**DISCLAIMER**

“Sort Lab” and “Simon Lab” are from previous class work and I was supplied with some files. “Bot” was a part of my Research project on the game Reversi and much of the code was obtained open source but was heavily modified by myself for the purpose of research.

To run the applications:

Sort Lab: Run the Lab6.java

Simon Lab: Run the Lab3.java

Reversi Bot: Run the TestWindow.java from the path reversi.ui.TestWindow

Here is a description of the code below:

**Sort Lab (using Hash)**

**Problem Summary:**

Hashing Lab is an extended study of the previous lab advanced storage lab. Three data structures must be used (hash, sort during, sort after) to sort and search through sets of people who have a name, email and SSN. A second list is used to search though the sorted list. The amount of compares is to be counted by the three data structures as well as the swaps for the data structures that are not the hash table. The search compares are to be counted as well.

**Problem Requirements:**

1. Sorting as you go. Use a standard sorted list (reference or array based) allowing up to 10,000 people.
2. Sorting after loading the data. For sorting, read the people into an array (allowing up to 10,000 people) and the sort the array using a O(n\*log(n)) sort.
3. Hashing. Create a hash table with room for 10,000 people (using the same type of people as the previous lab).
4. Collisions. When collisions occur, you are to determine how to handle them.
5. Data files that contain people.
6. Same interfaces as the previous labs must be used.

**System Design:**

**Testing Report**

My testing report matched my testing plan, this is because I did it by hand and was able to verify my results one I got my program working.

1. **Test Case 1: small scale test case ,14 unique people (random order)**

|  |  |
| --- | --- |
| **Input (List to Sort):**  test1 test1@gmail.com 310551509  test2 test2@gmail.com 175178753  test3 test3@gmail.com 989197001  test4 test4@gmail.com 813850777  test5 test5@gmail.com 894304318  test6 test6@gmail.com 985551757  test7 test7@gmail.com 1814079559  test8 test8@gmail.com 754851816  test9 test9@gmail.com 417707511  test10 test10@gmail.com 733732878  test11 test11@gmail.com 978507681  test12 test12@gmail.com 939508998  test13 test13@gmail.com 263918288  test14 test14@gmail.com 634869845 | **Input (List to Look up):**  test1 test1@gmail.com 310551509  test2 test2@gmail.com 175178753  test3 test3@gmail.com 989197001  test4 test4@gmail.com 813850777  test5 test5@gmail.com 894304318  test6 test6@gmail.com 985551757  test7 test7@gmail.com 1814079559  test8 test8@gmail.com 754851816  test9 test9@gmail.com 417707511  test10 test10@gmail.com 733732878  test11 test11@gmail.com 978507681  test12 test12@gmail.com 939508998  test13 test13@gmail.com 263918288  test14 test14@gmail.com 634869845 |

**Output:**

The hash table used 14 compare(s) to sort the data.

The merge used 38 compare(s) and 18 swap(s) to sort the data.

The array sorted list took 54 compare(s) and 48 swap(s) to sort the data.

14 is the amount of people given in the data.

105 is the amount of compares it took to look up people from one text file to the other.

**Why:** The hash only has 14 compares because my hash will use the last four digits to find an address and since these all have a unique set of last 4 digits the only compares are checking if the spot is null or not to see if it is open. Everything else I did by hand to see how many it would take. The last thing I output is 105 compares checking one list to the other and this is because the 2nd list will only search though the first list after the first list has already been sorted. These results are what I expected to see originally and I can now visually see the amount of swaps and compares it takes per data structure.

1. **Test Case 2: small scale test case ,7 unique people (in order)**

|  |  |
| --- | --- |
| **Input (List to Sort):**  Matt Matt@gmail.com 111111111  Matt Matt@gmail.com 111111112  Dave Dave@gmail.com 111111113  Tim Tim@gmail.com 111111114  Grant Grant@gmail.com 111111115  test test@gmail.com 111111116  test test@gmail.com 111111117 | **Input (List to Look up):**  Matt Matt@gmail.com 111111111  Matt Matt@gmail.com 111111112  Dave Dave@gmail.com 111111113  Tim Tim@gmail.com 111111114  Grant Grant@gmail.com 111111115  test test@gmail.com 111111116  test test@gmail.com 111111117 |

**Output:**

The hash table used 7 compare(s) to sort the data.

