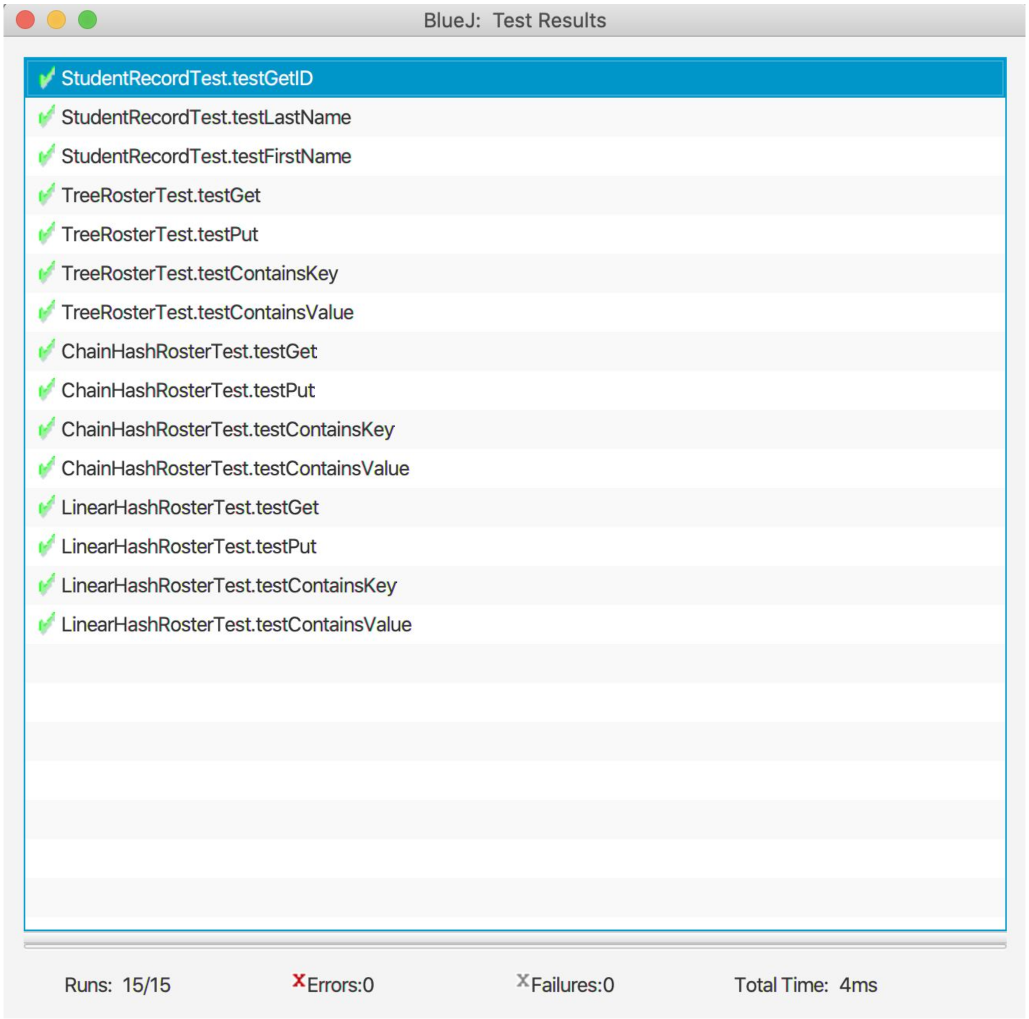
Dylan Maloy
CS150 Lab 12
Lab #12 write-up
11/24/19

Introduction:

The purpose of this lab was to better understand how different types of hashmaps are implemented. In the lab, we would implement both a **linearHashRoster** class and a **chainHashRoster** map which each had their implementations of the methods in the **Roster** interface. The chain implementation extended the **TreeRoster** class (chains were made out of treemaps). Both the linear and chain implementations had thresholds which would resize the array when exceeded. For the chain, it was the max chain length of any one chain, and for the linear implementation it was the % buckets filled.

