CIS 303 Algorithm Analysis and Design
p02-ProgBasicAnalysis Assignment: Review of Lists and Basic Algorithm Analysis

L. Grabowski

Learning Outcomes

After completing this assignments, students should be able to:

- Program concrete classes that implement a simplified `List` interface.
- Formulate a hypothesis about program performance based on prior experience.
- Conduct experiments that gather data about the time performance of the concrete classes.
- Write a lab report using LaTeX.

Directions:

- This is an individual experiment and analysis assignment. 60 points possible.
- The analysis must be typeset in LaTeX. See the template available in Brightspace to get started. Feel free to see me in office hours, or visit the CS tutors, if you need some help getting started with LaTeX.
- Download the source files from Gitea repo.
- Carefully read and follow the instructions for the analysis and report, below. Please note that the grading rubric for the assignment is available in the Gitea repo.

Methods

1. (5 points) **Background and hypothesis.** This assignment focuses on comparing the performance (running time) of `List` data structures that use different underlying implementations: an array and a linked list. **You may NOT use any Java collections for the assignment.** You are required to use the `List` interface, the `Link` class, and the `Fizzbin` object class that I provided for you.

   Follow these steps to formulate a hypothesis for your experiments.

   1. **Note: Formulate your hypothesis BEFORE you do the coding for the assignment!**
   2. *Gather background information.* To formulate your hypothesis, use what you know about arrays and linked lists in Java, as well as the idea of an abstract data type, specifically a `List` ADT. What is required to do the operations you must implement (`append, insert, delete, size, get`) using each of the underlying structures? What aspects of working with the different concrete implementations will require the most algorithmic "work"? That is, what will take time? Will one of the implementations be faster than the other in some operations, or even all the operations? Do these questions sound familiar? They should. There was a problem on your first assignment that asked you to consider the same questions.

3. *Hypothesis.* Using your reflections from your previous programming experience (and our first assignment), construct your hypothesis about the time performance of the two implementations – array and linked list – relative to each other. Remember that a hypothesis must be testable and verifiable/falsifiable. Be clear and specific in your language, and remember to directly tie the theoretical analysis of the structures' performance to the data that you will collect. In other words, what will your data show if it supports your hypothesis? It is good practice to also state your null hypothesis — what will the experimental results look like if your hypothesis is not supported? Remember that it is not enough to say something will be "faster" or "more efficient". Be specific, and state clearly how your data will show it's faster. Your hypothesis must connect to what you will measure in your experiments.

2. (30 points) **Program.** You will write concrete classes that implement a simplified `List` ADT. I have provided the following source code that you must use **without modification** for the assignment:

   - `Fizzbin.java`: a simple class that creates a data type, to be used to populate your `Lists`;
   - `Link.java`: a singly linked node class (from the textbook, with some modification);
   - `List.java`: an interface for a `List` ADT.

   You will be responsible for implementing the following:

   - Two different concrete classes that implement the provided `List` interface. One class must use an array to store the list, the other must use a linked list. The linked list must use the provided `Link` class for the nodes.
   - Test client that verifies the correctness of your `List` implementations. Name this class `ListTest.java`.
   - Experimental client to run experiments and gather data using your implementations. The requirements for experiments are given in the *Experiments* section, below. Name this class `ListExperiment.java`.
   - **You will hand in all 4 of these files.** You will also hand in a `README` file that meets the department standards (see documents in Brightspace).You do not have to hand in the 3 source files that I provided for you.
   - I am requiring standard naming of your client files so that my testing can be automated. If your clients require any command line arguments, you MUST include instructions in your `README` in the compilation and run section.

   As noted above, you will hand in your two client code files, but I will run your code with my own test client for functional testing. That means that your concrete classes must work correctly as described in the interface, without adding `public` methods beyond those defined in the interface. In other words, the API for the `List` must remain as given in the `List` interface.

3. (25 points) **Experiments, Analysis, and Report.** Refer to the documents in Brightspace in `Content->Course Info and Resources-Lab Report Information` for the report template, additional information, and guidance about writing your analysis.

   (a) *Experiments.* The objective of the experiments is to test the running times of `append`, `insert`, and `delete` operations on `Lists` with the two different underlying representations. The outline of the experiments is as follows.

   - You will collect timing data for the different conditions described below. *Notes on data collection:*
     - You must use nanoseconds instead of milliseconds for these experiments. Here is a quick example of how to use `nanoTime()`: How to measure elapsed time in nanoseconds with Java
     - To reduce the amount of data you collect, record the time every 500 insertions or deletions, instead of for every single operation. This will give you 20 data values for each condition rather than 10,000 data values.

