For office use only
T1 _____
T2 _____
T3 _____
T4 _____

For office use only
F1 _____
F2 _____
F3 _____
F4 _____

2013

16th Annual High School Mathematical Contest in Modeling (HiMCM) Summary Sheet
(Please attach a copy of this page to your Solution Paper.)

Team Control Number: 4093
Problem Chosen: B

Please type a summary of your results on this page. Please remember not to include the name of your school, advisor, or team members on this page.

In a theoretical bank, a manager wishes to improve service and wants the average customer to wait less than 2 minutes for service and the average length of the waiting line to be 2 persons or fewer. Our model was designed to simulate the current conditions of the bank and to determine to what degree the service should be improved to meet the required conditions if the current services were ineffective.

We decided to model this problem using a computer program written in Java that would simulate, using a loop, an actual bank day in which 150 customers entered the bank and were processed. Then, we determined how many people were in line and stored the values in an array for later access; we also calculated the time that each person waited and stored those values into an array. However, the calculations were rather complex so instead of making a queue in a computer program, we stored each customer's arrival time, waiting time and leaving time as elements of arrays because we discarded the customer at the end of each loop. However, we were surprised at the data generated; in fact, so much so that we created another program that used altered logic with the same probabilities to confirm the results. After ten runs, results were consistent with our original model. Therefore, confident of the legitimacy of our results, we began to brainstorm ways to improve service.

However, we encountered a problem that forced us to optimize the bank service in two different manners, not one. The problem was that we were uncertain whether the manager wanted every "average customer" to wait less than 2 minutes for service, a requirement that would be theoretically impossible, or whether the manager wanted each customer to have an average wait time of less than 2 minutes for service, a requirement that could be more easily optimized and practically implemented. Ultimately, we were able to brainstorm two solutions to each of these conditions that would maintain reasonable burden on workers but provide a satisfying business-growing experience to customers. The solution for the first interpretation optimized the bank service so that 90% of all average customers wait in line for less than two minutes (35% two-minute service time and 65% one-minute service time). The solution for the second interpretation optimized the bank service so that all customers wait in line for an average of less than two minutes (40% three-minute service time, and in increase in two-minute service time by 5% and one-minute service time by 10%).

HiMCM 2013 Problem B

Table of Contents

Introduction

In a service based industry such as the field of banking, customer satisfaction is always a key concern when considering business efficiency. One of the leading complaints customers have with banks is the waiting time. A customer looking to use a bank's services usually wants to do one of a few actions: withdraw money, deposit checks and cash, or change or open an account. However, these processes take time that can add up, leaving the customers in the back of the line wondering if they will be able to spare enough time from their busy day to use a bank. Thus, an inefficient queuing system can cost the bank in lost opportunities, such as in transactions that do not occur because a customer does not want to wait in line, or in lost customers who leave because they feel that they can get better treatment at another bank. Therefore, both waiting time and queue length must be considered when evaluating bank efficiency. Our overarching goal for this model is to be able to effectively model bank users at a given model bank and optimize this bank's ability to serve customers well.

Restatement of the Problem

1. Build a mathematical model that ensures that the customers will wait no more than two minutes for service and that no more than two people will be in each service queue. The model will run for the estimated 150 customers per day and the probability of time between arrival and service time will be represented by the data chart below.
2. Determine whether current customer service meets the manager's aforementioned service guidelines, and model the minimum changes needed so that servers can meet manager guidelines if they do not already.
3. Write a non-technical letter to the manager concerning final recommendations.

Time between Arrival (minutes)	Probability	Service Time (minutes)	Probability
0	0.10	1	0.25
1	0.15	2	0.20
2	0.10	3	0.40
3	0.35	4	0.15
4	0.25		
5	0.05		

Assumptions and Justifications

1. Banks are closed on Sundays.

   1.1. Justification: Though most banks are closed on Sundays, some, such as the East West Bank, are not. Assuming that all banks are closed on Sundays would allow us not to have to work around a small minority of banks while still making our model fairly representative of all banks. Thus, when we model the theoretical bank's average opening time per day, we base our assumption on a six day week.

2. The top three banks by assets in America (Chase, Bank of America, and Citibank) are representative of all banks in America.

   2.1. Justification: Since most other banks below the top three in America are investment banks or credit card institutions, such an assumption would still be fairly accurate when applied to the real world. In addition, the top three banks, in terms of assets, comprise a large majority of the banking market.

3. There is only one queue in the bank.

   3.1. Justification: Assuming that there is only one queue in the bank helps simplify our model, thus reducing the complexity of results and number of variables in the model. Furthermore, the assumption

4. There is only one teller in the bank.

   4.1. Justification: Assuming that there is only one teller in the bank helps simplify our model and is only logical if we were to assume that there is only one queue in the bank.

5. There are no exceptions, interruptions or other events that could interrupt the time between arrival and service time of the bank during each simulation. The probabilities of each time between arrival and service time stay constant.

   5.1. Justification: Accounting for rare exceptions in the model is inefficient and difficult to model.

6. There are only 150 customers who enter the bank per day.

   6.1. Justification. The problem uses this assumption because there is no established algorithm to determine a bank's change in inflow of customers from day to day throughout the week.

Variables

Independent Variables

Arrival Time Between Customers:

The arrival time between customers for our bank can be calculated by a pseudorandom number generated within our code, determining what is the delay time between customers according the probabilities in the problem's table.

Service Time:

The service time that each customer uses is an independent variable because it is independent of any other influence. In a real bank, customers utilize a number of services, such as depositing cheques, withdrawing money, maintaining their account, creating new accounts, etc. Because the different services that customers utilize require different amounts of time, it is impossible to model them discretely and instead an average service time is used. The service time must remain in a certain interval if the customers are not to wait for 2 minutes or more. For example, if two customers came in at the same time, then the service time of the first customer must be less than two minutes if the second customer is not to wait less than two minutes.

Time Elapsed Since Last Customer:

The time elapsed since the last customer is the time elapsed customer A walks in and Customer B walks in. This matters because the service time of Customer A minus the time elapsed since last customer equals the time that Customer B will have to wait in line.The time elapsed since last customer is a random number generated in our model, since in real life the time between customers is random and independent.

Timestamp:

The timestamp, or what time it is right now, is the input of the random number generator that we use in our model that gets converted into wait time and queue length.

Dependent Variables

Wait time:

The time that each customer will have to wait in line. This equals the time that service takes for customer A minus the time elapsed since last customer. In our model, we use timestamps in our random number generator to generate time elapsed and service time, under certain parameters to find the wait time. Our goal is to have the limit(wait time minus 2 minutes) to go to zero.

Line length:

The length of the line is based of service time and arrival time between customers. The line length is based on wait time which is in turn dependent on time elapsed since last customer and the service time.

