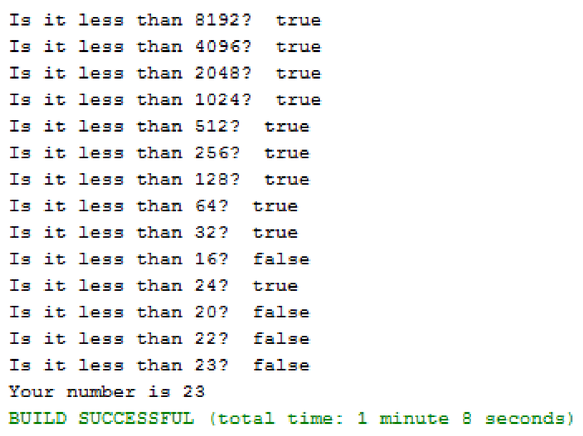
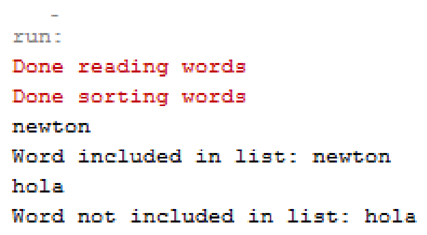
Activity 8

1. Develop an implementation of Questions.java that takes the maximum number `N` as command-line input. Prove that your implementation is correct.



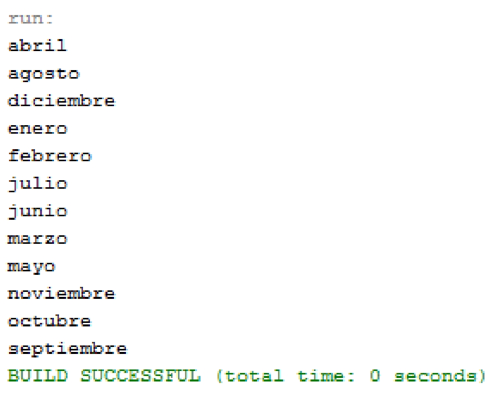
```
Is it less than 8192?  true
Is it less than 4096?  true
Is it less than 2048?  true
Is it less than 1024?  true
Is it less than 512?  true
Is it less than 256?  true
Is it less than 128?  true
Is it less than 64?  true
Is it less than 32?  true
Is it less than 16?  false
Is it less than 24?  true
Is it less than 20?  false
Is it less than 22?  false
Is it less than 23?  false
Your number is 23
BUILD SUCCESSFUL (total time: 1 minute 8 seconds)
```

2. Modify BinarySearch.java so that if the search key is in the array, it returns the smallest index `i` for which `a[i]` is equal to key, and otherwise, it returns `-i`, where `i` is the smallest index such that `a[i]` is greater than key.



```
run:
Done reading words
Done sorting words
newton
Word included in list: newton
hola
Word not included in list: hola
```

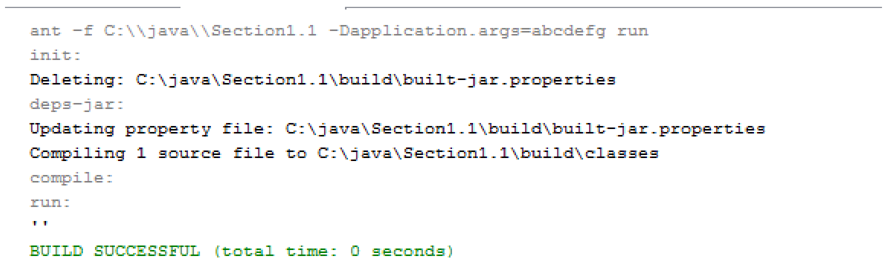
3. Write a program Dedup.java that reads strings from standard input and prints them on standard output with all duplicates removed (in sorted order).



```
run:
abril
agosto
diciembre
enero
febrero
julio
junio
marzo
mayo
noviembre
octubre
septiembre
BUILD SUCCESSFUL (total time: 0 seconds)
```

