**INFO 6205 Spring 2022 Project  
  
The Menace**

**Team Members**

Name: Jayanth Vakkalagadda  
(NUID): 002950342

Name: Sai Meghana Surapaneni

(NUID): 002929424

**Introduction**

***Aim***

* Implement “The Menace” by replacing matchboxes with values in a hash table (key will be the state of the game)
* Train the Menace by running games played against “human” strategy,   
  which is based upon optimal strategy
* You will need to choose values for:

|  |  |
| --- | --- |
| **Parameter** | **Definition** |
| alpha | the number of ”beads” to in each “matchbox” at the start of the game, may be different for each move: first move, second move, etc.) |
| beta | the number of ”beads” to add to the “matchbox” in the event of a win |
| gamma | the number of ”beads” to take to the “matchbox” in the event of a loss |
| delta | the number of ”beads” to add to the “matchbox” in the event of a draw |

**Human strategy**

* Chooses optimal strategy with probability p\*. In the “zone,” chooses random move

**Implement logging**

* Log each training run with date/time, win/loss/draw, and p. If you vary alpha,   
  beta, etc. then record these values also (A suitable value of “p” might be 0.9)
* For your final match(es), log every move taken by the “Menace” and its  
  opponent
* Choose SLF4j as your logging framework (if you’re used to something else, or   
  not using Java, then choose whatever logging framework you like)

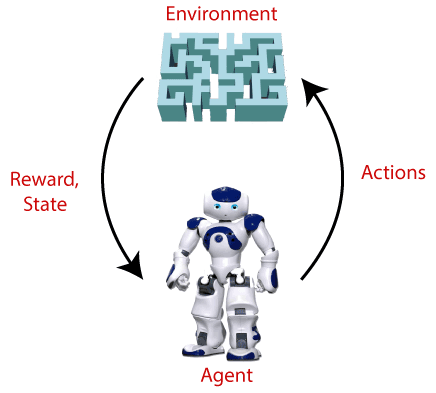
**Unit tests**

* You must run your unit tests before you start training. Show the date/time of   
  your most recent unit test run.
* Tests must have good coverage (each method must be covered).

***Approach***

**A picture containing toiletry, cosmetic

Description automatically generatedA picture containing toiletry, cosmetic

Description automatically generated****A picture containing text, person

Description automatically generatedA picture containing text, indoor

Description automatically generated**

Terms used in Reinforcement Learning

* **Agent():** An entity that can perceive/explore the environment and act upon it.
* **Environment():** A situation in which an agent is present or surrounded by. In RL, we assume the stochastic environment, which means it is random in nature.
* **Action():** Actions are the moves taken by an agent within the environment.
* **State():** State is a situation returned by the environment after each action taken by the agent.
* **Reward():** A feedback returned to the agent from the environment to evaluate the action of the agent.
* **Policy():** Policy is a strategy applied by the agent for the next action based on the current state.
* **Value():** It is expected long-term retuned with the discount factor and opposite to the short-term reward.
* **Q-value():** It is mostly similar to the value, but it takes one additional parameter as a current action (a).

**Program**

***Data Structures***

***Classes***

***Driver:*** This class has the main method for the project. All the User Inputs like parameter changes, number of iterations of the training/final matches are read from the main method.

