**Runtime Complexity Analysis of RabbitSimulation.java**

To estimate the runtime efficiency of my RabbitSimulation.java class and its methods, I have assumed that all basic operations take the same amount of constant time.

**Results:**

*main* – O(n2)

*rabbitSimulation* – O(n)

*avg* – O(n)

*stdDev* – O(n)

End result: The RabbitSimulation java class has a complexity of O(n2).

**Analysis of Individual Methods**

main Method:

A single *for* loop has a complexity of O(n), with n being a number proportionate to the number of iterations or items being looped over. The most complex structure in this method is a nested *for* loop. This is because for every iteration of the outside loop, there is an entire loop to be completed before the outside loop can move on to the next iteration. In this case, I have two different *for* loops nested inside the outside *for* loop.

The first *for* loop itself can be approximated as having a runtime complexity of O(1) because it is a loop that loops 10 times every time. However, since the *rabbitSimulation* method is called for every iteration in this loop and *rabbitSimulation* has a runtime complexity of O(n), the total complexity of the *main* method increases to O(n2).

There is another *for* loop nested in the outside-most loop that loops over the size of an ArrayList of undetermined magnitude. This would mean this portion of the *main* method has a complexity of O(n2), which does not change the result.

The *.avg* and *.stdDev* methods that are present in the outer-most *for* loop (used in the final print statements) have complexities of O(n) (covered later), so it does not change the result. No methods inside of these loops have a complexity larger than O(1), so this result is final – the complexity of the *main* method is O(n2).

rabbitSimulation Method:

The most complex structure in the rabbitSimulation method is a triple nested *for* loop. However, there is a catch – the runtime complexity of this method is not O(n2) – it is O(n).

The outside-most *for* loop loops exactly 365 times, to represent the number of days that pass. This will prove to be inconsequential in terms of the *for* loop inside of this loop. The loop that lies inside of the loop that increments once for each day that passes loops over an ArrayList containing all the Rabbit objects designated in the parameters of the method. Not only is the number of rabbits an unknown number determined by the text file that is read in by the program, but it grows exponentially over time due to the defined behavior of the rabbits (the females may give birth to a litter of 3-8 kits about every 35-45 days or so). This means that the ArrayList holding the Rabbit objects could contain thousands, tens of thousands, or maybe even hundreds of thousands of Rabbit objects. This would mean the runtime complexity of this loop is O(n). The *.add, .get* and *.size* methods for ArrayLists have complexities of O(1). Furthermore, the *.getAge, .incrementAge, getGestTime, .getDUF, .getSex, .decrementDUF, .impregnate, .decrementGestTime, .resetGestTime* and *.resetDUF* methods of the Rabbit class all have a complexity of O(1). The *.add* method of the ArrayList class also has a runtime complexity of O(1). Finally, the *.nextInt* method of the Random class has a complexity of O(1), meaning that nothing inside any of the loops contribute significantly to the overall complexity, and only the loops themselves do.

There is a *for* loop inside of the loop that loops over all of the Rabbit objects, but it has a defined number of iterations that always lies between 3 and 8, meaning it has a complexity of O(1). Not only this, but the methods inside of it have been explained to have complexities of no larger than O(1), meaning this loop does not contribute significantly to the overall complexity of the method, just like the outer-most loop.

avg Method:

This method contains one *for* loop over an ArrayList of an unspecified length, making the complexity O(n). No other actions performed inside this method involve a complexity of more than O(1), so the result is final.

stdDev Method:

This method calls the *avg* method, but it is not inside any loop and does not involve anything that has a complexity larger than O(n). So far, the complexity is O(n), since the complexity of the *avg* method is O(n).

There is a *for* loop that loops over an ArrayList of an unspecified length, meaning the complexity of this portion of the method is O(n). Thus, the total complexity of the *stdDev* method is O(n), since the *avg* method call and the *for* loop have the same complexity and are not involved with one another.

**Can the Complexity be Improved?**

Main Method:

There is no way to decrease the estimated complexity. The rabbitSimulation method must be called one time for each trial (row of the text file that is filled). This requires rabbitSimulation, a method with a complexity of O(n), to be called an unknown number of times, meaning in a worst-case scenario the complexity of the *main* method will be O(n2).

rabbitSimulation Method:

There is no way to decrease the estimated complexity. A loop over all the Rabbit objects must be completed for every day that passes to check which rabbits can give birth and act accordingly. This means there is no way to improve the estimated complexity past O(n), since an unknown number of rabbits are given to start with, and the ArrayList of rabbits increases exponentially with time.

avg Method:

There is no way to decrease the estimated complexity. To find the average of an unknown number of integers requires a *for* loop to loop over some sort of container that contains an unknown number of integers. This means that the complexity cannot be decreased past O(n).

stdDev Method:

There is no way to decrease the estimated complexity due to the same reason as the *avg* method’s complexity cannot be decreased past O(n) – a loop must loop over a container of an unknown number of integers to find the standard deviation of the integers in that container.

**Conclusion:**

Overall, a few minute optimizations in the conditional statements and methods used in the class overall may slightly decrease the complexity, but in the end, it would prove negligible, as no large decreases in runtime complexity in any of the methods can be made without sacrificing some of the things that the program does.