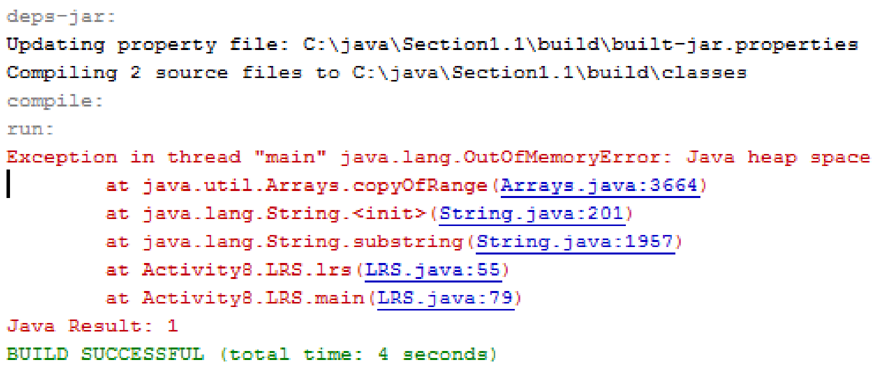
4. Add code to LRS.java to make it print indices in the original string where the longest repeated substring occurs.

Adriana Lucia Alvarez Brito



```
ant -f C:\\java\\Section1.1 -Dapplication.args=abcdefg run
init:
Deleting: C:\java\Section1.1\build\built-jar.properties
deps-jar:
Updating property file: C:\java\Section1.1\build\built-jar.properties
Compiling 1 source file to C:\java\Section1.1\build\classes
compile:
run:
''
BUILD SUCCESSFUL (total time: 0 seconds)
```

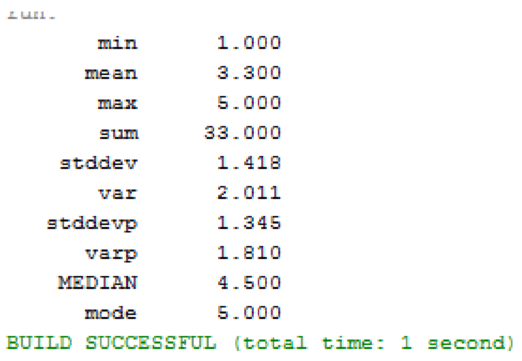
5. Find a pathological input for which LRS.java runs in quadratic time (or worse).



```
deps-jar:
Updating property file: C:\java\Section1.1\build\built-jar.properties
Compiling 2 source files to C:\java\Section1.1\build\classes
compile:
run:
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
        at java.util.Arrays.copyOfRange(Arrays.java:3664)
        at java.lang.String.<init>(String.java:201)
        at java.lang.String.substring(String.java:1957)
        at Activity8.LRS.lrs(LRS.java:55)
        at Activity8.LRS.main(LRS.java:79)
Java Result: 1
BUILD SUCCESSFUL (total time: 4 seconds)
```

6. **Median.** Add to StdStats.java a method `median()` that computes in linearithmic time the median of a sequence of *N* integers.

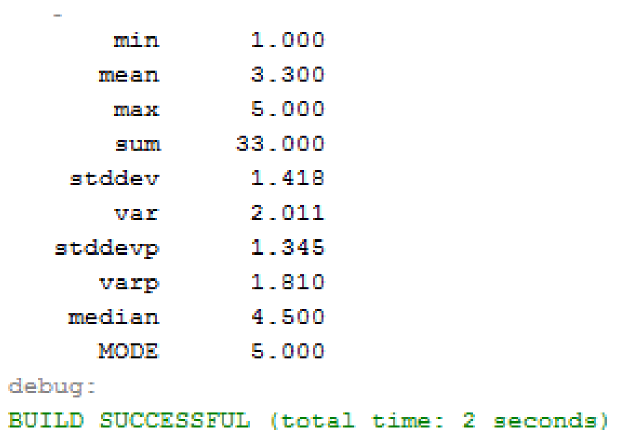
   Hint: reduce to sorting.



```
run:
       min       1.000
      mean       3.300
       max       5.000
       sum      33.000
    stddev       1.418
       var       2.011
   stddevp       1.345
      varp       1.810
    MEDIAN       4.500
      mode       5.000
BUILD SUCCESSFUL (total time: 1 second)
```

7. **Mode.** Add to StdStats.java a method `mode()` that computes in linearithmic time the mode (value that occurs most frequently) of a sequence of *N* integers.

   Hint: reduce to sorting.