Unit tests:

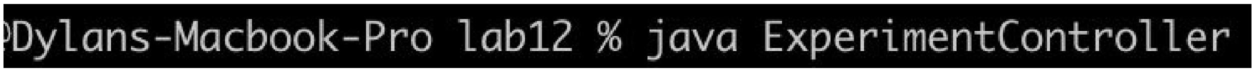


✓ StudentRecordTest.testGetID
✓ StudentRecordTest.testLastName
✓ StudentRecordTest.testFirstName
✓ TreeRosterTest.testGet
✓ TreeRosterTest.testPut
✓ TreeRosterTest.testContainsKey
✓ TreeRosterTest.testContainsValue
✓ ChainHashRosterTest.testGet
✓ ChainHashRosterTest.testPut
✓ ChainHashRosterTest.testContainsKey
✓ ChainHashRosterTest.testContainsValue
✓ LinearHashRosterTest.testGet
✓ LinearHashRosterTest.testPut
✓ LinearHashRosterTest.testContainsKey
✓ LinearHashRosterTest.testContainsValue

Runs: 15/15 Errors:0 Failures:0 Total Time: 4ms

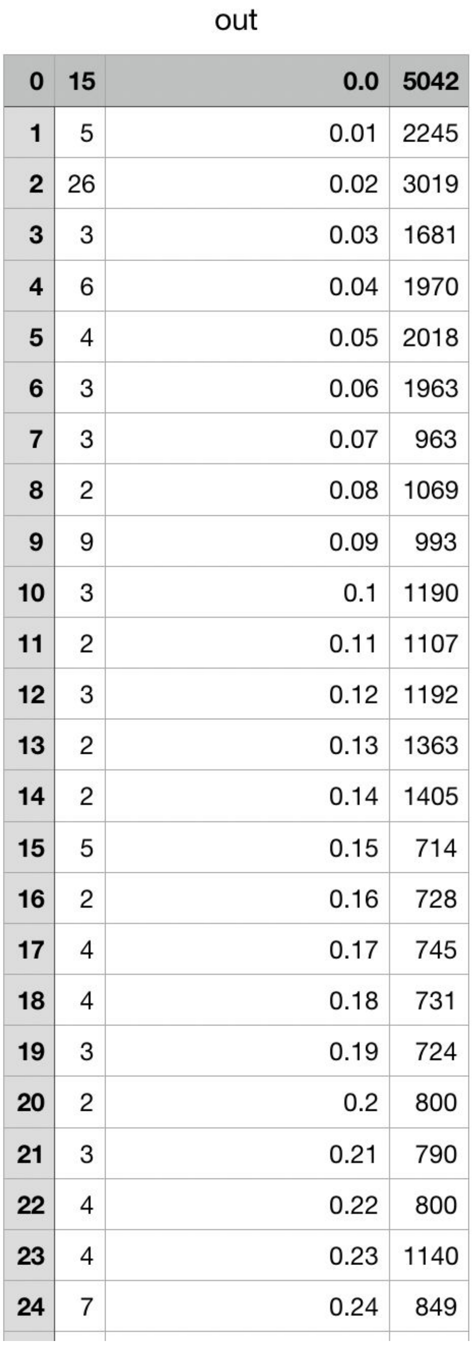
- BlueJ unit testing window

Required Input/Output:



```
Dylans-Macbook-Pro lab12 % java ExperimentController
```

