Use a loop invariant to prove that dequeue() correctly returns the oldest item in the Queue. (HINT: Use the value \(\text{TimeStamp}(x)\) indicating when \(x\) is inserted into the Queue.)

(b) What is the worst case time complexity of the dequeue operation? (Assume that push, pop, and isEmpty take constant time.)

(c) Prove that enqueue and dequeue both run in amortized linear time using the potential function method.