***Game: It has the train and watch***

***Human***

***Operations***

***OptimalStrategy***

***Permutations***

***Rewards***

***BeadsFile***

***CSVreadWrite***

***Algorithm***

***Invariants***

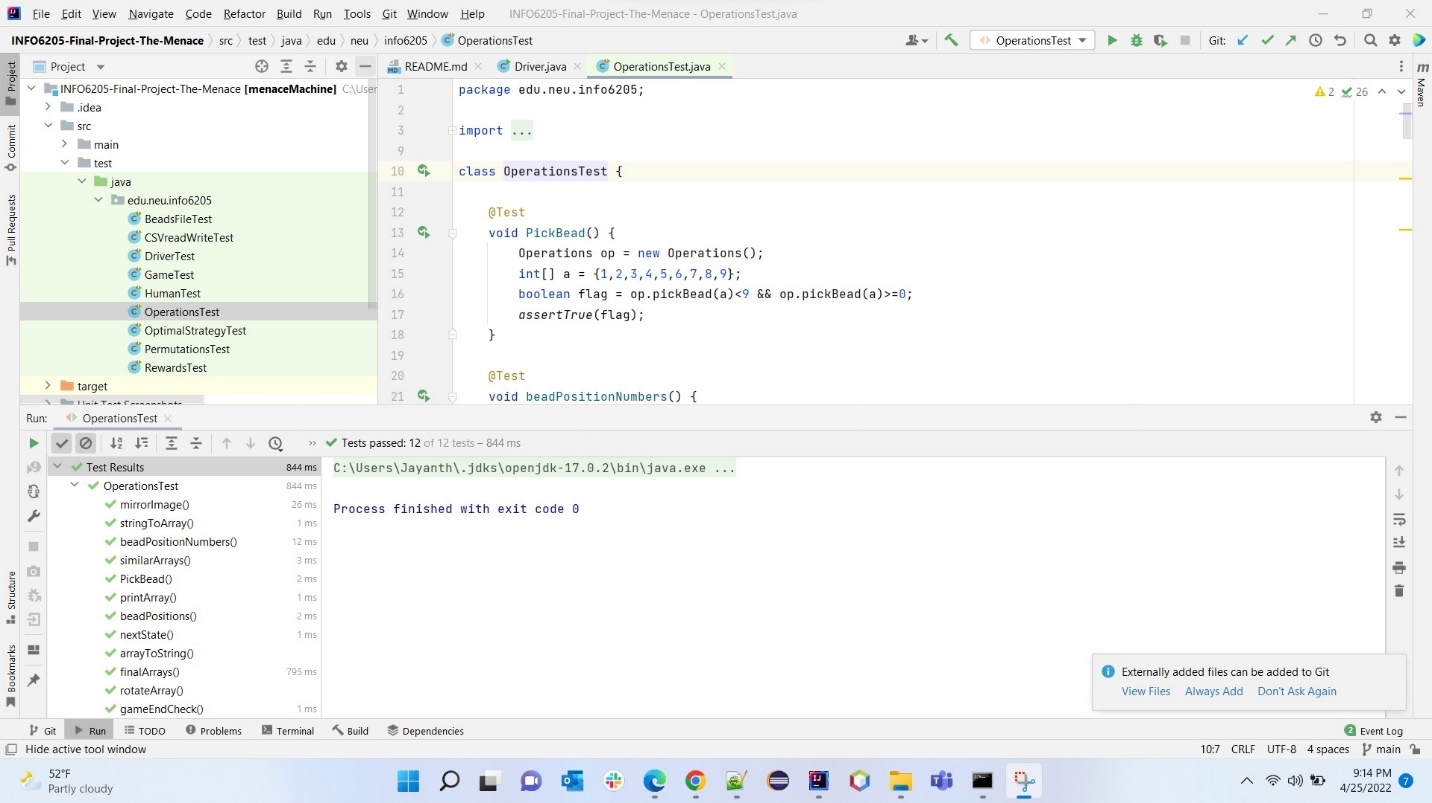
**Flow Charts**

**Observations & Graphical Analysis**

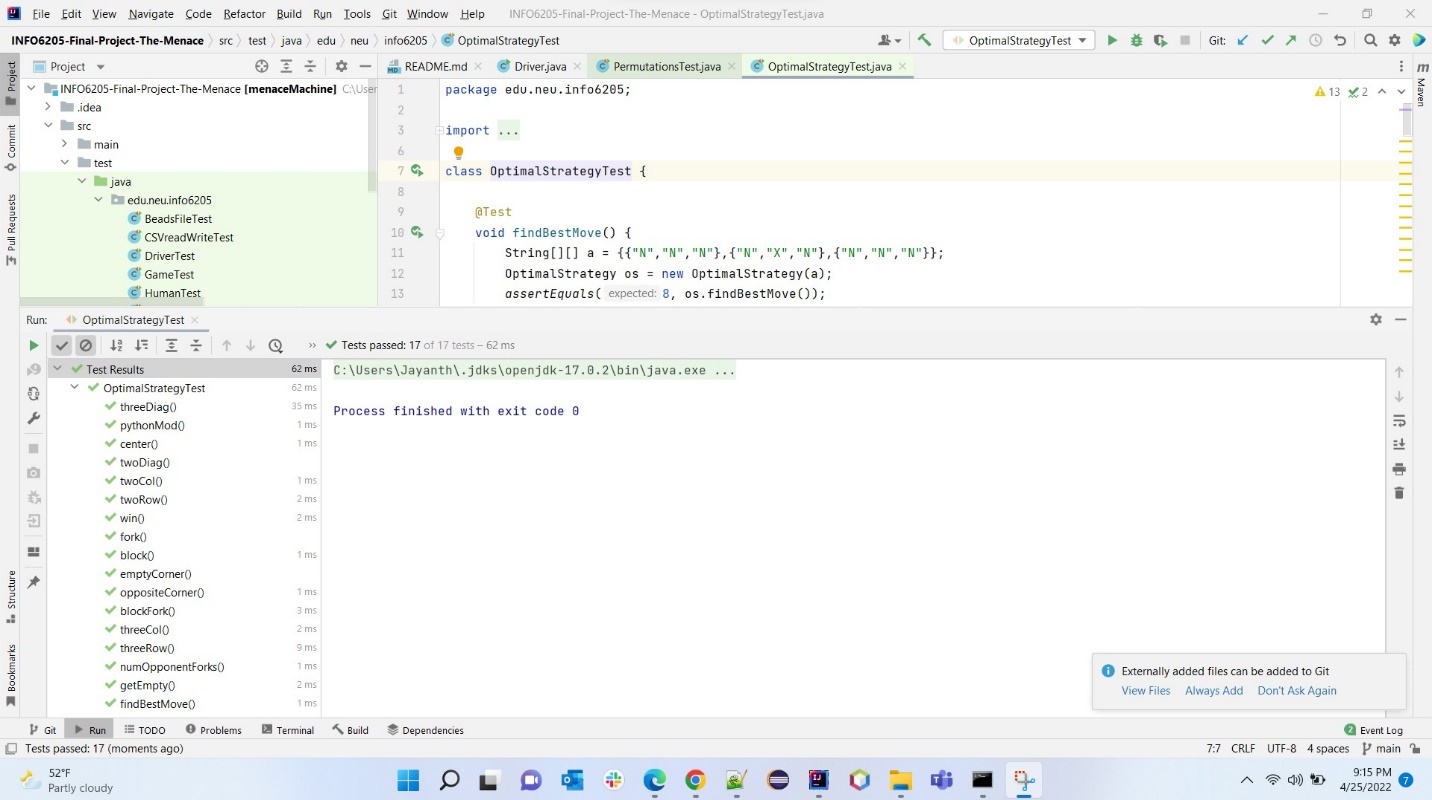
**Results & Mathematical Analysis**

**Unit tests**

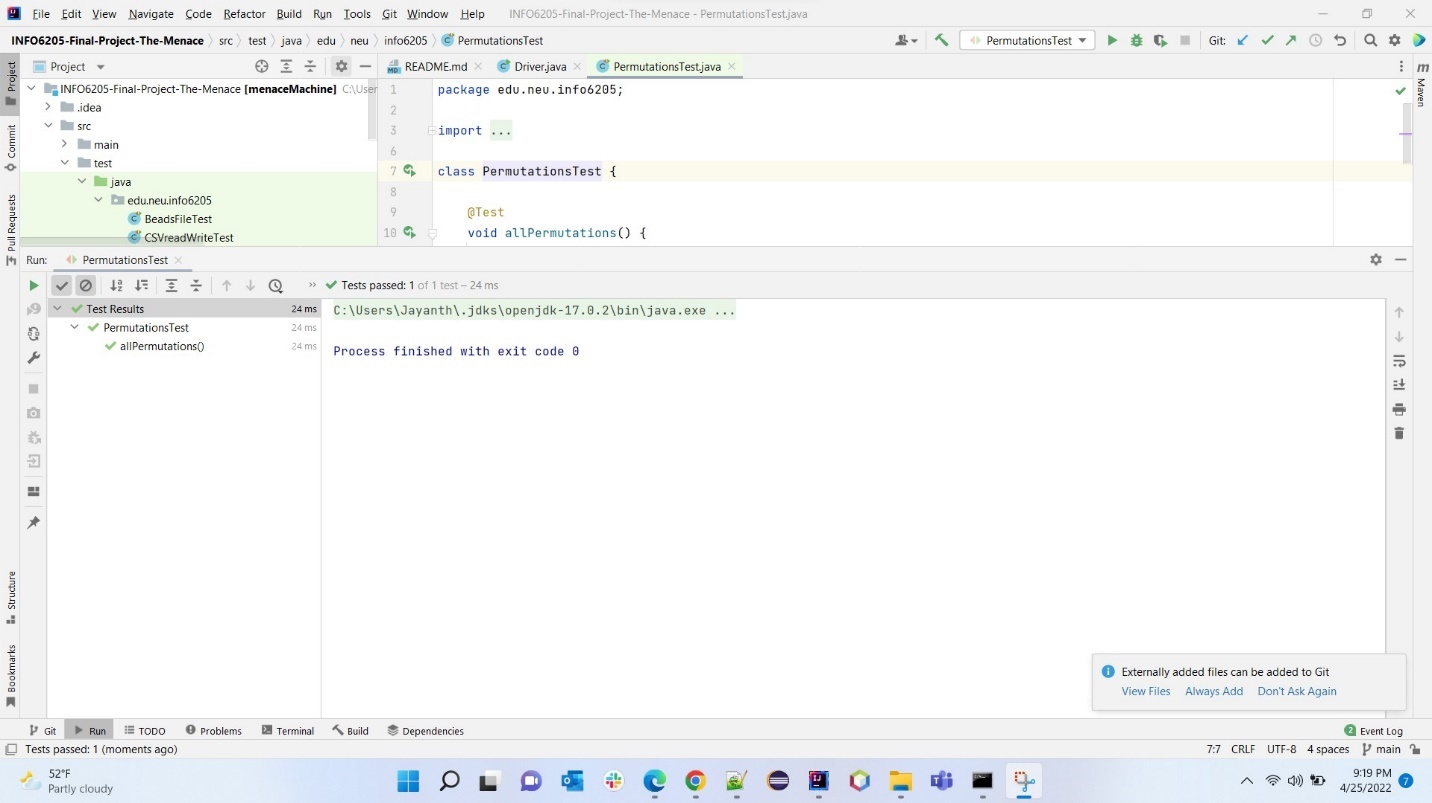
***OperationsTest.java***

****

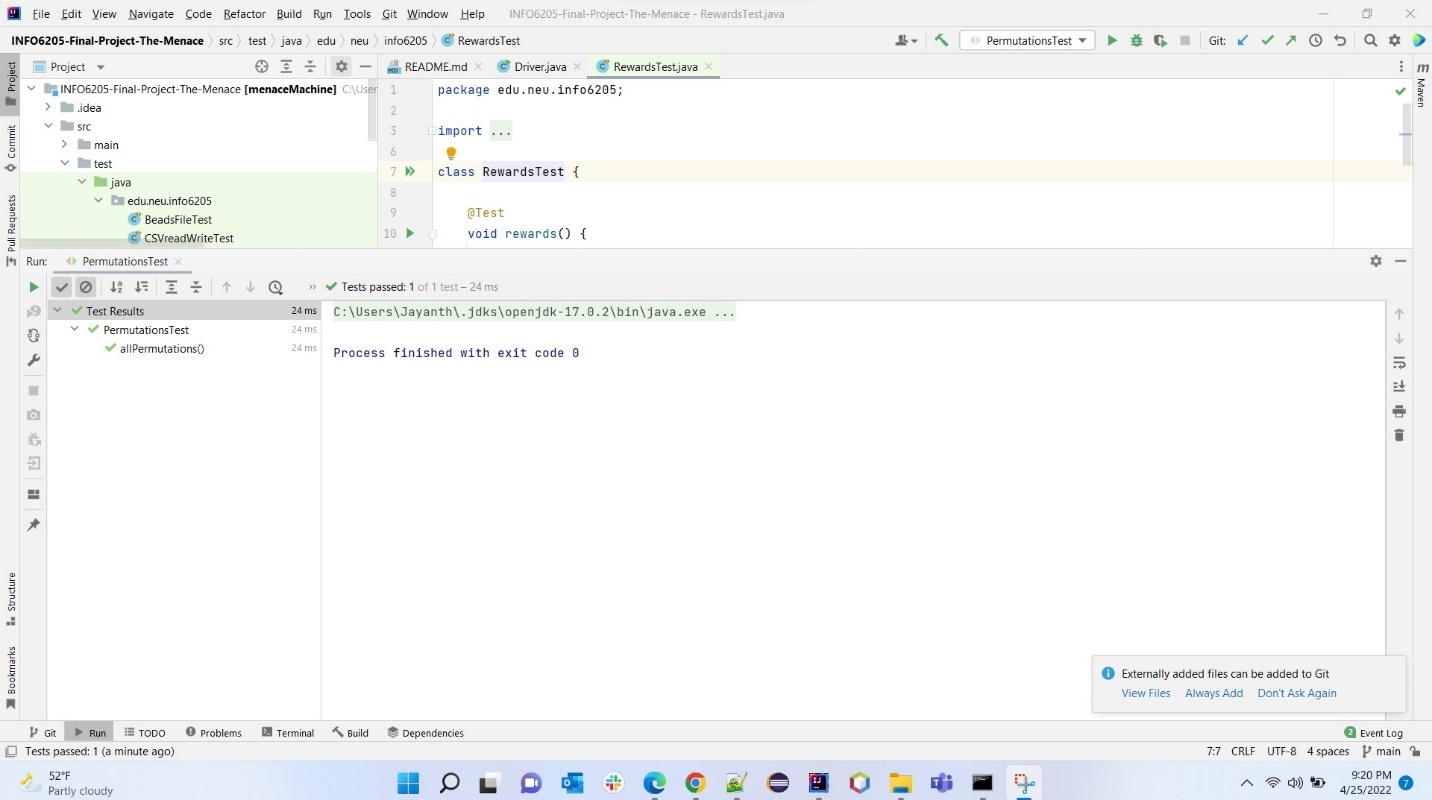
***OptimalStrategyTest.java***

****

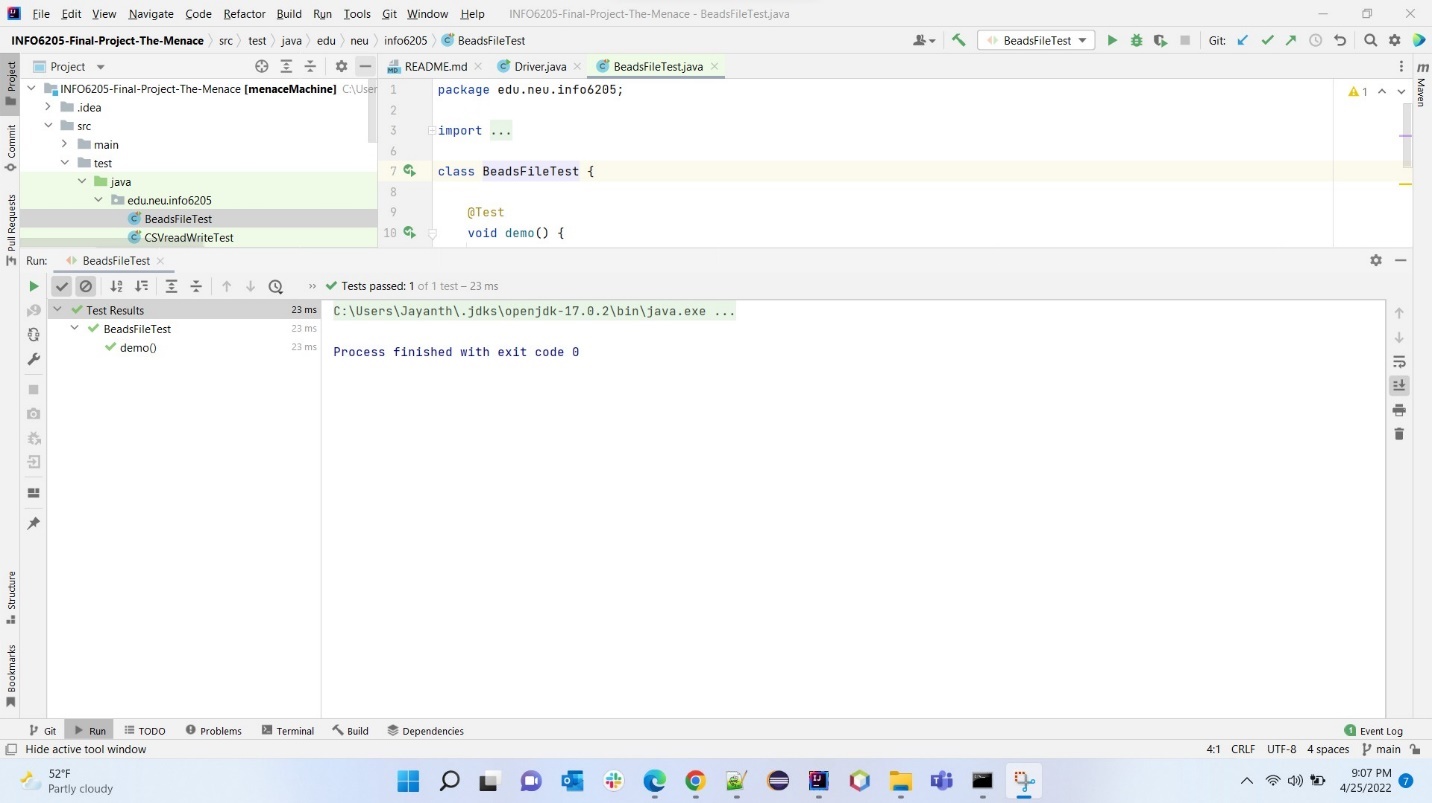