Hypotheses

After modeling the current demands upon the branch, we have determined that the current service platform will not allow the bank to meet its goals:

*Wait time less than two minutes for any customer: we have determined that, on an average day, between 110 and 130 customers will have to wait in line for at least two minutes, assuming the current service platform. The modeled average wait time for any given customer is about 7 minutes.

*Queue no longer than two persons at any time: we have determined that, on an average operating day of 501 minutes, the line will be at least three people long for about 144 minutes from the time the first customer enters to the time the last customer leaves, assuming the current service platform. The modeled average line length at any given minute of the day is about 3 persons.

Growth will be limited until customers can be satisfied.

With this in mind, we have found that the best way to optimize customer service is to increase the speed at which customers are processed. Thus, we recommend worker training modules to speed worker's transaction capabilities. The bank should eliminate four-minute service times (currently 15% of business) and transfer that 15% to three-minute service times, then eliminate 15% of three-minute service times and transfer that 15% to two-minute service times, and so forth (essentially eliminating four-minute services and transferring the excess 15% up the service ladder one step at a time until the mix of service times stood at 40% for one minute, 20% for two, and 40% for three) through worker efficiency training.

Even then, however, more than a third of customers would wait for longer than two minutes and the line would be longer than 2 persons for about an hour each day. Therefore, we recommend achieving a final 65%/35% split between one minute and two minute service times. One opportunity to achieve this might be to parallelize tasks: many banking customers walk into branches seeking to service accounts and to withdraw or deposit money; however, account servicing requires a platform banker, while balance modification requires a teller. In a traditional branch, the customer would visit the teller first and then the banker, but the bank instead would

be able to save time by allowing the customer to fill out a balance slip when he entered (only if there was a line), and then routing any customer with an account service request to the banker first. The banker would submit the customer's balance slip to the tellers if there was a line or, if there was no line, simply perform the maintenance request and then send the customer to the teller as usual. In this way, customers who only wished to deposit or withdraw would quickly be able to do so if there were no line, but those with maintenance requests would be able to avoid longer lines since their money would be deposited or withdrawn while they themselves discussed their accounts with a banker.

Realistically, however, these changes will require ambitious efforts that will tax the bank and might strain the bank's workers. Instead, it might consider hiring one more teller to process quick requests and one more banker to service accounts. Ultimately, the bank might simply wish to lower your standards: training workers, parallelizing tasks, and hiring more staff will allow its branch to proceed so that only a few customers each day would have to wait for about three minutes in lines of three or less while most would wait less than two minutes in lines of two or less. At the same time, the workers would not feel overburdened and would be able to continue providing your customers with cheerful, useful service.

Analysis of Problem and Contemplation of Various Modeling Methods

The nature of the problem requires that we carry out a simulation using probability-based random numbers to model the problem. There are three such ways to accomplish this task. First, geometrically: we could make two dart boards whose different regions represent different times between arrival and service time, respectively, and throw darts at each board in sets of 150 to generate random "average" customers to model a bank day with 150 customers. This is a highly inefficient method of modeling the problem whose results are completely dependent on our ability to throw darts accurately or inaccurately. This geometric probability method is also not the purpose of this competition.

Secondly, algebraically: we can come up with a set of equations to randomly simulate various intervals between arrival and intervals of service. Though this method is much more mathematical and objective than the previous method, it is still very inefficient, as it would require copious amounts of calculations. Additionally, careless mathematical errors would make the model inaccurate and skew results.

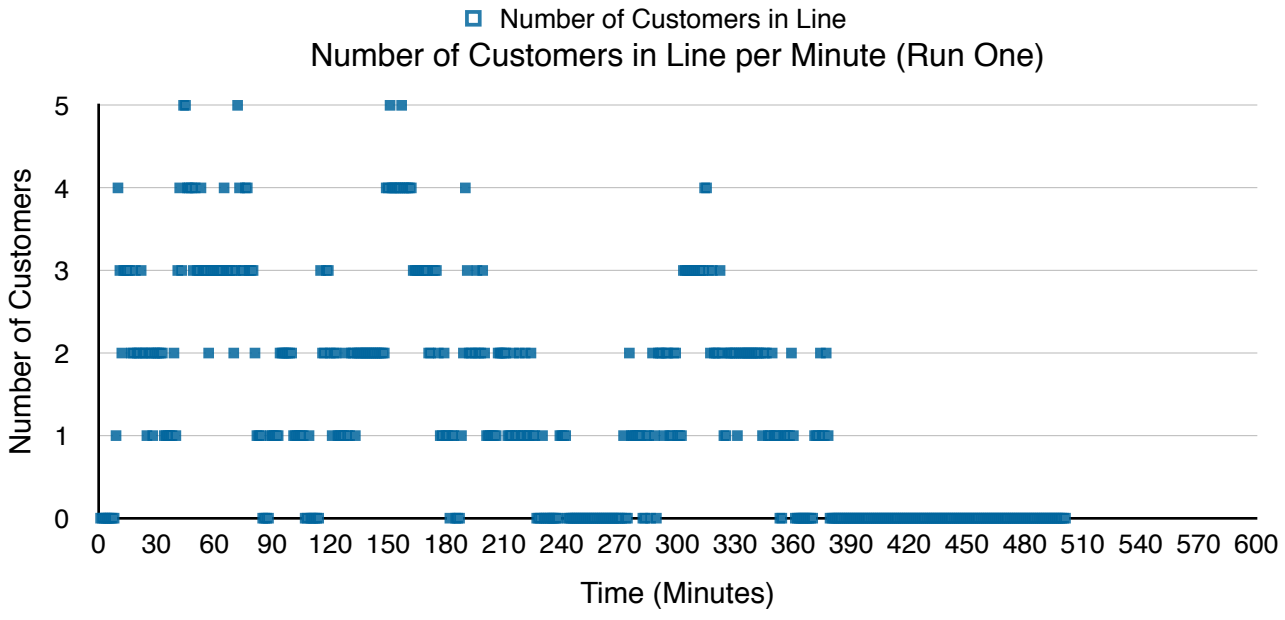
A third option is to code a program that can randomly generate numbers based on the given probabilities and correlated to given times between arrival and service times. We believe