The merge used 11 compare(s) and 0 swap(s) to sort the data.

The array sorted list took 21 compare(s) and 0 swap(s) to sort the data.

7 is the amount of people given in the data.

28 is the amount of compares it took to look up people from one text file to the other.

**Why:** The hash only has 7 compares because my hash will only be checking if the spot is null or not and will not have any collisions. Both merge and array sorted have 0 swaps. It takes 28 compares to search the two lists because I check the two and increase the location by one if it’s not a match so in this case the searches would look like (1+2+3+4+5+6+7) to find all 7 inputs.

1. **Test Case 3: small scale test case ,7 unique people (1 swap)**

|  |  |
| --- | --- |
| **Input (List to Sort):**  Matt Matt@gmail.com 111111112  Matt Matt@gmail.com 111111111  Dave Dave@gmail.com 111111113  Tim Tim@gmail.com 111111114  Grant Grant@gmail.com 111111115  test test@gmail.com 111111116  test test@gmail.com 111111117 | **Input (List to Look up):**  Matt Matt@gmail.com 111111112  Matt Matt@gmail.com 111111111  Dave Dave@gmail.com 111111113  Tim Tim@gmail.com 111111114  Grant Grant@gmail.com 111111115  test test@gmail.com 111111116  test test@gmail.com 111111117 |

**Output:**

The hash table used 7 compare(s) to sort the data.

The merge used 11 compare(s) and 1 swap(s) to sort the data.

The array sorted list took 21 compare(s) and 1 swap(s) to sort the data.

7 is the amount of people given in the data.

28 is the amount of compares it took to look up people from one text file to the other.

**Why:** There is 1 swap for both the merge and the array. The searching compares are the same because it searches the sorted version of the list.

1. **Test Case 4: small scale test case ,7 not all unique people (hashing collision)**

|  |  |
| --- | --- |
| **Input (List to Sort):**  Matt Matt@gmail.com 111111111  Matt Matt@gmail.com 111111111  Dave Dave@gmail.com 111111111  Tim Tim@gmail.com 111111199  Grant Grant@gmail.com 111111111  test test@gmail.com 111111113  test test@gmail.com 111111112 | **Input (List to Look up):**  Matt Matt@gmail.com 111111111  Matt Matt@gmail.com 111111111  Dave Dave@gmail.com 111111111  Tim Tim@gmail.com 111111199  Grant Grant@gmail.com 111111111  test test@gmail.com 111111113  test test@gmail.com 111111112 |

**Output:**

The hash table used 19 compare(s) to sort the data.

The merge used 14 compare(s) and 6 swap(s) to sort the data.

The array sorted list took 16 compare(s) and 10 swap(s) to sort the data.

7 is the amount of people given in the data.

22 is the amount of compares it took to look up people from one text file to the other.

**Why:** The hashing deals with collisions when it cannot find a unique address by adding a plus one. This had some collisions because they have the same SSN and some collisions because their spot has been taken such as the last two people on my list.

1. **Test Case 5: large scale test case**

|  |  |
| --- | --- |
| **Input (List to Sort):**  10,000 random people | **Input (List to Look up):**  the same 10,000 people |

**Output:**

The hash table used 15160 compare(s) to sort the data.

The merge used 120452 compare(s) and 59092 swap(s) to sort the data.

The array sorted list took 25062704 compare(s) and 24942289 swap(s) to sort the data.

10000 is the amount of people given in the data.

50005000 is the amount of compares it took to look up people from one text file to the other.

**Testing Instructions**

To run the program, you will need all of the java files in one directory. (Lab6, Advanced, People...) To run a different txt file, you need to go inside of advanced and change the “people2.txt” to the appropriate name of the file you want to sort or search in either sortList() or searchList(). Next you will need to change the value of LISTSIZE to the amount of people on the .txt document. If you do not change the LISTSIZE to the amount of people you have you will get an error.

**Management Report**

This lab took longer than any lab this far, so I was happy that we had the extra week to do it. I also used my first due date extension for this lab. I estimated that this lab would take me 11-12 hours but in all it took nearly 20 hours give or take. That is by far the longest I’ve spent on a lab, but I needed to make sure I got a good grade. I put in a lot of effort and learned a lot.