- Input command



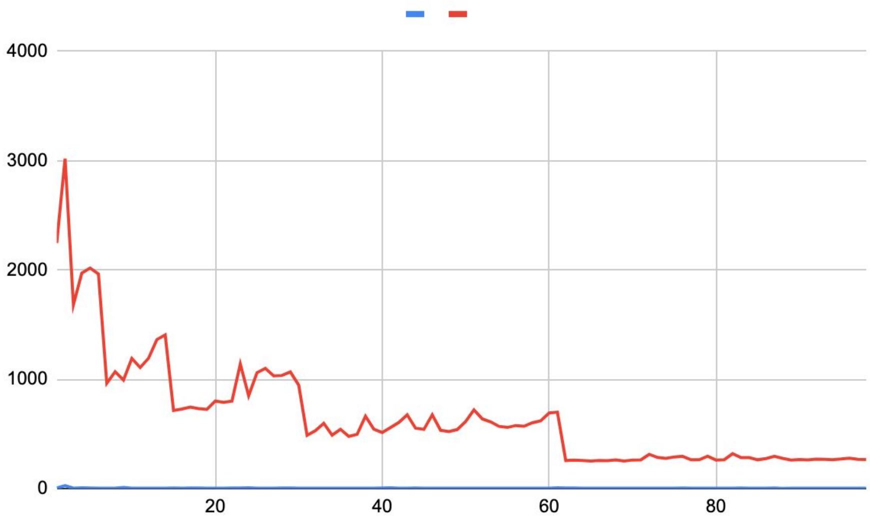
out

0	15	0.0	5042
1	5	0.01	2245
2	26	0.02	3019
3	3	0.03	1681
4	6	0.04	1970
5	4	0.05	2018
6	3	0.06	1963
7	3	0.07	963
8	2	0.08	1069
9	9	0.09	993
10	3	0.1	1190
11	2	0.11	1107
12	3	0.12	1192
13	2	0.13	1363
14	2	0.14	1405
15	5	0.15	714
16	2	0.16	728
17	4	0.17	745
18	4	0.18	731
19	3	0.19	724
20	2	0.2	800
21	3	0.21	790
22	4	0.22	800
23	4	0.23	1140
24	7	0.24	849

- Output out.csv file

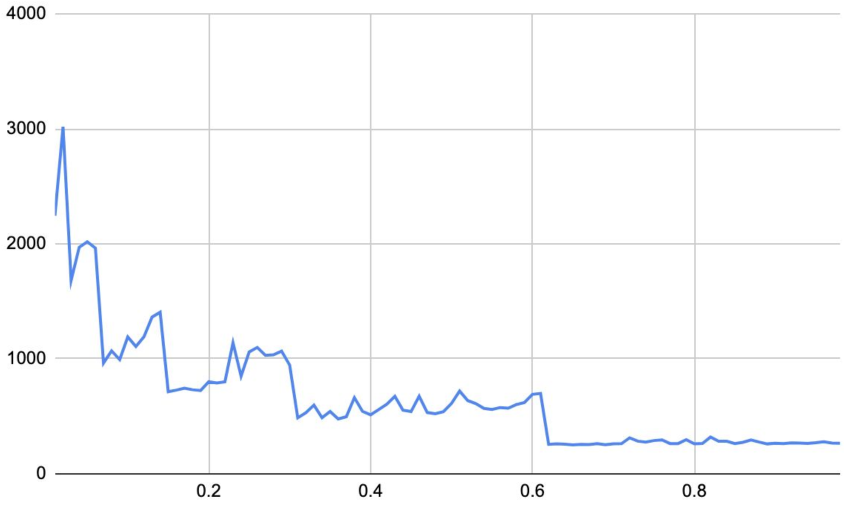
Data:

Y-axis: time in ms (add 10000 buckets)
X-axis: threshold (80 for chain = 0.8 for linear)
Blue = chain, Red = linear



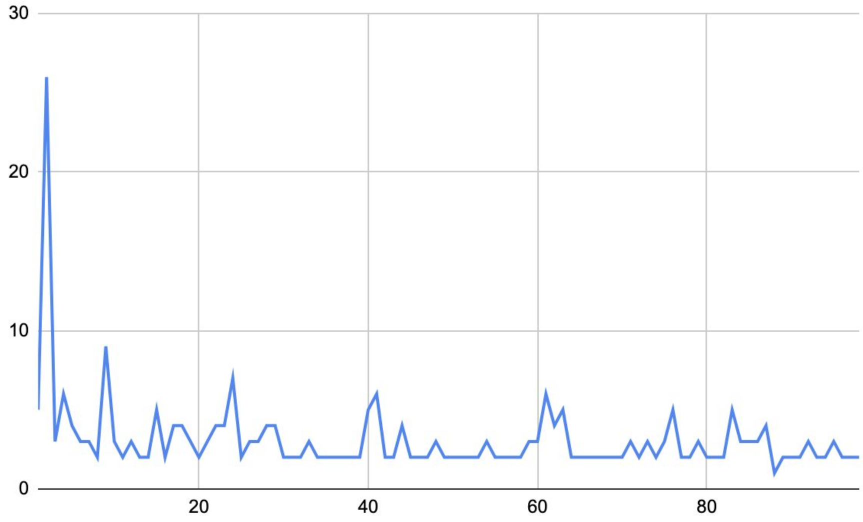
- Time complexity of Chain (blue) vs Linear (red)

Y-axis: time in ms (add 10000 buckets)
X-axis: threshold (float between 0 and 1)