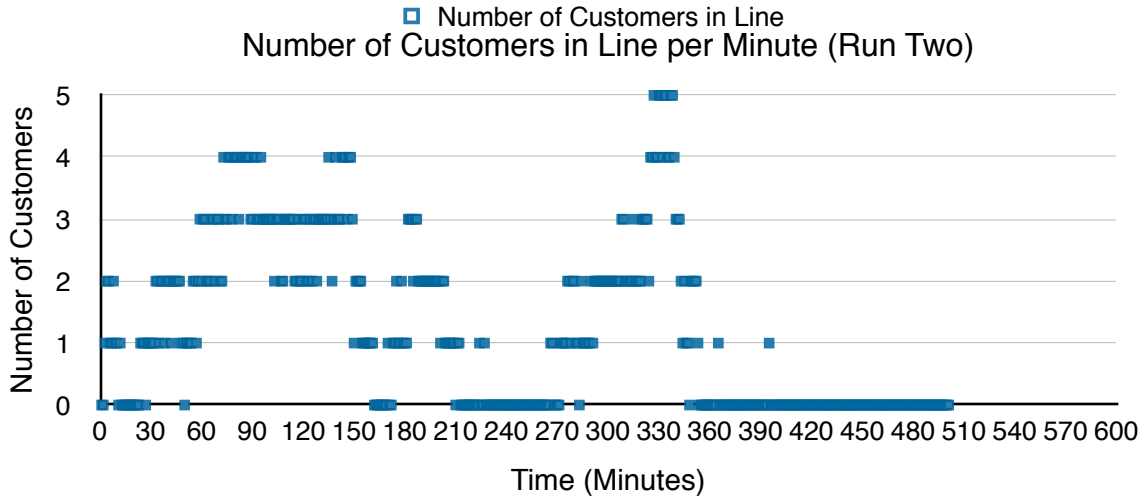
this option to be the best of the three. This method is highly objective, efficient and mathematical. Though a truly random number can never be reached, computer programs can offer the closest thing to a random number, and therefore are highly objective. Furthermore, after the initial time taken to code the program, we can efficiently simulate many problem models, and when optimization is needed, we can easily modify and rerun the program to find conditions when the manager's guidelines are met. Also, computer science is a highly mathematical field, and thus is well suited as a mathematical model.

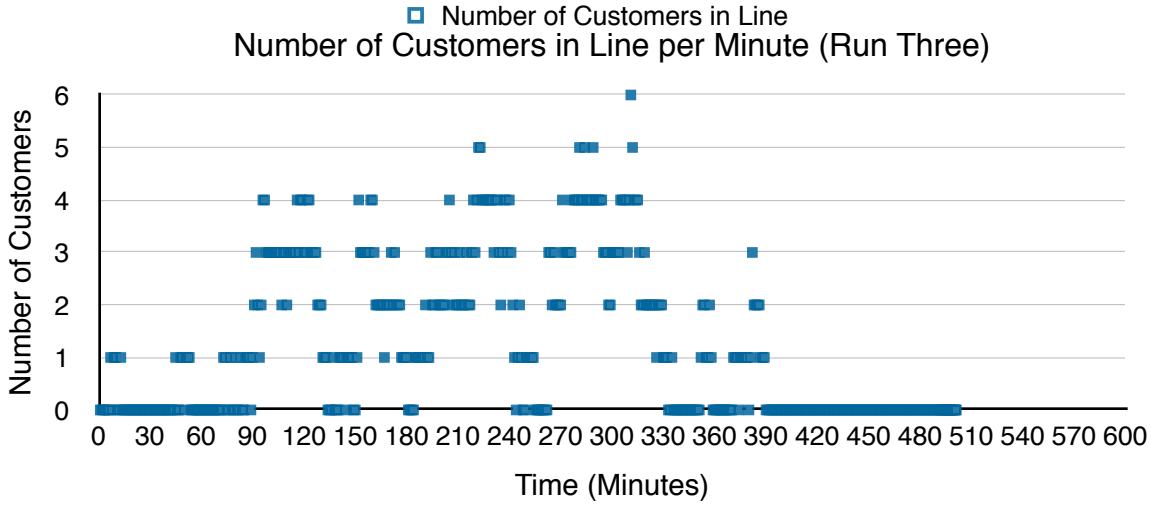
Thus, through an analysis of the two possible interpretations of the problems and of the three various ways through which we can model the problem, we have decided to model both interpretations of the problem with a computer program. This allows us to not only assess all natures of the problem as perceived by us in a mathematical, objective and efficient fashion. By doing so, we hope that we can gather accurate results and produce an effective, yet minimal, model that meets the manager's guidelines.

Our Mathematical Model

We designed a computer program written in Java (Appendix A) and compiled in JCreator. The program can generate two pseudorandom numbers that would determine, based on the probabilities provided in the problem, the time between arrivals of each customer as well as how long each customer's service time was. The program was run through the Java compiler for 150 customers as specified, an average bank day was calculated to be approximately 8.4 hours (**Appendix C**) and five data sets were taken. A more comprehensive evaluation of the program logic can be found in (**Appendix B**). Multiple data sets were taken in order to reduce the effect of error upon the average of the five data sets. Results for the existing time between arrivals and service time are demonstrated below in both graph form and data table. The first three data sets were graphed.







	Data Set 1	Data Set 2	Data Set 3	Data Set 4	Data Set 5	Average
Number of Customers who Waited at Least 2 Minutes	147	106	95	113	135	119.2
Number of Minutes Line Exceeded 2 Customers	308	79	33	162	140	144.4
Average Wait Time Per Customer	18.413	4.06	2.98	3.4	6.093	6.989
Average Line Length	7.424	1.558	1.064	1.256	2.374	2.735

Each average wait time per customer and average line length was the average of 150 customers. We then took the average of each simulation we ran to determine a more accurate approximation of average wait time per customer and average line length.

Optimizing Our Models

We have found two different possible ways to interpret it. **The first interpretation is that the average customer cannot wait in line more than two minutes and the line length must never exceed two customers.** However, it is near impossible to reach 100% efficiency in this interpretation, and the probability that the solution will never have an individual wait in line for two minutes or more. Therefore the probability that the solution exists is extremely hard for us to reach. Therefore, we are aiming to modify the bank services as many customers as possible to wait less than 2 minutes in line, and we found that we can obtain about 90% of customers to wait less than two minutes, and if we have 90% of customers waiting less than two minutes, the line will always have an average of two or less people. According to our optimization model (represented by Definition 1), the **manager would need to have service times of 3 and 4 minutes cut, service time of 2 minutes be at most 35%, and service time of 1 minute be at least 65% to meet the goal**. As a result of the sheer impracticality of the service time cuts, and the fact that the manager's guidelines in this interpretation would never be met, we developed a second interpretation to optimize. **The second interpretation of the problem is that the average wait time of all customers is less than two minutes and the line must never exceed two customers.** Under this interpretation, the average line length was always less than two. According to our optimization model for this interpretation (Definition 2), **the service time of 4 minutes would need to eliminated, the service time of 1 minute increased by 10% and the service time of 2 minutes increased by 5%** to achieve the minimal changes necessary to achieve the manager's guidelines.

