Computer Science 501
Data Structures & Algorithms
The College of Saint Rose
Fall 2015

Lab 7: The Two Towers Problem

Due: 6:00 PM, Wednesday, October 28, 2015, and 6:00 PM, Wednesday, November 4, 2015 (empirical study only)

You have two main tasks: to complete an empirical study of sorting algorithms, and to learn more about Java iterators by completing a program that iterates over subsets to solve an interesting problem. The empirical study is due in two weeks, while the iterator lab is due in one week.

You may work alone or in a group of up to four on this lab. There is a lot of work here, so everyone is encouraged to find some classmates to work together. Of course, collaboration with your partner is unrestricted. You may discuss the lab with your classmates and give and receive some help, but your submission must be your own work (or that of you and your teammates, if you choose to form a group).

Getting Set Up

To get your BlueJ environment set up for this week's lab assignment, start BlueJ and choose "New Project" from the "Project" menu. Navigate to your folder for this course and choose the name "Lab7" (no spaces) for the project.

Create a document where you will record your answers to the lecture assignment and lab questions. If you use plain text, call it "`lab7.txt`". If it's a Word document, you can call it whatever you'd like, but when you submit, be sure you convert it to a PDF document "`lab7.pdf`" before you submit it.

Lecture Assignment Questions

We will usually discuss these questions at the start of class on the lab due date, so no credit can be earned for late submissions of lecture assignment questions.

? LA Question 1:
Bailey Exercise 9.10, p. 213. (3 points)

? LA Question 2:
Bailey Exercise 9.12, p. 213. Note: *shuffled* data would be data where each element has an equal probability of being at each position. (5 points)

? LA Question 3:
Suppose you wish to determine the highest floor in an n-story building from which an object can fall without breaking. You have 2 such objects to experiment with, but once one breaks, it can no longer be used for subsequent tests. Design the most efficient algorithm you can to solve this problem and state its efficiency class. (5 points)

Sorting Algorithm Empirical Study

Your two week task is to complete an empirical study of sorting algorithms. Your analysis should include all combinations of these sorting algorithms:

- selection sort
- insertion sort
- merge sort
- quicksort using first element as pivot
- quicksort using random pivot
- quicksort using median of three to determine pivot

with these input data distributions:

- an array filled with n random integer values in the range 0 to $2n$
- an array filled with n integer values sorted in ascending order
- an array filled with n integer values sorted in descending order
- an array filled with n integer values "nearly" sorted in ascending order (specifically, about 90% of the values should start in their ultimate location)

Please note that this means you will be performing 24 case studies. As with the previous timing lab, automation of the gathering of timing statistics will be essential. Controlling your experiments with command-line parameters and managing multiple runs with scripts will make your task much more manageable.

Timings and Analysis

Use your program(s) to generate basic operation counts and timing data for each sorting algorithm on an appropriate range of data sizes and distributions. Compare your results with your expectations based on our efficiency analysis of these algorithms. If you find discrepencies, how might they be explained? Please include as many of the details of your test environment as you can: the type of processor or processors in the computer including clock speed, cache sizes, memory sizes, the operating system and version running on the computer, and the Java version you are using.

Tips, Tricks, Precautions, and Suggestions

- Be careful that you don't reuse an array of values for multiple runs, since all but the first could end up having already-sorted data as input.
- A simple tabular format of output will help you manage the creation of tables and/or graphs. Something like

  ```
  10000 selection random .034693
  ```

  might indicate for an input size of 10,000, using a selection sort on random input took .034693 seconds.
- The runs should vary the input array size for each combination of sorting algorithm and input data type. To get meaningful results, you want a pretty wide range of sizes. You might start with an array of size 1000 (or better yet 1024) and double the size until you have an input size of 1,000,000 (or better yet 1,048,576). Take the average or best times (and justify your choice) for some number of runs, probably a few dozen to a few hundred. Then, plot your results. Make sure your graphs have a meaningful title, legend, and axis labels. Then see if the numbers fit the expected behavior (*e.g.*, n^2 or $n \log n$). Also, make sure your graphs have a meaningful scale on each axis.
- Include your graphs and your analysis of them in your writeup. The raw numbers are useful, but should be submitted separately as part of a big table or spreadsheet, or possibly as an appendix. There are likely to be too many numbers for anyone to want to look at them all.