**Lessons Learned**

It took me awhile, but I know have a deep understanding for both merge sorting and hash sorting. This lab tied a lot of things together for me that I was not sure on the past few labs. I also now realize I should’ve done more sooner not because I ran out of time on this lab but because I don’t have much time to study for my final exam now.

**Future Improvements**

In the future I must start the lab earlier this is nearly always true. I did a decent job overall with my time management though and I did not run out of time however I think that I could’ve finished sooner so I could now be studying for the final exam instead. My spacing in this lab is much better but still not perfect.

**Appendices**

On this lab I used classes modified from the book, this is documented in the headers of the class.

**Simon Game**

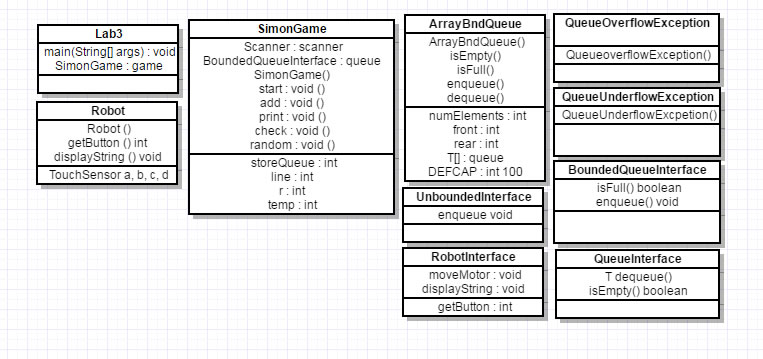
**Problem Summary:**

Simon is a game that asks the user to hit back the buttons that the game shows in order starting at one and incrementing by one each time.

**Problem Requirements:**

* Uses queues to track Simon game
* NXT Robot
* Command line input/output

**System Design:**



**Testing Report**

These test cases were preformed both inside of JAGRASP and command line.

1. Test Case 1: 2 3 2 1 2

input: 2 3 2 1 2

expected outcome: Correct!

outcome: Correct!

1. Test Case 1: 2 3 2 1 2 1

input: 2 3 2 1 2 3

expected outcome: Wrong!

outcome: Wrong!

1. Test Case 1: 1

input: e

expected outcome: Wrong!

outcome: Exception in thread "main" java.util.InputMismatchException

1. Test Case 1: 1

input: 2

expected outcome: Wrong!

outcome: Wrong!

1. Test Case 1: 4 2 3 4 2 1 4 3 3

input: 4 2 3 4 2 1 4 3 3

expected outcome: Correct!

outcome: Correct!

1. Test Case 1: 2 2 2

input: 2 2 2

expected outcome: Correct!

outcome: Correct!

1. Test Case 1: 4

input: 9

expected outcome: Correct!

outcome: Wrong!

**Testing Instructions**

To run the program, you will need all of the class files in one directory. (Lab3, SimonGame, ArrayBndQueue, BoundedQueueInterface, QueueInterface, QueueOverflowExcpetion, QueueUnderflowException, Robot, RobotInterface, SimonGame, UnboundedQueueInterface) The user can then run by running the Java file “Lab3” in command line or can run the program inside of JGRASP.

**Management Report**

Working on this lab I spent about 2 hours during lab time Wednesday and 1 hour shortly after lab in my dorm. This was mostly spent both wrapping my head around the program and working on an outline for my UML. After getting my UML approved from a TA and showing Dr. Bareiss I started to work on the SDR Saturday evening for another 2 hours. Next I spent 6 hours Tuesday working and finally finishing everything. This is a total of 10-11 hours of work for Simon Lab compared to the estimated time of 4-5 hours. Tuesday took much longer than I thought to finish my Simon game code and I ended up running out of time for the robot.

**Lessons Learned**

I now do I have a deep understanding for queues and how they work, I also realize I need to put in more time even if I think that problem or lab will be easy. I ended up running out of time on something that I thought would only take a couple hours. I should’ve started earlier so I could’ve finished my robot classes.

**Future Improvements**

I will start to get a head start on future labs. To helped me finish on time and finish with a much better quality of work.

