CS61B, Fall 2011 **HW #9** **P. N. Hilfinger**

Due: Mon., 14 November 2011

We have skeleton files for your solutions available in `~cs61b/code/hw9` and also in the Subversion repository. Place answers to non-programming questions in the file `hw9.txt`.

1. The file HashTesting.java contains various routines and classes for testing and timing hash tables. Compile this file in your directory. The file contains a wrapper class String1, which simply contains a String and returns it via the toString method. The .equals method on this class compares the Strings in the two comparands. The hashCode method is an adaptation of that used in the real String class. It gives you the ability to "tweak" the algorithm by choosing to have the hashCode method look only at some of the characters. Take a look at class String1, and especially at its hashCode method.

The test

```
java HashTesting test1 $MASTERDIR/lib/words 1
```

will read a list of about 100000 words from `$MASTERDIR/lib/words` (a small dictionary taken from an Ubuntu distribution), store them in a hash table (the Java library class HashSet), and then check that each is in the set. It times these last two steps and reports the time.

The argument 1 here causes it to use the same hashing function for the strings as Java normally does for java.lang.String. If you run

```
java HashTesting test1 $MASTERDIR/lib/words 2
```

the hash function looks only at every other character, presumably making it faster to compute.

Try this command with various values of the second parameter. Explain why the timings change as they do in hw9.txt.

2. The command

```
java HashTesting test2 N
```

for N an integer, will time the storage and retrieval of N^2 four-letter words in a hash table. These words all have the form $xxyy$, where the character codes for x and y vary from 1 to N (most of these "words" won't be readable). The command

```
java HashTesting test3 N
```

does the same thing, but the "words" have the form xXyY, where x and y vary as before, and $Y = 2^{16} - 31y - 1$, $X = 2^{16} - 31x - 1$.

Run these two commands with various values of N (start at 20). Explain in as much detail as you can the reasons for the relative timing behavior of these tests (that is, why test3 takes longer than test2), putting your answer in hw9.txt.

3. The class FoldedString is another wrapper class for Strings whose `.equals` and `.compareTo` methods ignore case (e.g., they treat "foo," "Foo," and "FOO" as equivalent). The command

```
java HashTesting test4 the quick brown fox
```

will treat the trailing command-line arguments ("the", "quick", etc.) as FoldedStrings, and insert them into a HashSet and (for comparison) a TreeSet, which uses a balanced binary search tree to store its data. The program will then check that it can find the all-upper-case version of each of the words in these sets (since they are supposed to compare equal).

Try running test4 as shown. HashSet fails to find the words entered into it, when they are upper-cased, while the TreeSet seems to work just fine. Explain why in hw9.txt.

The problem is in the FoldedString class. Change it so that

```
java HashTesting test4...
```

works. Try to do so in a reasonable way: make sure that large HashSets full of FoldedStrings will continue to work well.

4. (First familiarize yourself with the routines in java.util.Collections (for Lists) and java.util.Arrays.) Complete the methods in SortingStuff.java. In each case, the body consists of a single method call (possibly nested) so as to cause the variable `sorted` to become a sorted List or array of items. The answers do not involve assignments to sorted; it is modified in place. You may also find the on-line documentation for the String class and the Comparator interface to be useful.

5. The file SortTesting.java performs a quicksort, but provides a parameter that you can "tweak." When you run

```
java SortTesting N K
```

for positive integers N and K, the program will run the quicksort algorithm on an array containing a pseudo-random permutation of the integers 0 to $N-1$, but will stop the recursion when the size of a subarray is $\leq K$ (that is, it will leave those subarrays unsorted). The program then uses a straight insertion sort to finish the sorting process. Finally, it prints timing statistics. Although the arrays it sorts are pseudo-randomly permuted, we have arranged that you get the same array each time for any particular value of N to make it easier to compare times.

Try running the program several times with arguments like this:

```
java SortTesting 4000 4000
java SortTesting 8000 8000
java SortTesting 16000 16000
java SortTesting 32000 32000
```

Explain the rate of increase in times in hw9.txt.

Now try running the program with arguments like this:

```
java SortTesting 32000 1
java SortTesting 1000000 1
java SortTesting 2000000 1
java SortTesting 4000000 1
java SortTesting 8000000 1
```

Explain the difference between these results and the ones before in hw9.txt.

Finally, try running the program with the following sequence:

```
java SortTesting 2000000 1
java SortTesting 2000000 5
java SortTesting 2000000 10
java SortTesting 2000000 15
<some more in here>
java SortTesting 2000000 100
```

Explain these results in hw9.txt.

6. In SortTesting.java, the recursive part of the quicksort procedure looks like this:

```
quicksort (A, L, M-1, K);
quicksort (A, M+1, U, K);
```

and the pivot method goes to some trouble to make sure that element M actually ends up containing the pivot value.

Suppose that we were to rewrite pivot so that it only guaranteed that $A[L..M-1]$ were all $< P$ and that $A[M..U]$ were all $\geq P$ (that is, we DON'T guarantee that A[M] == P). We would have to change the recursive calls to

```
quicksort (A, L, M-1, K);
quicksort (A, M, U, K);
```

Explain why the resulting program won't work in hw9.txt.

7. The file `RadixSort.java` provides the framework for a simple LSD radix sort program. Fill in the `sortByOneChar` method as indicated by its comment and test the resulting program (which is self-checking) with commands like:

```
java RadixSort $MASTERDIR/lib/words > /dev/null
```

to get just result-checking and timing, or

```
java RadixSort $MASTERDIR/lib/words > OUT
```

to actually write the results to a file.

8. The file `TrieSet.java` in the `hw9` staff code directory contains a skeleton for a set-of-Strings class For this exercise, we will *not* worry about compressing the nodes. Fill in the `contains` methods in the skeleton to comply with its comment. You will also have to pick a representation for `InnerTrieNode` and complete its constructor. Then fill in the `insert` methods to comply with their comments.