Optimization 2: Decrease probability of service time of 4 minutes by 10%, increase probability of service time of 3 minutes by 10%

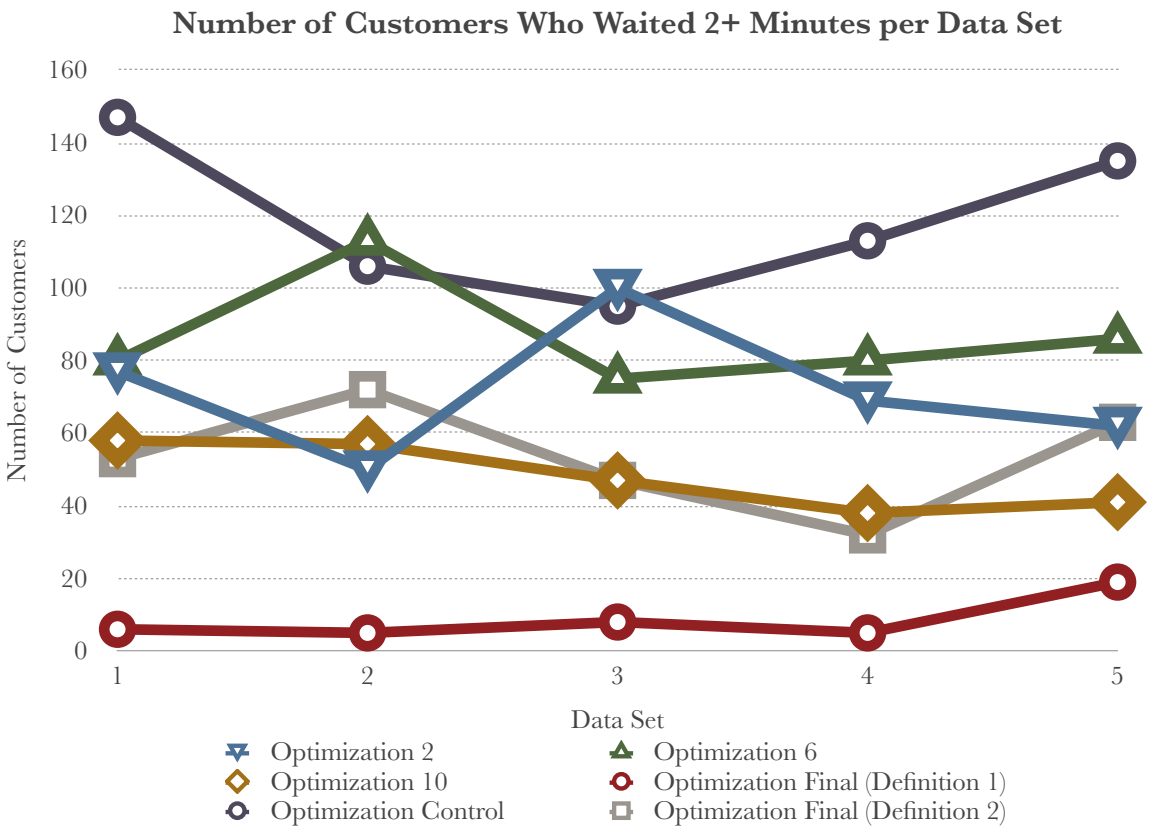
Optimization 6: Decrease probability of service time of 4 minutes by 15%, increase probability of service time of 3 minutes by 5%, increase probability of service time of 2 minutes by 10%

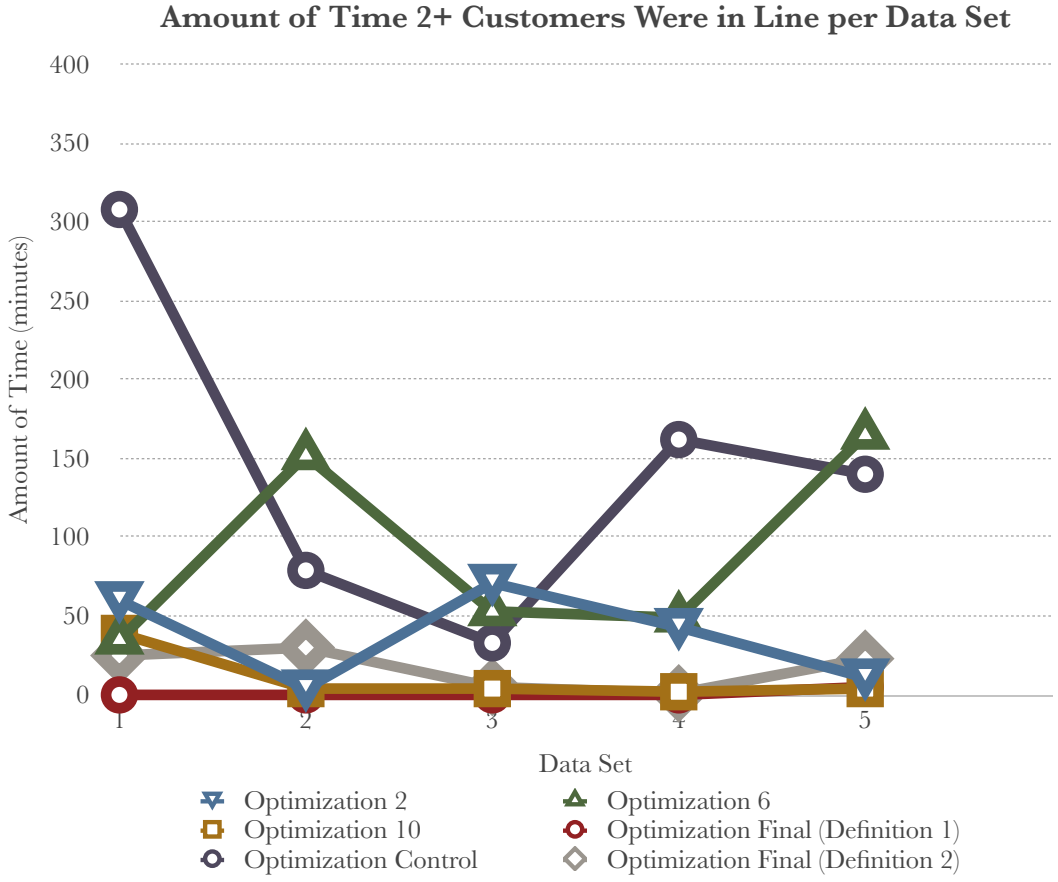
Optimization 10: Decrease probability of service time of 4 minutes by 15%, increase probability of 1 minute by 15%.