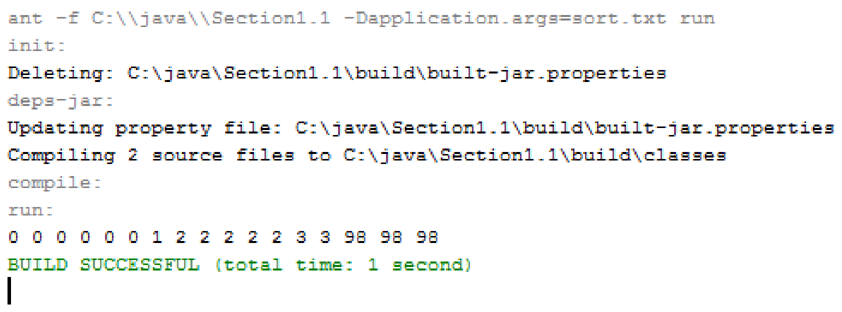
```
       -
          min        1.000
         mean        3.300
          max        5.000
          sum       33.000
       stddev        1.418
          var        2.011
      stddevp        1.345
         varp        1.810
       median        4.500
         MODE        5.000
debug:
BUILD SUCCESSFUL (total time: 2 seconds)
```

8. **Integer sort.** Write a *linear*-time filter IntegerSort.java that takes from standard input a sequence of integers that are between 0 and 99 and prints the same integers in sorted order on standard output. For example, presented with the input sequence

   ```
   98 2 3 1 0 0 0 3 98 98 2 2 2 0 0 0 2
   ```

   your program should print the output sequence

   ```
   0  0 0 0 0 0 1 2 2 2 2 2 3 3 98 98 98
   ```

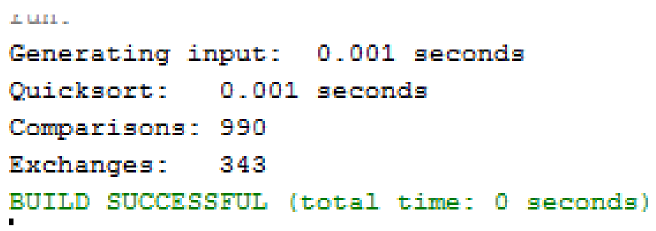


```
ant -f C:\\java\\Section1.1 -Dapplication.args=sort.txt run
init:
Deleting: C:\java\Section1.1\build\built-jar.properties
deps-jar:
Updating property file: C:\java\Section1.1\build\built-jar.properties
Compiling 2 source files to C:\java\Section1.1\build\classes
compile:
run:
0 0 0 0 0 0 1 2 2 2 2 2 3 3 98 98 98
BUILD SUCCESSFUL (total time: 1 second)
```

9. **Quicksort.** Write a recursive program QuickSort.java that sorts an array of randomly ordered distinct `Comparable` elements.

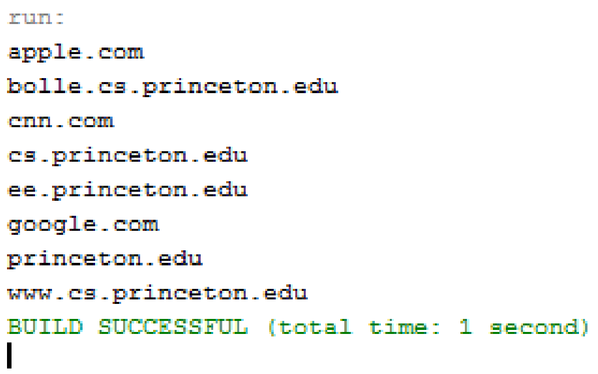
   Hint: Use a method like the one described in the previous exercise. First, partition the array into a left part with all elements less than *v*, followed by *v*, followed by a right part with all elements greater than *v*. Then, recursively sort the two parts.

   Extra credit: Modify your method (if necessary) to work properly when the elements are not necessarily distinct.



```
run:
Generating input:  0.001 seconds
Quicksort:   0.001 seconds
Comparisons: 990
Exchanges:   343
BUILD SUCCESSFUL (total time: 0 seconds)
```

10. **Reverse domain.** Write a program to read in a list of domain names from standard input, and print the reverse domain names in sorted order. For example, the reverse domain of `cs.princeton.edu` is `edu.princeton.cs`. This computation is useful for web log analysis. To do so, create a data type Domain.java that implements the `Comparable` interface, using reverse domain name order.



```
run:
apple.com
bolle.cs.princeton.edu
cnn.com
cs.princeton.edu
ee.princeton.edu
google.com
princeton.edu
www.cs.princeton.edu
BUILD SUCCESSFUL (total time: 1 second)
```