***PermutationsTest.java***

****

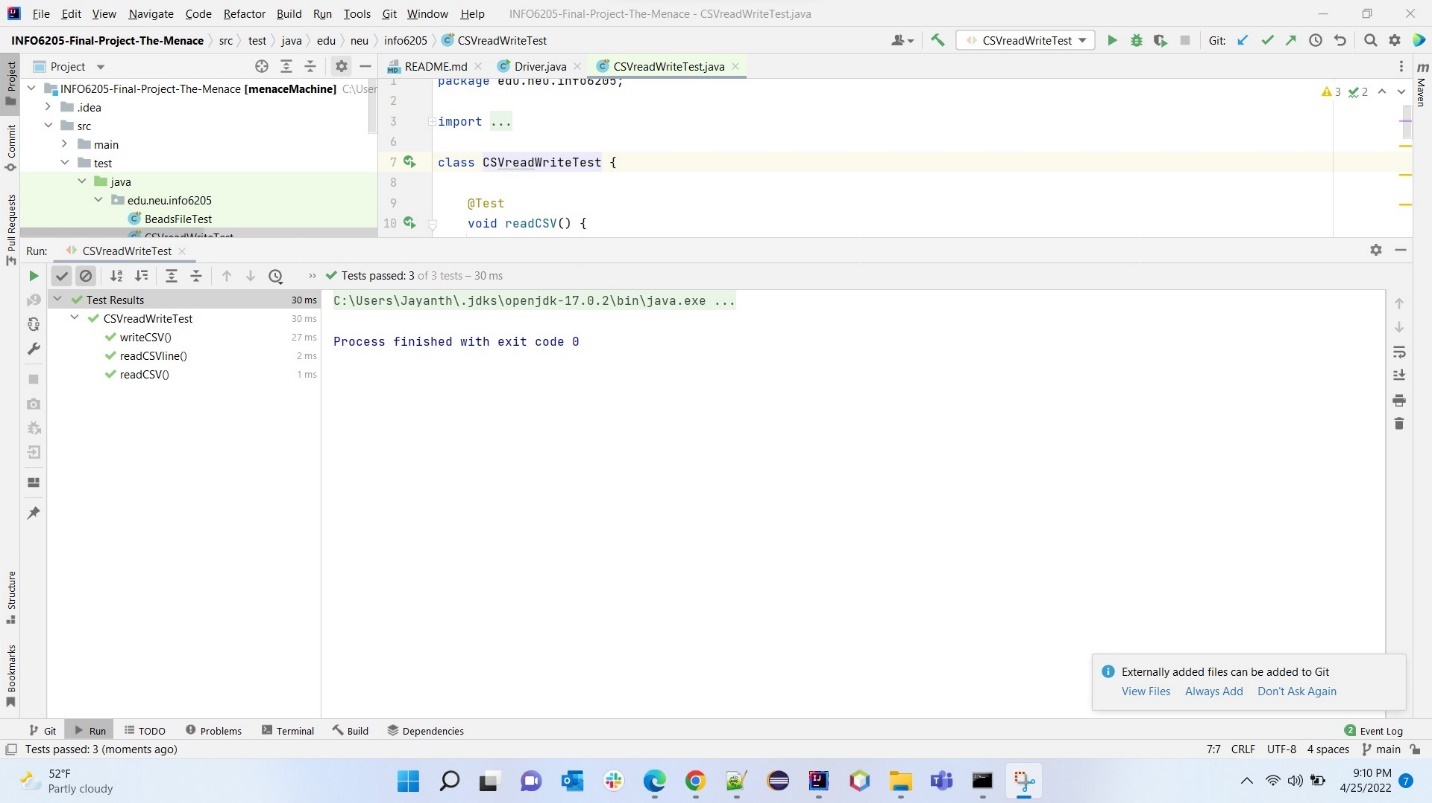
***RewardsTest.java***

****

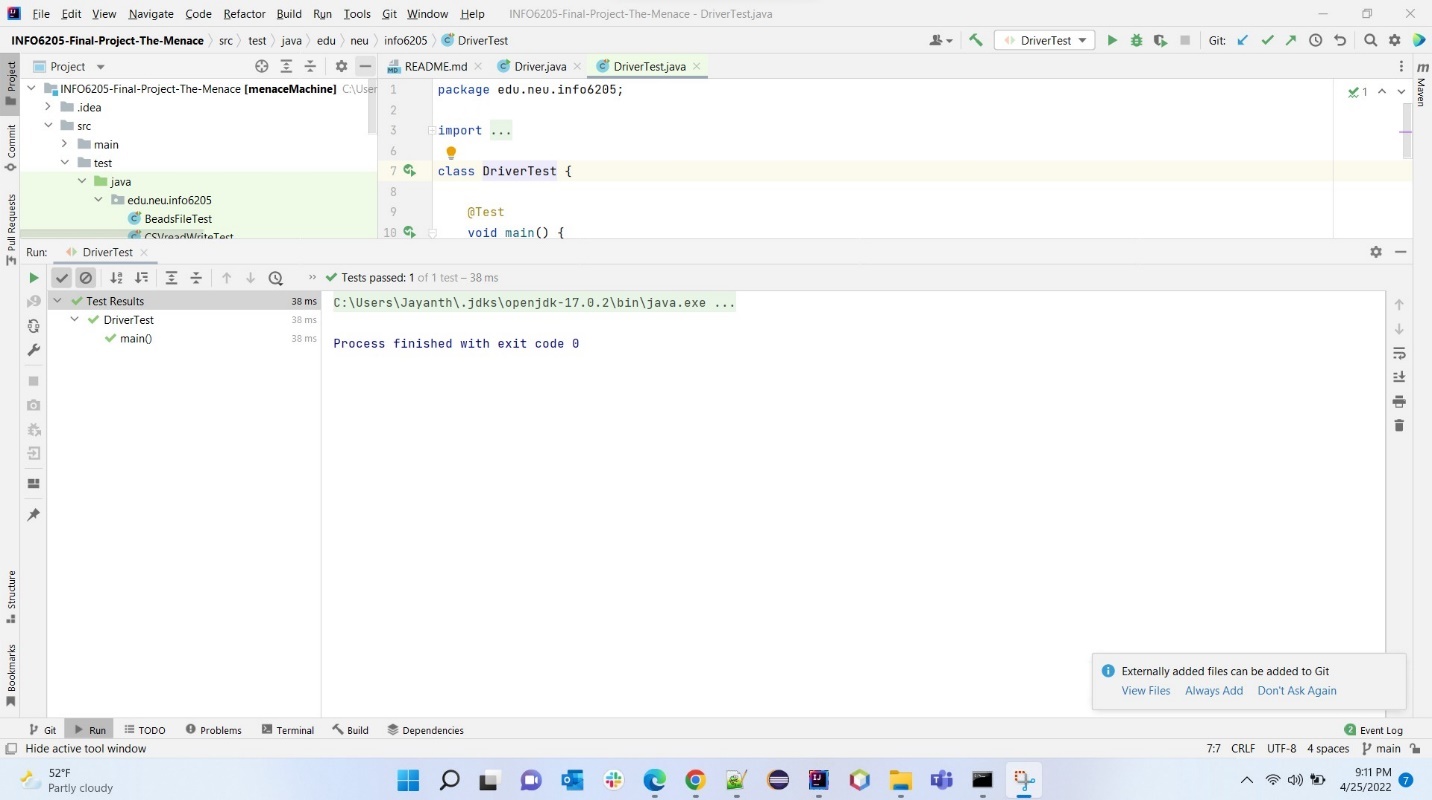
***BeadsFileTest.java***

****

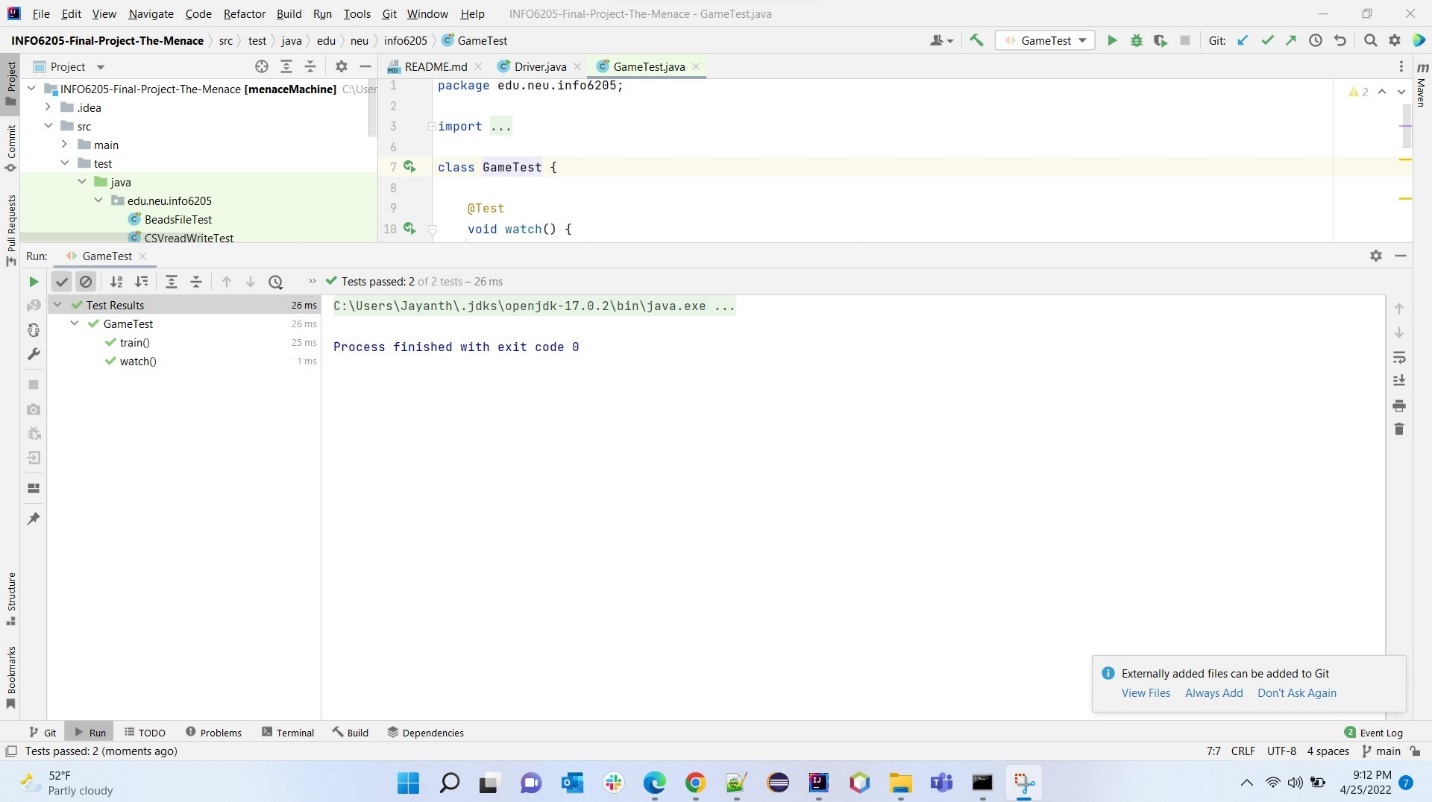
***CSVreadWriteTest.java***

****

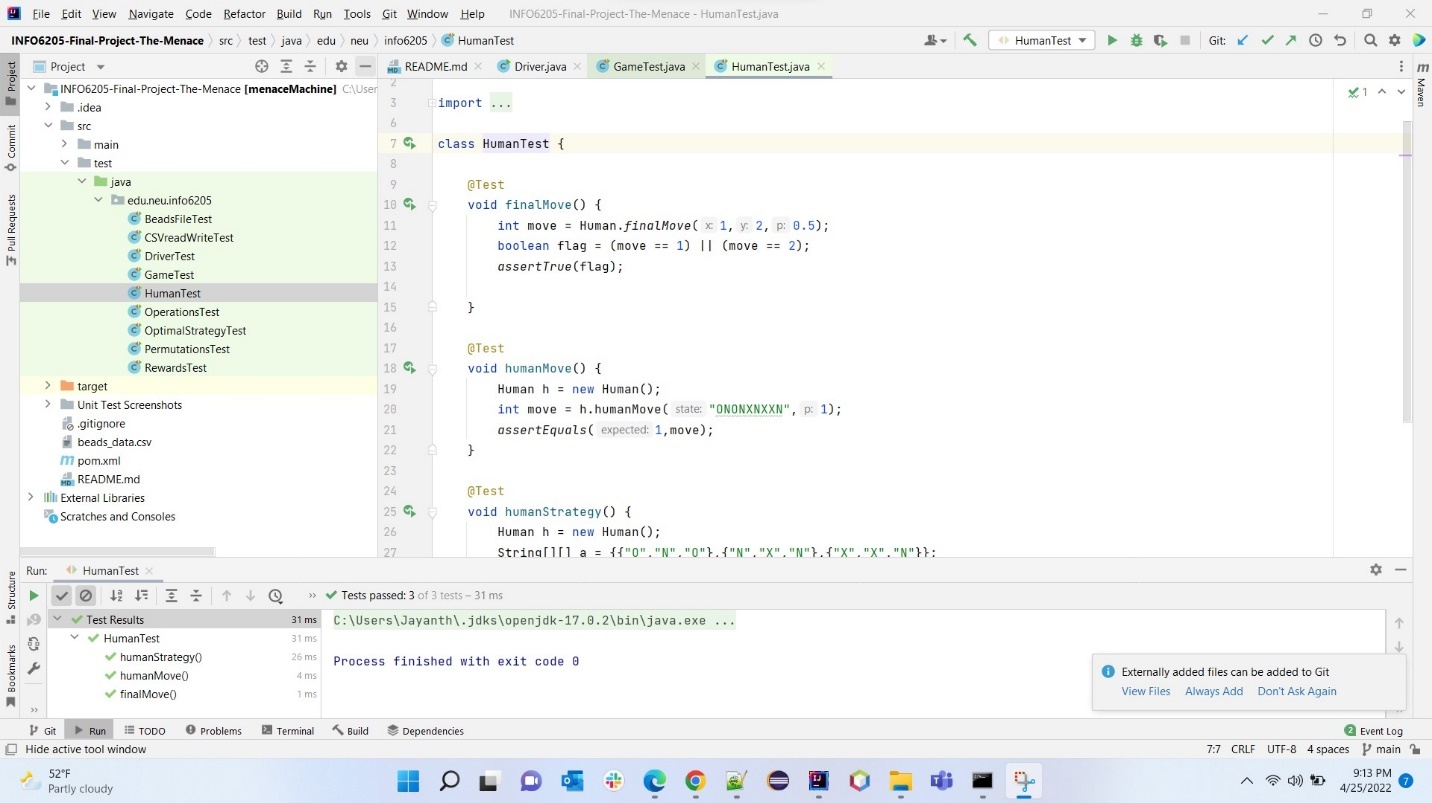
***DriverTest.java***

****

***GameTest.java***

****

***HumanTest.java***

****

**Conclusion**

**References**

* Problem Statement

<https://youtu.be/TtisQ9yZ2zo?t=2210>

* Logic for Optimal Strategy

<https://en.wikipedia.org/wiki/Tic-tac-toe#Strategy>

* Logic for CSV I/O operations

https://www.baeldung.com/opencsv