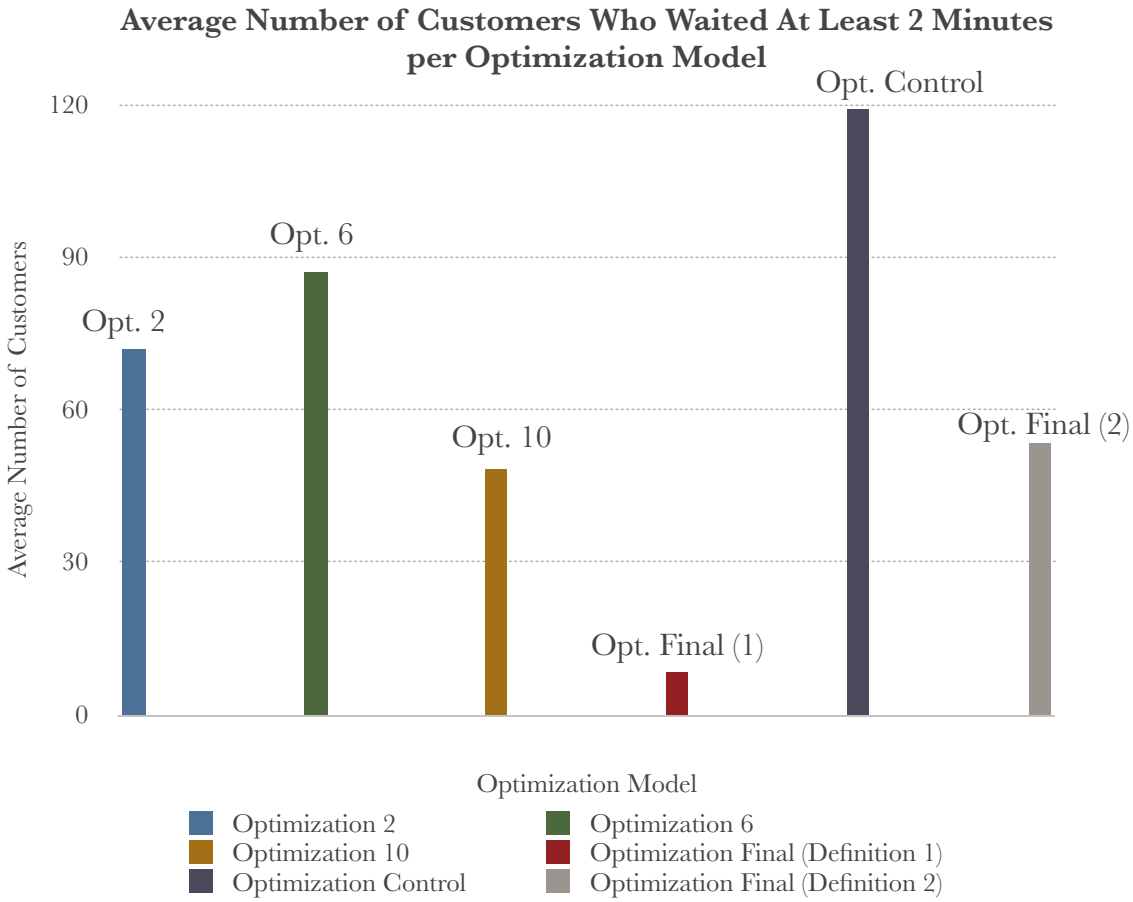
Control: Manager's original service Time.

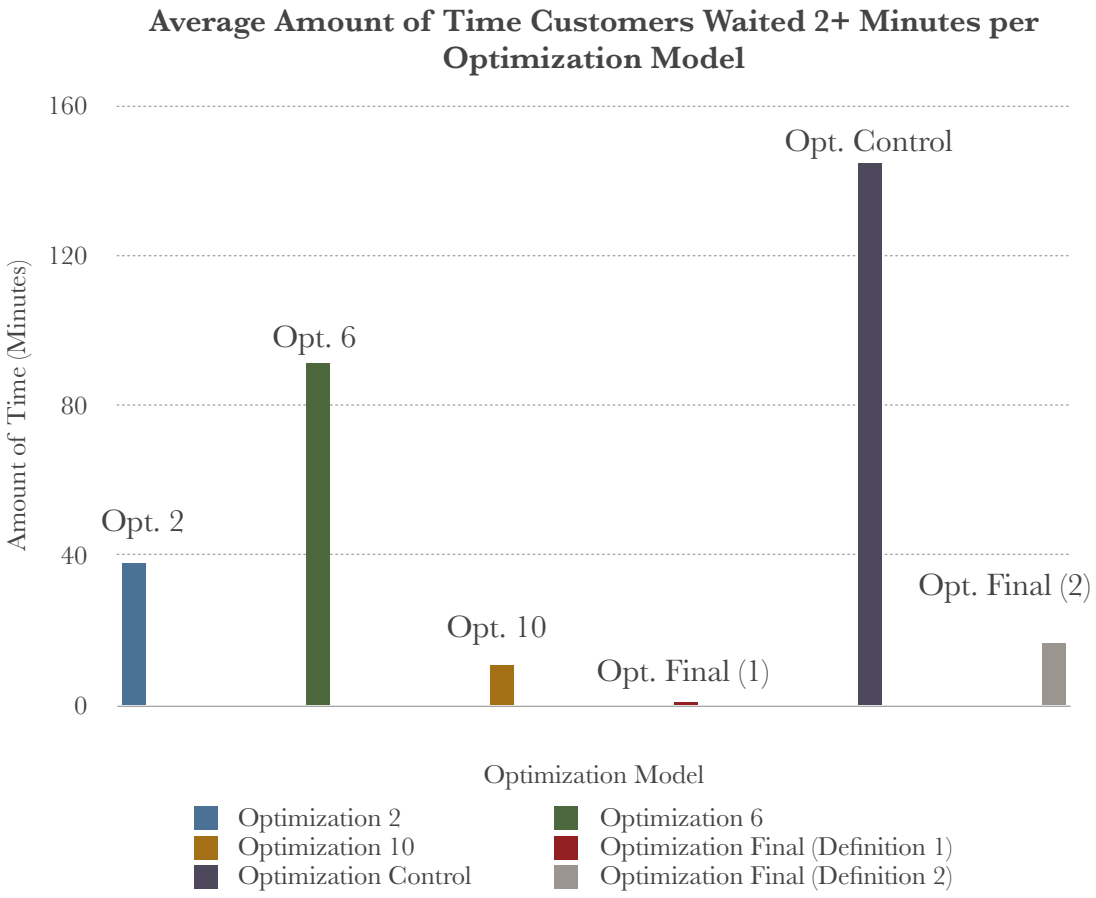
Definition 1: Optimization based off of first interpretation.

