```java
// Project 2: Sort
// Many headaches came from this project.
// Andrea Smith
// CSCI 1913

//  SORT. Sort a linear singly-linked list of INTs.

class Sort
{

//  NODE. A node in a linear singly linked list of INTs.

  private static class Node
  {
    private int  number;  //  The INT in the node, duh.
    private Node next;    //  The NODE that follows this one, or NULL.

//  Constructor. Initialize a new NODE with NUMBER and NEXT.

    private Node(int number, Node next)
    {
      this.number = number;
      this.next = next;
    }
  }

//  MAKE NODES. Return a list of NODEs that contains INTs from NUMBERS in order
//  of their appearance.

  private static Node makeNodes(int ... numbers)
  {
    if (numbers.length > 0)
    {
      Node first = new Node(numbers[0], null);
      Node last  = first;
      for (int index = 1; index < numbers.length; index += 1)
      {
        last.next = new Node(numbers[index], null);
        last = last.next;
      }
      return first;
    }
    else
    {
      return null:
    }
  }

//  WRITE NODES. Write the INTs from a list of NODEs in paired square brackets,
//  separated by commas, with a newline at the end.

  private static void writeNodes(Node nodes)
  {
    System.out.print('[');
    if (nodes != null)
    {
      System.out.print(nodes.number);
      nodes = nodes.next;
      while (nodes != null)
      {
        System.out.print(", ");
        System.out.print(nodes.number);
        nodes = nodes.next;
      }
    }
    System.out.println(']');
  }

//  SORT NODES. Sort UNSORTED, a list of NODEs, into nondecreasing order of its
//  NUMBER slots, without making new NODEs.

  private static Node sortNodes(Node unsorted)
  {
    if (unsorted == null || unsorted.next == null)
    {
      return unsorted; // The list is already sorted
    }
    else
    {
      int step = 1;
      Node right = null;
      Node left = null;
      Node leftTemp = null;
      Node rightTemp = null;

        while (unsorted != null)
        {
            if (step % 2 == 0) // EVEN STEPS CASE
            {
              rightTemp = right; // holds the whole list for nex
```

```java
          rightTemp = right; // holds the whole list for now
          right = unsorted;
          unsorted = unsorted.next; // unsorted.next is everything to the right of the first int, so this deletes first int in unsorted
          right.next = rightTemp; // Add that number to right
          rightTemp = unsorted;
          step++;

        }
        else // ODD STEPS CASE, same as EVEN but left
        {
          leftTemp = left;
          left = unsorted;
          unsorted = unsorted.next;
          left.next = leftTemp;
          leftTemp = unsorted;
          step++;
        }
    }
    // SORTING
    right = sortNodes(right);
    left = sortNodes(left);
    Node sorted = null;
    Node end = null;
    Node temp = null;

    // COMBINING (and also kind of sorting)

    // Deals w special case where list is empty
    if (left != null && right != null) // continue til left and right are empty
    {
      if (left.number <= right.number) // Delete from left, add to end of sorted
      {
        sorted = left; // Sorted has all the values so they can be iterated thru again
        end = left;
        temp = left.next;
        left = temp;      // First number deleted from left
        end.next = null; // Everything to the right of the number deleted
      }

      else // Same here but Delete from right, add to end of sorted
```

```
        {
          sorted = right;
          end = right;
          temp = right.next;
          right = temp;
          end.next = null;
        }
      }

      // Only executes when both lists have something in them, then tacks on whatever is left with the if statements after the loop
        while (left != null && right != null)
        {
          if (left.number <= right.number)
          {
            end.next = left;
            end = end.next;
            temp = left.next;
            left = temp;
            end.next = null;
          }
          else
          {
            end.next = right;
            end = end.next;
            temp = right.next;
            right = temp;
            end.next = null;
          }
        }

      // If one of the sides is left sorted but nonempty, add the entire thing to the end of sorted
      if (left != null)
      {
        end.next = left;
      }
      else if (right != null)
      {
        end.next = right;
      }

  return sorted;
  }
}
```

```java
//  MAIN. Run some examples. The comments show what must be printed.

  public static void main(String [] args)
  {
    writeNodes(sortNodes(makeNodes()));      //  []
    writeNodes(sortNodes(makeNodes(1)));     //  [1]
    writeNodes(sortNodes(makeNodes(1, 2)));  //  [1, 2]
    writeNodes(sortNodes(makeNodes(2, 1)));  //  [1, 2]

    writeNodes(sortNodes(makeNodes(5, 8, 4, 9, 1, 2, 3, 7, 6)));
    // [1, 2, 3, 4, 5, 6, 7, 8, 9]

    writeNodes(sortNodes(makeNodes(9, 8, 7, 6, 5, 4, 3, 2, 1)));
    //  [1, 2, 3, 4, 5, 6, 7, 8, 9]

    writeNodes(sortNodes(makeNodes(3, 1, 4, 5, 9, 2, 6, 8, 7)));
    //  [1, 2, 3, 4, 5, 6, 7, 8, 9]
    writeNodes(sortNodes(makeNodes(420, 69, 96, 0)));
    // [0, 69, 96, 420]
    writeNodes(sortNodes(makeNodes(80085, 3, 6, 9)));
    // [3, 6, 9, 80085]
    writeNodes(sortNodes(makeNodes(38, 17)));
  }
}
```