IMPERATIVE PROGRAMMING 2017

MOISE GABRIEL

QUESTION 3

```
class Bag {
  private var list = new Bag.Node(0, null) // dummy header
  private var end = list
  def add(n1: Bag.Node): Unit = {
      n1.next = null
      end.next = n1
  }   end = n1

  // (a)
  def findMin: Bag.Node = {
      var current = list.next
      var minNode = current
      var min = current.datum
      if (current.next == null) return current // just one node in the list
      else current = current.next
      // Invariant i: minNode.datum = min(L(list.next, current))
      while (current != null)
      { if (min > current.datum) {min = current.datum; minNode = current}
        current = current.next
      }
      // current = null => minNode.datum = min(L(list.next, null)), so we return minNode
      minNode
  }
  // (b)
  def remove (n: Bag.Node): Unit = {
      var current = list.next
      var prev = list
      // Invariant i: n.datum is not in L(list.next, prev) && prev.next = current
      while (n != current) {prev = prev.next; current = current.next}
      // n == current, so now we want to delete current from the list
      prev.next = prev.next.next
  }
```

```
    // (c) The dummy header helps us get rid of the special case when we need to delete the
first node of the list
    def delMin : Bag.Node = {
        var minNode = this.findMin
        remove (minNode)
        minNode
    }
    // (d)
    def sort : Bag = {
        var sortedList = new Bag
        var k=0 // used for the invariant and to keep track of how many elements there
are in the list
        // invariant i: sortedList contains the first k elements of the list, sorted increasingly
        while (list.next != null) // as long as the list is non-empty
        { var minNode = this.delMin
          sortedList.add (minNode)
          k += 1
        }
        sortedList
    }
}
// Companion object
object Bag {
    class Node (var datum : Int, var next : Node)
}
```