* sl4j logging framework

<https://www.slf4j.org/manual.html>

* Reinforcement learning elements

<https://www.javatpoint.com/reinforcement-learning>

* (Part 1) Implementation of three methods in *Timer.java,* & check this implementation by running the unit tests in *BenchmarkTest.java*and*TimerTest.java*
* (Part 2) implementation of *InsertionSort* (in the *InsertionSort* class) & check this implementation by running the unit tests in *InsertionSortTest*
* (Part 3) Implementation of a main program to run the following benchmarks: measure the running times of this sort, using four different initial array ordering situations: random, ordered, partially-ordered and reverse-ordered.
* Using doubling method for choosing *n* and test for at least five values of *n*
* Drawing conclusions from the observations regarding the order of growth

**Relationship Conclusion**

* Order of growth of the running time of Insertion Sort (Randomly ordered array of size *N*) is
* Order of growth of the running time of Insertion Sort (Ordered array of size *N*) is
* Order of growth of the running time of Insertion Sort (Partially ordered array of size *N*) is
* Order of growth of the running time of Insertion Sort (Reverse ordered array of size *N*) is
* In terms of order of growth, for the running time of Insertion sort:

**Evidence to the Conclusion**

* Running time of the insertion sort for an array of ‘n’ numbers has been captured
* Each time the size of the array would be doubled and running time would be captured again (5 different sizes of array)
* Every time, we run the insertion sort algorithm, we make sure to test on four different states of the array ()

**Random Ordered Array**

*Various sizes of the Array and the running time of the Insertion sort*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Randomly Ordered Array** | | | | | | |
| **Array Size** | **Time** | **Ratio (Time/Previous Time)** | **lg(Array Size)** | **lg(Time)** | **Log Ratio** | **Slope** |
| 1000 | 1.84 | - | 9.97 | 0.88 | 11.33 |  |
| 2000 | 7.99 | 4.34 | 10.97 | 3.00 | 3.66 | 2.12 |
| 4000 | 24.35 | 3.05 | 11.97 | 4.61 | 2.60 | 1.61 |
| 8000 | 57.85 | 2.38 | 12.97 | 5.85 | 2.21 | 1.25 |
| 16000 | 237.68 | 4.11 | 13.97 | 7.89 | 1.77 | 2.04 |
| 32000 | 1171.63 | 4.93 | 14.97 | 10.19 | 1.47 | 2.30 |
| **Avg slope** | | | | | | 2.04 |

*Analysis of experimental data (the running time of insertion sort with random ordered input)*

