Refactoring the Queue Simulator

Overall in my queue simulator there are three main areas I would refactor. Frist, I would create a driver class to run the simulator and simplify the API. Next, I would rewrite how I handle customers who's wait time exceeds the amount of time in a day to simplify my run-time complexity, and I would change the data structure used to store wait times to eliminate the need to sort the times after the data is generated. Finally, I would remove unneeded functions which I wrote but never used.

All of the code which drives the simulator takes place in main(), including the code which creates the random service times for the customers and the loop which runs the simulator. The loop also contains a nested loop to handle the bank queue. I could take that additional loop and place it inside of the bank class in order to help abstract and simplify the bank queue. Additionally, I would take the code used to format and print the results to the command line and place the code into additional methods within the driver class.

As my simulator executes, customers with wait times greater than the amount of time in a day are given a total wait time of -1, and after the simulator loop finishes all -1 values are removed. This adds additional operations and increases the run-time of the simulator. If I add a few more conditional statements I could avoid adding those -1 values to the vectors of wait times, eliminating the need to remove them in the first place. Additionally, I have to sort the vectors of wait times in order to determine the 10^{th}, 50^{th}, and 90^{th} percentile. If I change the data structure to store the wait times in a binary tree, I could implement binary insertion sort to insert the wait time in-order, eliminating the need to sort the wait times and reducing complexity.

Finally, I wrote a function that allows customers to enter the queue based on the customer arrival rate and random number generation. I ultimately did not used this method as it added more complexity and did not produce more useful results than simply letting customers enter the queue in an evenly spaced manner based on the customer arrival rate.