LAB NOTES

Week 6 - CS150
March 8th 2020
Evan Vu

1. Introduction

This week's lab primary goal is to implement Stack and Queue data structures. A Stack is a data structure in which elements are added on top of each other (via push method) and accessed from the top. It is a Last In First Out data structure, where you can only modify or inspect the most recent node or item. A Queue is a data structure in which elements are added at the end and accessed from the top. It is a First In First Out data structure, where you can only modify or inspect the least recent node or item.

2. Unit Test

JUnit Testing for the Node Class:

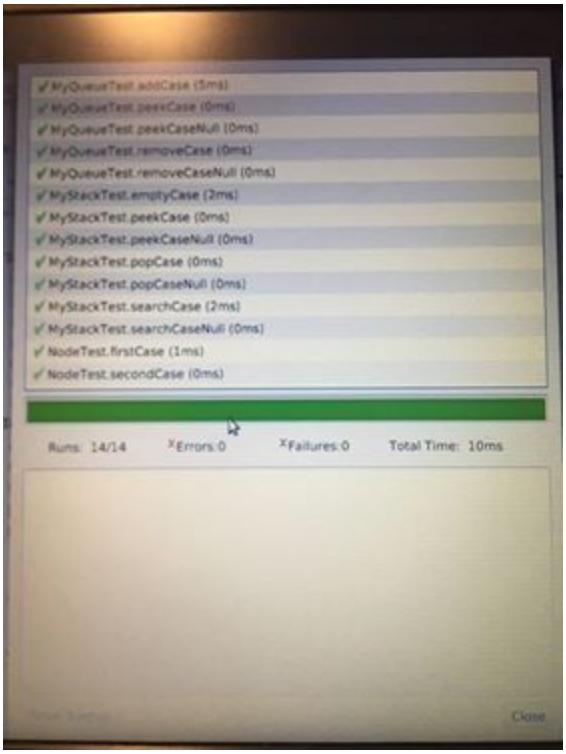
- firstCase: test the setValue() method and getNext() method
- secondCase: test the setnext() method and calling getNext() on a null node

JUnit Testing for Stack Class:

- emptyCase: call empty method on an empty and an occupied stack
- peekCase: test peeking after adding items to the stack
- peekCaseNull: test peeking not having added items to the stack
- popCase: test popping after adding items to the stack
- popCaseNull: test peeking not having added items to the stack
- searchCase: search a value in the stack
- searchCaseNull: search a value not in the stack

JUnit Testing for Queue Class:

- addCase: adding new items to the queue
- removeCase: test removing after adding items to the queue
- removeCaseNull: test removing not having added items to the queue
- peekCase: test peeking after adding items to the queue
- peekCaseNull: test peeking not having added items to the queue



3. Required Output

After testing different methods in ExperimentController with different input sizes, these are the runtimes recorded:

Methods MyStack.push() and MyQueue.add():