Include the source code for all programs and scripts you used to generate your timings.

Each case study in your writeup is worth 2 points, which will include your raw timing and operation count data, the graphs of your timing results and operation counts, and your analysis of the results, including a comparison with your theoretical expectations. When different operations have different complexities, please be sure to address that.

The Two Towers Problem

The one-week part of your assignment is the laboratory at the end of Chapter 8 in Bailey. You will gain experience writing your own generally-useful `Iterator` class and use it to solve an interesting problem.

Notes and Hints

When you implement your `SubsetIterator` that `extends AbstractIterator`, be sure to import `structure5.*` and `java.util.Iterator` at the top of your file. Also remember that your `SubsetIterator` should use generic types. It is far from obvious how to accomplish this, so here is what your class header should look like for the `SubsetIterator`:

```
public class SubsetIterator<E,T extends Vector<E>> extends AbstractIterator<T>
```

During the implementation of your `SubsetIterator`'s `get` and `next` methods, you will need to create a `Vector<E>` and return it as a `T` to satisfy the required return types specified by `AbstractIterator<T>` for those methods. Java's syntax for this is again not entirely obvious. This should work:

```
  T subset = (T)new Vector<E>();
```

For testing, write a `main` method of your `SubsetIterator` that creates a `Vector<Integer>` with the `Integers` from 0 through 7, creates a `SubsetIterator<Integer,Vector<Integer>>` with this `Vector<Integer>`, and then prints out all subsets returned and a count of the subsets returned. Make sure you end up with 256 subsets printed. Please leave this `main` method in your program when you submit.

Write the `main` method to solve the two towers problem in a separate class called `TwoTowers`, proceeding as specified in the lab description in the text.

? Question 1:
Answer thought question 1 from Bailey p. 177. (1 point)

? Question 2:
Answer thought question 2 from Bailey p. 177. (2 points)

? Question 3:
What is the time complexity of this problem? Explain briefly. (1 point)

? Question 4:
How long does it take your program to find the answer to the 20-block problem? (hint: try the `time` command rather than instrumenting your source code with a timer). (1 point)

? Question 5:
Based on the time taken to solve the 20-block problem, about how long do you expect it would take to solve the 21-block problem? What is the actual time? (1 point)

? Question 6:
How long does it take to solve the 22-, 23-, 24-, and 25-block problems? (1 point)

? Question 7:
Do these agree with your expectations, given the time complexity of the problem? (1 point)

? **Question 8:**
Estimate the time that would be needed to solve the 40- and 60-block problems, and explain briefly how you arrived at these approximations. (You can try these if you're very patient, but for the purposes of this question, just estimate based on the run times of the smaller problems). (2 points)

Submitting

Before 6:00 PM, Wednesday, October 28, 2015, and 6:00 PM, Wednesday, November 4, 2015 (empirical study only), submit your lab for grading. There are two things you need to do to complete the submission: (i) Copy your file with the answers to the lecture assignment and lab questions into your project directory. Be sure to use the correct file name. If you prepared your answers in Word, export to a PDF file and submit that. (ii) Email a copy of your lab (a `.7z` or `.zip` file containing your project directory) to *terescoj@strose.edu*. Please use a meaningful subject line such as "Joe Student Lab7 Submission".

Grading

This assignment is worth 100 points, which are distributed as follows:

Feature	Value	Score
LA Question 1 (9.10)	3	
LA Question 2 (9.12)	4	
LA Question 3 (n-story building)	5	
General sorting algorithm test framework	5	
Java code for specific sorting algorithms	5	
Presentation and analysis of timing results	48	
"Two Towers" program design and style	2	
"Two Towers" program documentation	4	
"Two Towers" correctness: `SubsetIterator`	8	
"Two Towers" correctness: `TwoTowers`	6	
Question 1 (Thought Question 1)	1	
Question 2 (Thought Question 2)	2	
Question 3 (complexity)	1	
Question 4 (20-block time)	1	
Question 5 (21-block expectation and time)	1	
Question 6 (22- through 25-block times)	1	
Question 7 (complexity match?)	1	
Question 8 (40- and 60-block estimates)	2	
Total	100	