  - You may record data more frequently if you wish (for example, every 250 operations), but *do not* reduce the collection frequency.
- For `insert` and `append` experiments, begin with an initial `List` size of 10,000 `Fizzbin` objects.
- Starting always with your original `List`, insert 10,000 new items to the `List` so that the insertions occur in different places: (a) at the head of the `List` (`insert(0)`), (b) at the tail (this is `append()`, and (c) at arbitrary positions chosen randomly. You need to track the timing for each condition separately.
- For `delete`, begin with an initial `List` size of 20000 `Fizzbin` objects. Delete 10,000 items from the initial `List`, with the same process as described above for insertions – delete the first element, delete the last element, delete an element at random.
- Since this is the first experiment assignment, you need run only 1 set of all experiments. You do not have to run replicate experiments.

(b) *Analysis and Report.* Following the guidance in the "Sample Lab Report" and using the LaTeX template provided in Brightspace, write your formal lab report. Be sure that you fill in all the sections of the report with information that is relevant to the current experiment. Do not leave my explanatory/placeholder text anywhere in the report.

1. *Problem.* The introduction will include the theoretical analysis that you uncovered in your background research
2. *Methods.* In the Methods section, you will include relevant excerpts of your source code. Do not copy/paste all your code into the report; include ONLY portions that you think are important to the problem at hand. You must include the experiment parameters.
3. *Results.* As part of reporting your results, include the following line plots (8 total), using `List` size as the x-axis and time as the y-axis:
   - Insertions:
     - Insertions at the head of the `List`.
     - Insertions at the tail of the `List` (`append`).
     - Insertions at random positions in the `List`.
     - Grand average of all insertions (both `insert` and `append`).
   - Deletions:
     - Deletions at the head of the `List`.
     - Deletions at the tail of the `List`.
     - Deletions at random positions in the `List`.
     - Grand average of all deletions.
   - Be sure you make **line plots**. This is the best type of plot to visualize the kind of data we are analyzing.
   - Because of the number of plots, you may want to look into the LaTeX `subfigure` package. This is not required, and don't worry if the plots get placed strangely in your document. Feel free to come see me for suggestions about how to make your plots behave.
4. *Discussion.* Here, you will explain what the results *mean.* This is the interpretation of the results. Dig into the details of what you observe in the results and what those observations imply. The discussion needs to flesh out what the plots show. Data plots report results, they do not interpret or discuss the results. Remember that the focus of the assignment is the time performance of `insert` and `delete` with the two underlying implementations of `List`. Pay particular attention to any details that jump out of the plots at you, such as big spikes in either direction. What may be causing those things? It's your job to explain what we are seeing in your data.
5. *Conclusion.* In the Conclusion, you will review the results and state explicitly whether your results supported your hypothesis or failed to support it. You must state precisely how the results support or refute your hypothesis. You must also suggest further experiments that will give additional evidence for the hypothesis or determine why the hypothesis was falsified. Please see the sample report for more information about what must be in the conclusion.

Assignment Submission.

Deliverables: What to turn in.

You will turn in the following files for your assignment:

1. Your source code files:
   (a) Java source files of the 2 concrete `List` implementations;
   (b) Test client for the `Lists`;
   (c) Experimental client (total of 4 source files).
2. `README` file.
3. The raw data files that were produced by the experiments. Depending on how you set up your data files, this could be a lot of files.
4. The tool that you used to produce your data plots. For most students, this artifact is the data spreadsheet that you created to make your plots, including both the raw data and the plots you generated. **Note:** If you used an online spreadsheet (such as Google Sheets) be sure that you download the finished spreadsheet as an Excel file, NOT as a PDF. I need to have an "executable" version of the spreadsheet, not an image of it. If you write scripts to make your plots (for example, in Python), you must turn in the scripts.
5. PDF of the analysis lab report, typeset in LaTeX. Be sure to edit the author and assignment information, as shown in the template. The date will be automatically generated by LaTeX. Turn in ONLY the PDF of the file, not the `.tex` source. Please be sure that your name appears in both the file name of the lab report and in the document itself (see example below).

Submit all required files to your class organization and repo in Gitea by the published deadline.

File naming conventions for homework

For files you turn in for any assignment that are NOT program source files or data files, the file name must include your name. For example, my lab report for this assignment would be named `p02ReportGrabowski.pdf` and the data spreadsheet would be named `p02DataAnalysisGrabowski.xlsx`.