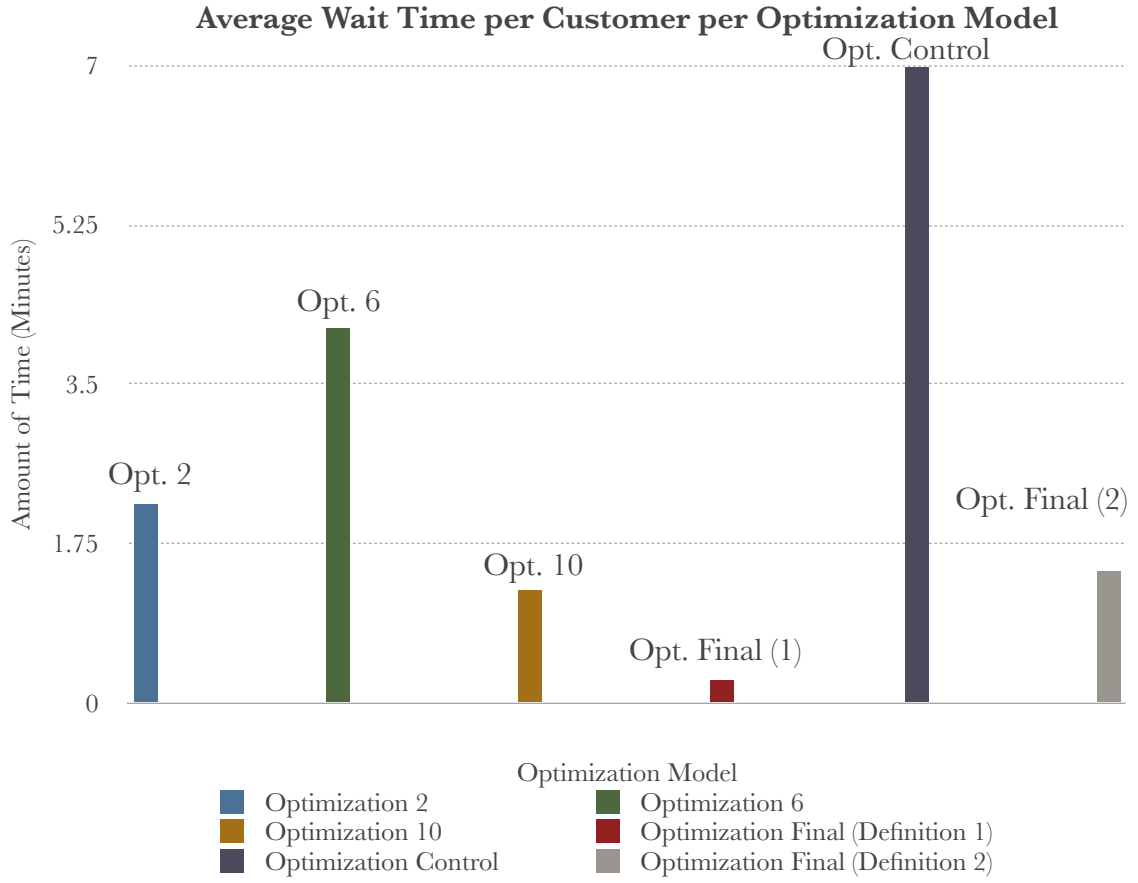
Definition 2: Optimization based off of second interpretation

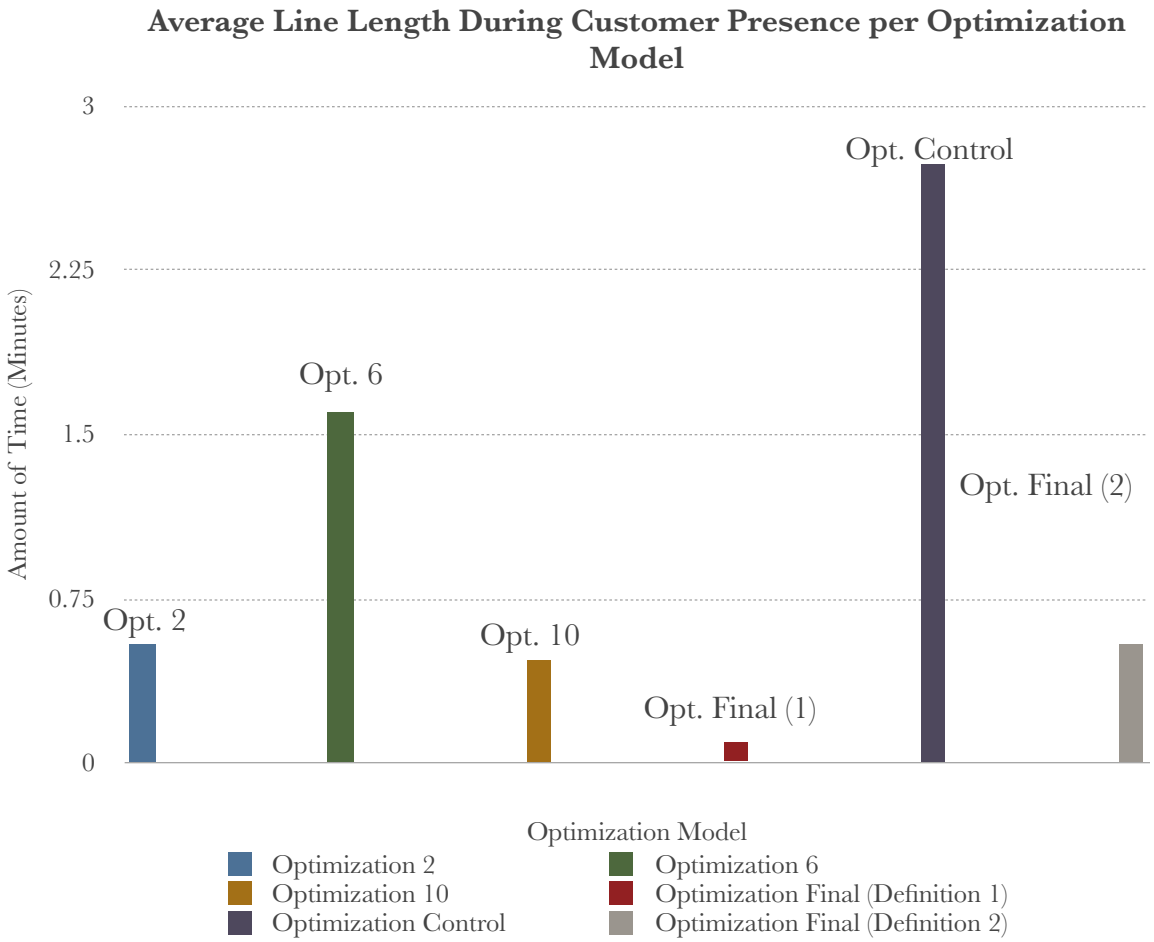

Number of Customers Who Waited 2+ Minutes per Data Set
160
140
120
100
80
60
40
20
0
Number of Customers
1
2
3
4
5
Data Set
Optimization 2
Optimization 6
Optimization 10
Optimization Final (Definition 1)
Optimization Control
Optimization Final (Definition 2)