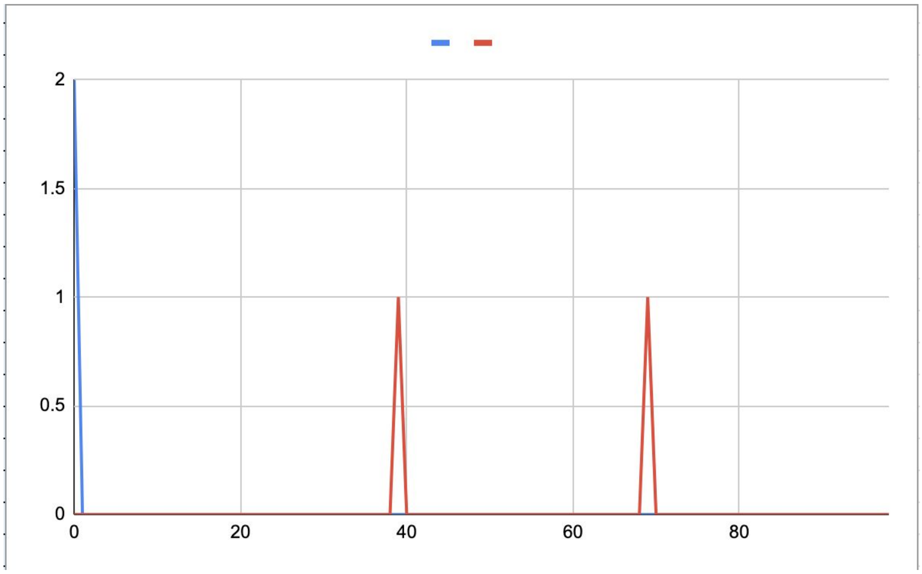
- Time complexity of linear implementation

Y-axis: time in ms (add 10000 buckets)
X-axis: threshold (max chain length)



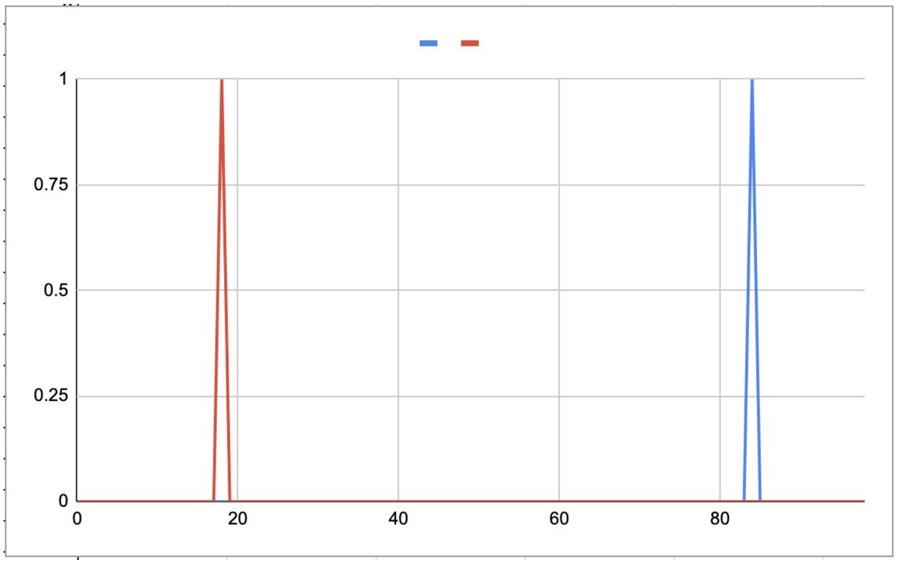
- Time complexity of chain implementation

Y-axis: time in ms (fetch bucket)
X-axis: threshold (80 for chain = 0.8 for linear)
Blue = chain, Red = linear



- Time complexity of linear and chain search (successful)

Y-axis: time in ms (fetch bucket)
X-axis: threshold (80 for chain = 0.8 for linear)
Blue = chain, Red = linear



- Time complexity of linear and chain search (unsuccessful)

Data Analysis:
The chain implementation turned out to be a lot more efficient than the linear implementation. The chain implementation was also a lot less volatile when looking at the graph comparing the two implementations. In terms of searches, they were pretty consistent across the board. Also - unsuccessful searches had a similar speed as successful ones.

Trouble Report:
This section is not applicable because all of my methods work as intended.

References:
Lysecky, R. (2019). Data Structures Essentials. ZyBooks.