```java
import java.util.ArrayList;

/**
 * Ian Sulley
 *
 * Honor Code Statement:
 * I affirm that I have carried out the attached academic
 * endeavors with full academic honesty, and in accordance with
 * the Union College Honor Code and the course syllabus.
 *
 *
 *
 * Finds the minimum element and the index of the minimum element in a ArrayList of Strings
 *
 * Sorts an ArrayList of Strings too! (using selection sort)
 *
 */
public class ListProcessor
{
    /**
     * Swaps elements i and j in the given list.
     */
    private void swap(ArrayList<String> aList, int i, int j)
    {
        String tmp = aList.get(i);
        aList.set(i, aList.get(j));
        aList.set(j, tmp);
    }

    /**
     * Finds the minimum element of a list and returns it.
     * Non-destructive (That means this method should not change aList.)
     *
     * @param aList the list in which to find the minimum element.
     * @return the minimum element of the list.
     */
    public String getMin(ArrayList<String> aList)
    {
        int startIndex = 0;
        String startElement = aList.get(startIndex);

        return getMinR(aList,startElement, startIndex);
    }

    /**
     * Recursive function for finding the minimum element
```

```java
in the list
     * @param aList list being examined
     * @param minElement current minimum element
     * @param index of aList currently being checked
     * @return minimum element in the list
     */
    private String getMinR(ArrayList<String> aList, String minElement, int index) {
        if (index == aList.size()) {
            return minElement;
        }
        else {
            if (aList.get(index).compareTo(minElement) < 0) {
                return getMinR(aList, aList.get(index), index+1);
            } else {
                return getMinR(aList, minElement, index+1);
            }
        }
    }

    /**
     * Finds the minimum element of a list and returns the index of that
     * element. If there is more than one instance of the minimum, then
     * the lowest index will be returned.  Non-destructive.
     *
     * @param aList the list in which to find the minimum element.
     * @return the index of the minimum element in the list.
     */
    public int getMinIndex(ArrayList<String> aList)
    {
        int minIndex = 0;
        int startIndex = 0;
        String startElement = aList.get(startIndex);

        return getMinIndexR(aList,startElement, startIndex, minIndex);
    }

    /**
     *  recursively finds the index of the the minimum value in the ArrayList
     * @param aList list being searched
     * @param minElement current minimum element found
     * @param currentIndex current index of the arraylist
```

```java
we are on
     * @param minIndex the index of the minimum element found
     * @return
     */
    private int getMinIndexR(ArrayList<String> aList, String minElement, int currentIndex, int minIndex) {
        if (currentIndex == aList.size()) {
            return minIndex;
        }
        else {
            if(aList.get(currentIndex).compareTo(minElement) < 0){
                return getMinIndexR(aList, aList.get(currentIndex), currentIndex+1, currentIndex);
            }
            else{
                return getMinIndexR(aList, minElement, currentIndex+1, minIndex);
            }
        }
    }


    /**
     * Sorts a list in place. I.E. the list is modified so that it is in order.
     *
     * @param aList: the list to sort.
     */
    public void sort(ArrayList<String> aList)
    {
        int startIndex = 0;

        sortR(aList, startIndex);




    }

    /**
     * recursively sorts the array list using the selection sort method
     * @param aList list being sorted
     * @param startIndex where the sublist of aList begins, so we can find the minimum of the sublist
     */
    private void sortR(ArrayList<String> aList, int startIndex){
        if (aList.size() == startIndex) {
            return;
        }
        else {
            ArrayList<String> currentList = new ArrayList<String>(aList.subList(startIndex, aList.size()));
            int minIndex = getMinIndex(currentList) + startIndex;
            swap(aList, minIndex, startIndex);
            sortR(aList, startIndex+1);
        }
    }
}



```