Version 1 Reflection:

I ran into some issues while I was programming, and it took me a long time to figure out what I was doing wrong. My code was breaking as it was iterating through the Array. I managed to fix it by including the Array list within the main method itself. I ran the completed program 10 times to ensure that it was working correctly. Then I extracted that algorithm and created another program to run this algorithm 100 times to test the percentage breakdown of it being truly 50/50, a coin flip. It was close enough that it can be counted as 50/50 due to the low number of trials.

Version 2 Reflection:

In this version, I enhanced the contestant class to include the position and totalPosition records. The recordPosition method is used to record the position of each contestant at the end of each round, and the calculateAveragePosition method calculates the average position over all rounds.

The program now iterates through multiple rounds, recording and displaying the positions of contestants in each round. Then, it calculates and prints the average positions for each contestant, demonstrating that the algorithm does not bias any contestant towards one end of the line for most of the contest.

Version 3:

With 43 contestants:

|  |  |
| --- | --- |
| Number of Rounds | Time in ms |
| 10 | 51 |
| 100 | 273 |
| 1000 | 2232 |
| 10000 | 21302 |

A graph with a line

Description automatically generated

Version 4:

With 45,000 contestants:

|  |  |
| --- | --- |
| Number of Rounds | Time in ms |
| 10 | 318 |
| 100 | 2866 |
| 1000 | 27672 |
| 10000 | 277346 |

A graph with a line

Description automatically generated

Q1. The folks running the game don’t like the optics of moving people to the front or end of the line each time. Think about a different algorithm for moving people around in the list. You can't decrease the size of the list and the list will get rearranged in some way during each round. Write a paragraph or two on what would have to change in your program to implement the new algorithm, still using an ArrayList. Be as specific as you can with regard to your code. Consider whether it would be easy or hard to implement and estimate whether it will take more or less time to run than the current implementation. Be sure to explain your reasons.

**To implement a different algorithm that avoids the optics of moving people to the front or end of the line in each round, we can consider a swap-based approach. Instead of rearranging the entire list, we can randomly select pairs of contestants and swap their positions. This can be achieved by iterating through the list and swapping the position of each contestant with another randomly chosen contestant. This way, individuals move within the list without being consistently pushed to the extremes.** T**he code needs modifications in the section where new positions are calculated. Instead of creating a new list and populating it with contestants based on correctness, we would iterate through the existing list and swap positions between contestants. The swapping process should be randomized to ensure a dynamic rearrangement in each round.**

**The implementation is relatively straightforward but requires careful handling of the swapping logic. This approach may take slightly more time to run compared to the original algorithm because the swap operation involves more computations than simply adding or removing elements from a list. However, the difference in runtime may not be substantial, and the trade-off is made for the sake of the desired optics in the game.**

Q1. In a sentence or two, what did you learn?

In a sentence or two, what did you like about this project?

In a sentence or two, what did you find confusing or would like to see done differently regarding this project?

In a sentence or two, if you had another hour or two, what would you like to add to the project or how would you do things differently?

**I learned to be more careful in my programming choices. I made a major change between versions 3 and 4 where I stopped the code from printing out the locations specifically of each contestant, but still recording it. This way the program would have a significantly faster runtime and be more manageable to test with.**

**I really liked that there was an exercise and not just a generic programming experiment. It walked us through an idea and the thought process behind it, but it still left us free to learn and experiment.**

**Sometimes the verbiage of the requirements was confusing to me, but I really just needed to keep trying and approach them in a different way in order to see results.**

**If we had more time, I’d like to play around more with the runtime using a different algorithm. Rather than using a coinflip to determine if they move to the back or stay at the front, I’d like to see the runtime whilst using the swapping algorithm I mentioned earlier. Or even a rotational algorithm dividing the contestants into small groups with a positional value that rotates as they get questions right or wrong.**