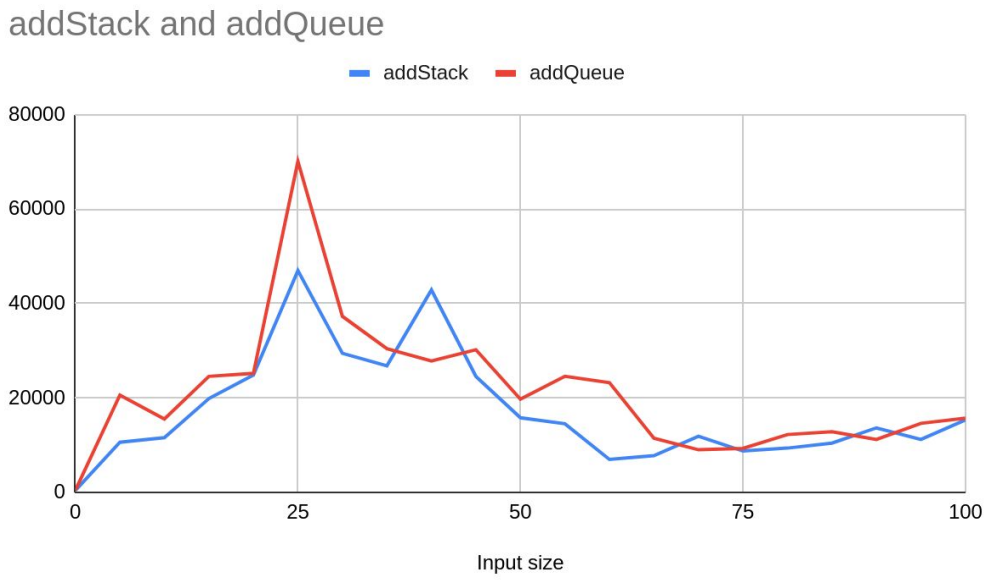


Figure 1: Runtime between MyStack.push() and MyQueue.add()

Methods MyStack.pop() and MyQueue.remove():

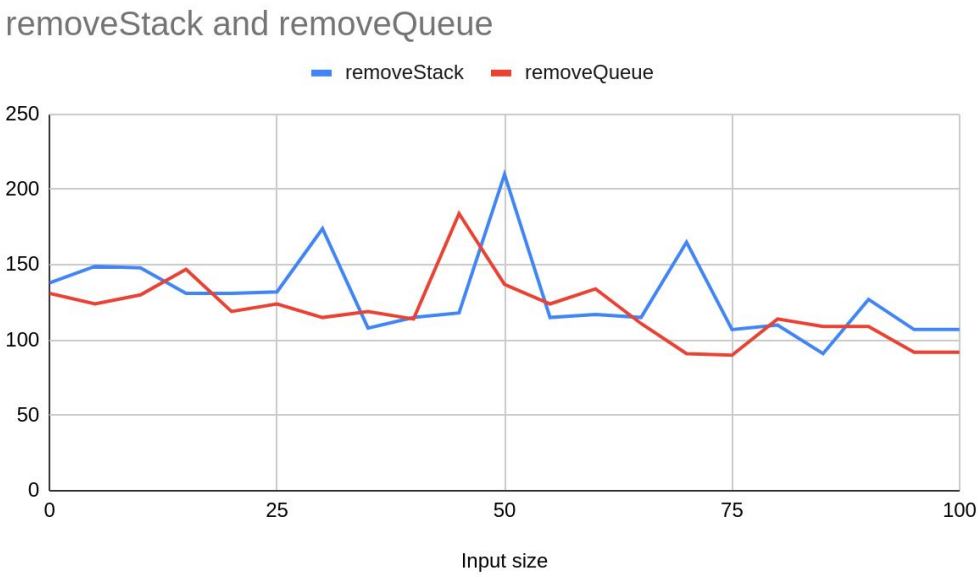


Figure 2: Runtime between MyStack.pop() and MyQueue.remove()

Runtime of search method in MyStack class:

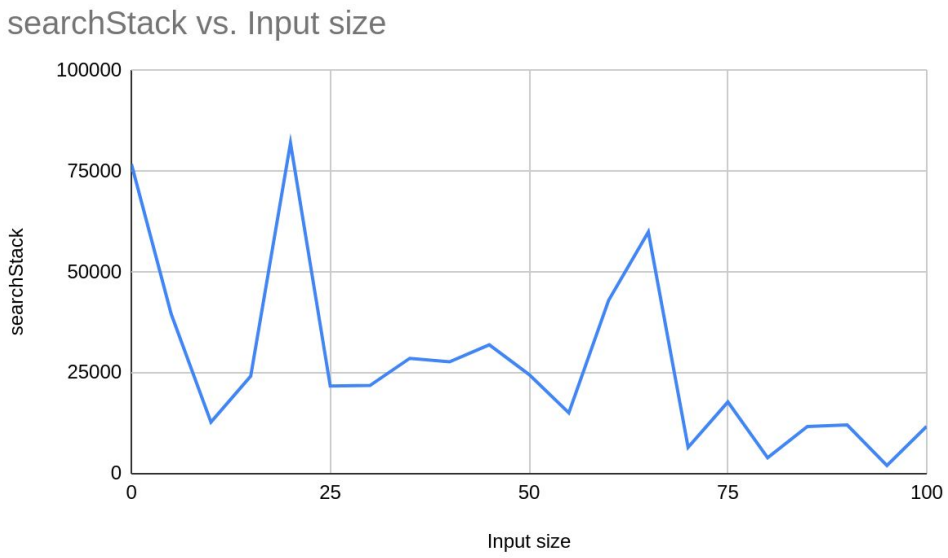


Figure 3: Runtime of MyStack.search()

Runtime of transferring a queue to stack and vice versa:

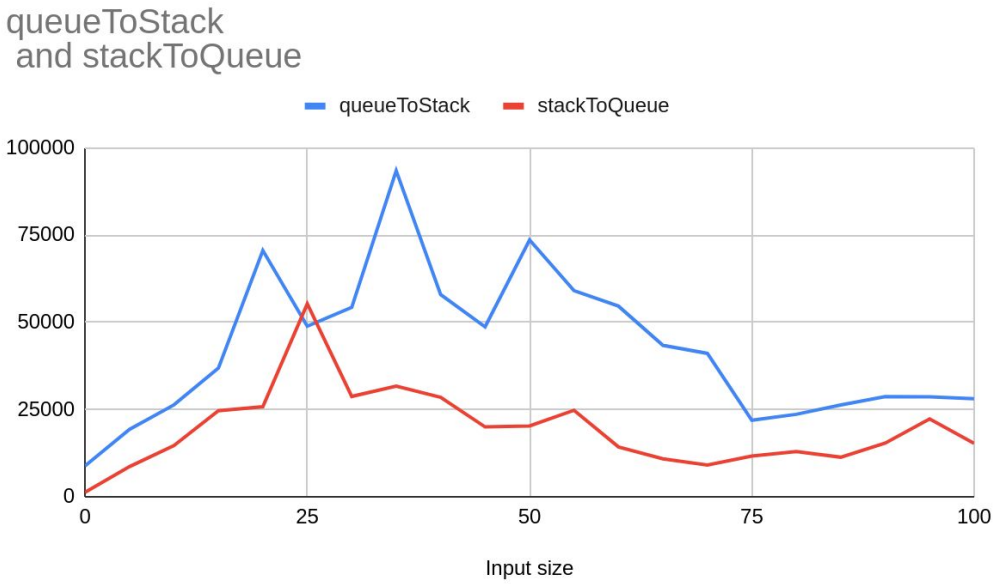


Figure 3: Runtime of transferring queue to stack and vice versa

Discussion:

According to Figure 1 and Figure 2, it can be seen that the add and remove methods for both stack and queue class are similar in running time. However, there is a consistent discrepancy in the running time of transferring a queue to a stack and stack to queue. According to Figure 3, running time for transferring a queue to a stack is longer than transferring from a stack to a queue.

As to the running time of the search method, even though the experimenter expected that run time will increase as input size increases, there is no significant relationship among those two. This might be because of the randomization of the searched element and elements added to the stack .

4. Trouble Report

A screenshot from the computer could not be obtained since the experimenter has not discovered how to take a screenshot on a chromebook that runs Ubuntu.

There is no JUnit test class for MyStackIterator since it is similar to Week 5's linked list iterator and has been tested then.

5. References

java.io (Java Platform SE 8). (n.d.). Retrieved February 16, 2020, from https://docs.oracle.com/javase/8/docs/api/java/io/package-summary.html

java.time (Java Platform SE 8). (n.d.). Retrieved February 16, 2020, from https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html

java.util (Java Platform SE 8). (n.d.). Retrieved February 16, 2020, from https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html

Random (Java Platform SE 8). (n.d.). Retrieved February 16, 2020, from https://docs.oracle.com/javase/8/docs/api/java/util/Random.html

Iterator (Java Platform SE 8). (n.d.). Retrieved February 16, 2020, from https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html