Amount of Time 2+ Customers Were in Line per Data Set
400
350
300
250
200
150
100
50
0
Amount of Time (minutes)
1
2
3
4
5
Data Set
Optimization 2
Optimization 6
Optimization 10
Optimization Final (Definition 1)
Optimization Control
Optimization Final (Definition 2)


Average Number of Customers Who Waited At Least 2 Minutes per Optimization Model
120
90
60
30
0
Average Number of Customers
Opt. 2
Opt. 6
Opt. 10
Opt. Final (1)
Opt. Control
Opt. Final (2)
Optimization Model
Optimization 2
Optimization 6
Optimization 10
Optimization Final (Definition 1)
Optimization Control
Optimization Final (Definition 2)


Average Amount of Time Customers Waited 2+ Minutes per Optimization Model
160
120
80
40
0
Amount of Time (Minutes)
Opt. 2
Opt. 6
Opt. 10
Opt. Final (1)
Opt. Control
Opt. Final (2)
Optimization Model
Optimization 2
Optimization 6
Optimization 10
Optimization Final (Definition 1)
Optimization Control
Optimization Final (Definition 2)


Average Wait Time per Customer per Optimization Model
Amount of Time (Minutes)
7
5.25
3.5
1.75
0
Opt. 2
Opt. 6
Opt. 10
Opt. Final (1)
Opt. Control
Opt. Final (2)
Optimization Model
Optimization 2
Optimization 6
Optimization 10
Optimization Final (Definition 1)
Optimization Control
Optimization Final (Definition 2)


Average Line Length During Customer Presence per Optimization Model
Amount of Time (Minutes)
3
2.25
1.5
0.75
0
Opt. 2
Opt. 6
Opt. 10
Opt. Final (1)
Opt. Control
Opt. Final (2)
Optimization Model
Optimization 2
Optimization 6
Optimization 10
Optimization Final (Definition 1)
Optimization Control
Optimization Final (Definition 2)

Strengths and Weaknesses

Strengths	Weaknesses
Running the simulation on a computer actually acted as an average bank day with real customers: each time, we generated a series of 150 customers, each who had his own probabilistically-determined time elapsed since last customer's arrival and his own probabilistically-determined service time.	Because of time constraints, we ran the simulation of 150 customers each only five times and therefore ran the experiment for only 150 trials x 5 runs = 750 customers. Ideally, we would have analyzed thousands of customers hundreds of times, but although the computer could have accomplished this quickly, it would have taken far too long for us to present the data in a meaningful chart of patterns.
Our program is not specific to this problem: we could easily modify this model simply by editing constant declarations (for example, even as the manager trained his employees to process transactions more quickly, we could change the probabilities of each service time to reflect the upgraded branch platform)	Numbers generated are only pseudorandom: they come from a complex hashing algorithm based upon a timestamp, but they are deterministic nonetheless

Conclusion

Sensitivity Testing

Our model is not very sensitive to outside changes: in fact, the reason our recommendations to management were so severe was that (especially considering that the management desired absolute zero levels for excess wait times or queue lengths) even incremental manual changes did not affect the results significantly. The only independent variables in this experiment were the probabilities of each customer entering at a given time and using a certain service length, and although in real life fluctuations would occur, they are not be shown to strongly alter results.

Extra questions:

Given extra time, we would, of course, ideally have tested out the model's predictions in real life but, more importantly, we would have actually discussed a real bank's operations with actual management and figured out where the most time is lost. For instance, if deposits/withdrawals (vault runs) take the most time, ATMs could solve that fairly easily; if account maintenance takes the most time, perhaps worker training and an updated consumer website might be in order. We would also would have taken into account a model of the week had the influx of customers per day not been assumed to be constant.

Non-technical Letter to the Manager

11/11/2013

ATTN: Manager

XYZ Bank

4771 Campus Drive

Irvine, CA, 92608

Dear Manager,

After considering your request and modeling the current demands upon your branch, we have determined that your current service platform will unfortunately not allow you to meet your goals:

*Wait time less than two minutes for any customer: we have determined that, on an average day, between 110 and 130 of your customers will have to wait in line for at least two minutes, assuming your current service platform. The modeled average wait time for any given customer is about 7 minutes.

*Queue no longer than two persons at any time: we have determined that, on an average operating day of 501 minutes, your line will be at least three people long for about 144 minutes from the time the first customer enters to the time the last customer leaves, assuming your current service platform. The modeled average line length at any given minute of the day is about 3 persons.

The limitations imposed upon your branch are, unfortunately, inescapable results of the fact that your business is healthy and your customers arrive frequently and for extended periods. However, your growth will be limited until you are able to satisfy your customers to your establishment's high standards.

With this in mind, we have developed two ways to optimize your customer service and increase the speed at which your customers are processed. Thus, we recommend worker training modules to speed your worker's transaction capabilities. The first way optimizes that 90% of all customers wait in line for less than two minutes. If you were able to eliminate four-minute service times (currently 15% of your business) and three-minute service times (currently 40% of your business),

then increase two-minute service times to 65% and one-minute service times to 35% through worker efficiency training, you would seriously improve wait time. However, the second way optimizes that all 150 customers wait for an average of less than two minutes. This model is incredibly more efficient. If you were able to eliminate four-minute service times (currently 15% of your business) and improve two-minute service times by 5% and one minute service times by 10%, the process would achieve the goal, albeit the number of customers who wait in line or two minutes or more and the line length would increase five fold. Furthermore, both optimization models also reduce the line length to fit your expectations.

One opportunity to achieve this might be to parallelize tasks: many banking customers walk into branches seeking to service accounts and to withdraw or deposit money; however, account servicing requires a platform banker, while balance modification requires a teller. In a traditional branch, the customer would visit the teller first and then the banker, but you instead would be able to save time by allowing the customer to fill out a balance slip when he entered (only if there was a line), and then routing any customer with an account service request to the banker first. The banker would submit the customer's balance slip to the tellers if there was a line or, if there was no line, simply perform the maintenance request and then send the customer to the teller as usual. In this way, customers who only wished to deposit or withdraw would quickly be able to do so if there were no line, but those with maintenance requests would be able to avoid longer lines since their money would be deposited or withdrawn while they themselves discussed their accounts with a banker.

