CS3460: Data Structures

Midterm Exam

Total Points: 25

Problems

Notes on Grading: Unless otherwise stated, all programs will receive input via `System.in` and will output solutions via `System.out`.

To simplify the grading process, all grading will be automated. When applicable, you will be provided with sample input/output files for testing. You can ensure that your program will receive full marks by testing it with these provided files.

```
$ java YourProgram < input.txt > output.txt
$ diff output.txt correct.txt
```

The first line executes the Java program, redirecting input from a file `input.txt` and writing the output to a file `output.txt`. The second line compares your program's output (now stored in `output.txt`) with the correct answer (stored in `correct.txt`). If these files match exactly, the `diff` program will print nothing. Otherwise, it will list the differences.

Submission: Please submit the files `Jigsaw.java` and `Trique.java`.

Honor Code

The following document is an individual exam meant to be worked on and completed by an individual student without the aid of any other student or outside references. Discussing the contents of this exam with another student or exchanging material of any kind with another student constitutes plagiarism. Please review Appalachian State University's guidelines on this matter.

By submitting this exam, you implicitly agree to uphold the honor code. You may use any code posted on AsULearn, any notes you have taken, and any labs you have submitted as reference material, as well as references online for general Java programming help (syntax, semantics, and standard library documentation). You may not search the Internet for specific solutions to problems.

In preparation for this exam, I have found a wide variety of code across the internet that seeks to solve similar types of problems, and I will be using MOSS, a tool for measuring software similarity, to help identify cases of plagiarism.

1. **The Jigsaw Mystery (10 points):** Every day, for as long as he can remember, Prof Waters receives an anonymous parcel: a small red box. Every day, he opens the parcel and inside is a single jigsaw puzzle piece. It looks like if he can just collect all of the pieces, he should (a lack of jigsaw puzzle skills notwithstanding) be able to assemble the puzzle and discover the meaning behind these mysterious parcels.

   The problem is: the jigsaw puzzle pieces are random. They are delivered in no particular order, and most frustratingly, sometimes there are duplicate pieces. Sure, most jigsaw puzzle pieces look alike, but there are **definitely** duplicates in this case.

   You have been given access to Prof Waters' spreadsheet. No mystery is complete without tabular data entry. The first line of input contains two numbers, k and n, where k is the total number of pieces in the completed jigsaw puzzle, and n is the total number of jigsaw pieces received so far. The next n lines each contain a single integer representing an "id" for that jigsaw piece (two matching pieces will be assigned the same id).

   Your job is to analyze this file and tell me whether the jigsaw could be completed or not. If not, your output should indicate how many jigsaw pieces have still yet to be delivered. Two sample inputs and outputs are shown below, and have also been provided as files. Please format your output to match the formatting shown in my example output files **exactly**.

   Your program should be named `Jigsaw.java`. For full credit, your program should work for $k \leq 10^6$ and $n \leq 10^7$.

`puzzle-01-in.txt`

```
5 10
1
2
1
1
1
3
4
3
2
5
```

puzzle-01-out.txt

```
The puzzle is complete.
```

puzzle-02-in.txt

```
5 10
1
2
1
1
1
3
4
3
2
4
```

puzzle-02-out.txt

```
Missing 1 jigsaw piece(s).
```

2. **Tricky Trique (15 points):** A **deque** is a data structure that combines the concepts of a stack and a queue by allowing operations at either end of the data structure. In short, a deque is a *double-ended queue*, where you can push and pop values to and from both the front and the back of the data structure. This can be implemented efficiently in a number of different ways to support operations in $O(1)$ time.

   But sometimes, two ends just isn't enough. Introducing, the newest evolution in the queue data structure: the **trique** (proposed slogan: "three ends are better than two"[1]). The trique supports the following four operations:

   - `push_back x` - insert the element x into the back of the trique.
   - `push_front x` - insert the element x into the front of the trique.
   - `push_mid x` - insert the element x into the middle of the trique. The inserted element x will be the new middle of the trique. If n is the size of the trique before inserting, the location for x is $\frac{n+1}{2}$
   - `get i` - prints out the value at the i^{th} index

   Write a program `Trique.java` that implements this an `int` trique and includes a static `main()` method which acts as a driver. This driver should read operations from standard input, simulating these operations on the trique and printing out any output. Specifically, the `get i` command should print the value at index i to standard output. The program will end when it receives an `end` command.

   trique-01-in.txt

```
push_back 9
push_front 3
push_mid 5
get 0
get 1
get 2
push_mid 1
get 1
get 2
end
```

[1] slogan should be considered a work-in-progress and is subject to changes before it is ready to be published

trique-01-out.txt

```
3
5
9
5
1
```

Inputs may be quite large, so expect to handle extremely large datasets. For full credit, your implementation should be able to perform each of these operations as quickly as possible.