**Appendices**

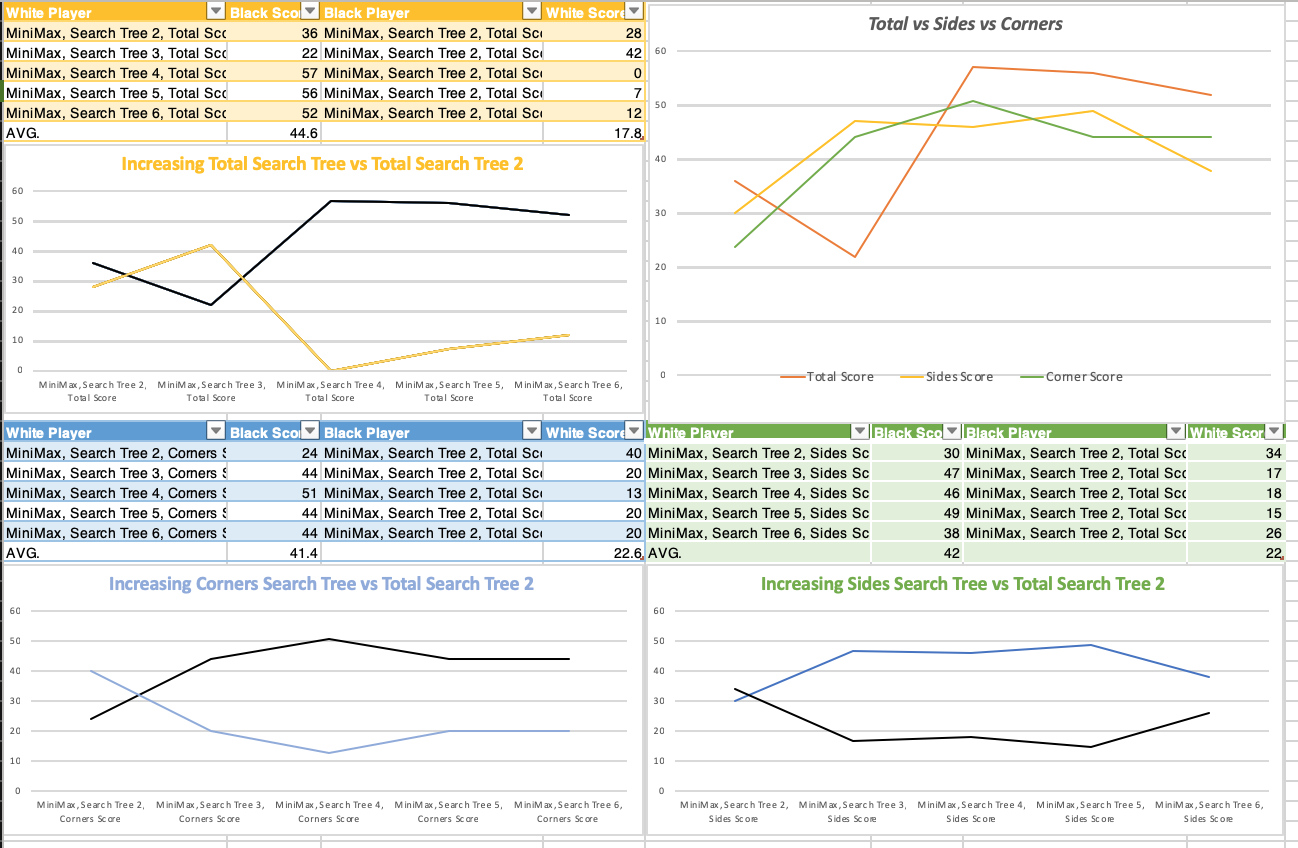
On this lab I used Dr.Bariess classes as well as classes modified from the book for programming class such as ArrayBndQueue, BoundedQueueInterface, QueueInterface, QueueOverflowExcpetion, QueueUnderflowException, Robot, RobotInterface, SimonGame, UnboundedQueueInterface.

**Reversi Project**

**Problem Summary:**

Reversi is a complex game in which I will be looking for the best strategies to play using AI.

**Testing Report**



**Testing Instructions**

To run the program, you will need all of the class files in one directory. The user can then run by running the Java file “TestWindow” from the path reversi.ui inside of Eclipse.