**Standard Plot: Running time T(n) Vs Array size N**

**Log-Log Plot: lg(T(n)) Vs lg(N)**

The equation of the log-log plot is

Which is equivalent to,

**Ordered Array**

*Various sizes of the Array and the running time of the Insertion sort*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Ordered Array** | | | | | | |
| **Array Size** | **Time** | **Ratio (Time/Previous Time)** | **lg(Array Size)** | **lg(Time)** | **Log Ratio** | **Slope** |
| 1000 | 0.0054 |  | 9.97 | -7.53 | -1.32 |  |
| 2000 | 0.0077 | 1.43 | 10.97 | -7.02 | -1.56 | 0.51 |
| 4000 | 0.0146 | 1.90 | 11.97 | -6.10 | -1.96 | 0.92 |
| 8000 | 0.0399 | 2.73 | 12.97 | -4.65 | -2.79 | 1.45 |
| 16000 | 0.0637 | 1.60 | 13.97 | -3.97 | -3.52 | 0.67 |
| 32000 | 0.1318 | 2.07 | 14.97 | -2.92 | -5.12 | 1.05 |
| **Avg slope** | | | | | | 0.92 |

*Analysis of experimental data (the running time of insertion sort with random ordered input)*

**Standard Plot: Running time T(n) Vs Array size N**

**Log-Log Plot: lg(T(n)) Vs lg(N)**

The equation of the log-log plot is

Which is equivalent to,

**Partially Ordered Array**

*Various sizes of the Array and the running time of the Insertion sort*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Partially Ordered Array** | | | | | | |
| **Array Size** | **Time** | **Ratio (Time/Previous Time)** | **lg(Array Size)** | **lg(Time)** | **Log Ratio** | **Slope** |
| 1000 | 0.49 |  | 9.97 | -1.03 | -9.68 |  |
| 2000 | 1.83 | 3.73 | 10.97 | 0.87 | 12.58 | 1.90 |
| 4000 | 7.16 | 3.91 | 11.97 | 2.84 | 4.21 | 1.97 |
| 8000 | 27.97 | 3.91 | 12.97 | 4.81 | 2.70 | 1.97 |
| 16000 | 114.85 | 4.11 | 13.97 | 6.84 | 2.04 | 2.04 |
| 32000 | 497.21 | 4.33 | 14.97 | 8.96 | 1.67 | 2.11 |
| **Avg slope** | | | | | | 1.97 |

*Analysis of experimental data (the running time of insertion sort with random ordered input)*

**Standard Plot: Running time T(n) Vs Array size N**

**Log-Log Plot: lg(T(n)) Vs lg(N)**

The equation of the log-log plot is

Which is equivalent to,

**Reverse Ordered Array**

*Various sizes of the Array and the running time of the Insertion sort*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Reverse Ordered Array** | | | | | | |
| **Array Size** | **Time** | **Ratio (Time/Previous Time)** | **lg(Array Size)** | **lg(Time)** | **Log Ratio** | **Slope** |
| 1000 | 1.67 |  | 9.97 | 0.74 | 13.47 |  |
| 2000 | 7.58 | 4.54 | 10.97 | 2.92 | 3.75 | 2.18 |
| 4000 | 28.86 | 3.81 | 11.97 | 4.85 | 2.47 | 1.93 |
| 8000 | 115.39 | 4.00 | 12.97 | 6.85 | 1.89 | 2.00 |
| 16000 | 485.59 | 4.21 | 13.97 | 8.92 | 1.57 | 2.07 |
| 32000 | 2202.95 | 4.54 | 14.97 | 11.11 | 1.35 | 2.18 |
| **Avg slope** | | | | | | 2.07 |

*Analysis of experimental data (the running time of insertion sort with random ordered input)*

**Standard Plot: Running time T(n) Vs Array size N**

**Log-Log Plot: lg(T(n)) Vs lg(N)**

The equation of the log-log plot is

Which is equivalent to,

**Output Screenshot**

**Graphical user interface, text, application

Description automatically generated**

**Graphical user interface, text, application, email

Description automatically generated**

**Output**

\*\*Randomly Ordered Array\*\*

2022-02-12 21:04:30 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 1000, Time: 1.84235

2022-02-12 21:04:30 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 2000, Time: 7.991303333333334

2022-02-12 21:04:31 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 4000, Time: 24.351119999999998

2022-02-12 21:04:32 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 8000, Time: 57.851236666666665

2022-02-12 21:04:33 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 16000, Time: 237.68453333333335

2022-02-12 21:04:41 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 32000, Time: 1171.62858

\*\*Ordered Array\*\*

2022-02-12 21:05:20 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 1000, Time: 0.005353333333333333

2022-02-12 21:05:20 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 2000, Time: 0.007703333333333334

2022-02-12 21:05:20 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 4000, Time: 0.014633333333333333

2022-02-12 21:05:20 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 8000, Time: 0.03994333333333333

2022-02-12 21:05:20 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 16000, Time: 0.06371666666666667

2022-02-12 21:05:20 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 32000, Time: 0.13181666666666667

\*\*Partially Ordered Array\*\*

2022-02-12 21:05:20 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 1000, Time: 0.48889

2022-02-12 21:05:20 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 2000, Time: 1.82751

2022-02-12 21:05:20 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 4000, Time: 7.16458

2022-02-12 21:05:20 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 8000, Time: 27.970056666666668

2022-02-12 21:05:21 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 16000, Time: 114.85384666666667

2022-02-12 21:05:25 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 32000, Time: 497.20865

\*\*Reverse Ordered Array\*\*

2022-02-12 21:05:42 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 1000, Time: 1.6678833333333332

2022-02-12 21:05:42 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 2000, Time: 7.575003333333333

2022-02-12 21:05:42 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 4000, Time: 28.861373333333333

2022-02-12 21:05:43 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 8000, Time: 115.39204

2022-02-12 21:05:47 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 16000, Time: 485.58698

2022-02-12 21:06:03 INFO Benchmark\_Timer - Begin run: Insertion Sort with 30 runs

N= 32000, Time: 2202.9538833333336

Process finished with exit code 0

**Unit Tests**

* ***TimerTest***

Graphical user interface, text, application, email

Description automatically generated

* ***BenchmarkTest***

Graphical user interface, text, application, email

Description automatically generated

* ***InsertionSortTest***

Graphical user interface, text, application

Description automatically generated