Realistically, however, these changes will require ambitious efforts that will tax you and might strain your workers. Instead, you might consider hiring one more teller to process quick requests and one more banker to service accounts. Ultimately, you might simply wish to lower your standards: training workers, parallelizing tasks, and hiring more staff will allow your branch to proceed so that only a few customers each day would have to wait for about three minutes in lines of three or less while most would wait less than two minutes in lines of two or less. At the same time, your workers would not feel overburdened and would be able to continue providing your customers with cheerful, useful service.

We wish you and your business luck.

Sincerely,

Team Limit DNE

Bibliography

1. Federal Reserve Board. "Top 50 Holding Companies Summary Page." Top 50 Holding Companies Summary Page. N.p., 30 Sept. 2013. Web. 11 Nov. 2013.

2. Horstmann, Cay S. Big Java. Hoboken, NJ: John Wiley, 2005. Print.

Appendices

Appendix A. Program Code

The following is the program we used to simulate 150 bank customers over a single bank day, with wait times and service times randomly generated based on the probability provided in the problem.

// indicates annotations not in the program.

Main Method Program

```
//Bank class calculates individual time between arrivals and service time for each individual
public class Bank
{
   public static void main (String[] args)
   {
      //Creates a time stamp for the system
      int timeStamp = 0;

      //Creates a mathematical constant of which the bank estimates it serves (150 people)
      final int MAXNUMBEROFPEOPLE = 150;

      //Creates a mathematical constant of approximately how many minutes (nearest int) the
bank is open in a six day week (weighted average)
      //Computed in previous calculations
      final int AVGNUMMINUTES = 501;

      //Creates integers for the wait time for each customer, the time between arrivals, and the
```

```
service time
        int[] waitTime = new int[MAXNUMBEROFPEOPLE];
        int timeBetweenArrivals;
        int serviceTime;

        //Creates a true/false boolean for each person to determine if they are waiting in line or
not (true for yes, false for no)
        boolean[] inLine = new boolean[MAXNUMBEROFPEOPLE];

        //Creates one counter to determine how many customers have to wait more than 2 minutes
for service
        //Creates one counter to determine how many minutes the line exceeds two people
        int waitOverflow = 0;
        int lineOverflow = 0;

        //Creates an array that stores each person's arrival time
        int [] arrivalTimes = new int[MAXNUMBEROFPEOPLE];

        //Creates an array that stores when each peron will leave the bank
        int [] afterService = new int[MAXNUMBEROFPEOPLE];

        //For each minute in a day, this array determines how many people are in the line every
minute
        int[] people = new int[AVGNUMMINUTES];

        //Sum to determine average length of the queue
        double sumLine = 0;
        //Sum to determine average wait time
        double sumWait = 0;
        //Average Line
        double averageLine = 0;
        //Average waitTime
        double averageWait = 0;

        //For first customer, there is no delay time, therefore there must be a separate object to
construct
        //Create a new customer (first in line)
```

```
        Customer firstInLine = new Customer();
        //Generate random time between arrival and service time for the first customer
        timeBetweenArrivals = firstInLine.randomDelayTime();
        serviceTime = firstInLine.randomServiceTime();
        //Input these random times into the constructor
        firstInLine = new Customer(timeBetweenArrivals, serviceTime);

        //Store arrival time and leaving time
        arrivalTimes[0] = timeBetweenArrivals;
        afterService[0] = serviceTime + timeBetweenArrivals;

        //Print out arrival time (in minutes from opening), transaction times, and when each
customer leaves
        //System.out.println("Arrival Time of Customer 1 (in minutes from opening): " +
arrivalTimes[0]);
        //System.out.println("Customer 1 had a transaction time of: " + serviceTime);
        //System.out.println("Finished service at: " + afterService[0]);

        //Perform same actions as above except that it calculates the time between arrivals and
determines if there is a wait time
        for (int i = 2; i <= MAXNUMBEROFPEOPLE; i++)
        {
            Customer nextInLine = new Customer();
            timeBetweenArrivals = nextInLine.randomDelayTime();
            serviceTime = nextInLine.randomServiceTime();
            //Prints out the time between arrivals
            System.out.println("\nThe time between arrivals of previous and current customer is: "
+ timeBetweenArrivals);
            nextInLine = new Customer(timeBetweenArrivals, serviceTime);

            //Sets arrival time of current customer to be the sum of time between arrivals and the
previous customer's arrival time
            arrivalTimes[i-1] = timeBetweenArrivals + arrivalTimes[i-2];

            //Prints out the arrival time
            System.out.println("Arrival Time of Customer " + i + " (in minutes from opening) is: "
+ arrivalTimes[i-1]);
```

```
            //If the arrival time of the current customer and the previous customer's leaving time
is negative
            //In other words, if the current customer arrives before the previous customer leaves
            if ((arrivalTimes[i-1] - afterService[i-2]) < 0)
            {
               //Calculates wait time of the current customer
               waitTime[i-1] = afterService[i-2] - arrivalTimes[i-1];
               //If the wait time is not less than 2 minutes (as stated in the problem), the counter
for the overflow cycles once
               if (waitTime[i-1] >= 2)
               {
                  waitOverflow += 1;
               }
               //The current customer's status for a period of time is that he/she must wait in line
               inLine[i-1] = true;
               sumWait += waitTime[i-1];
               averageWait = sumWait / MAXNUMBEROFPEOPLE;
               //Prints out wait time of customer, service time, and leaving time
               afterService[i-1] = afterService[i-2] + serviceTime;
               System.out.println("Wait Time of Customer " + i + " is: " + waitTime[i-1]);
               System.out.println("Customer " + i + " had a transaction time of: " + serviceTime);
               System.out.println("Finished service at: " + afterService[i-1]);

            }
            else
            {
               //If the customer doesn't have to wait, they go immediately after they arrive
               afterService[i-1] = arrivalTimes[i-1] + serviceTime;
               System.out.println("Customer " + i + " had a transaction time of: " + serviceTime);
               System.out.println("Finished service at: " + afterService[i-1]);
            }

      }
```

```
        //Runs the loop for every customer
        for (int j = 2; j <= MAXNUMBEROFPEOPLE; j++)
        {
            //If the customer is in line at any point in time, the statement will compute
            if (inLine[j-1])
            {
                //For the time the person starts waiting in line (inclusive), to the point where they are
processed (exclusive)
                for (int k = arrivalTimes[j-1]; k < (arrivalTimes[j-1] + waitTime[j-1]); k++)
                {
                    //For every minute a person is waiting in line, the array for the amount of people
per minute goes up
                    //Array (as all arrays in this program) are offset by one because indexes of arrays
begin at 0
                    people [k-1] += 1;
                }
            }
        }

        //Runs the clock for one day
        for (timeStamp = 1; timeStamp <= AVGNUMMINUTES; timeStamp++)
        {
            //Prints out each minute
            System.out.println("Time Stamp: " + timeStamp);

            sumLine += people[timeStamp-1];

            averageLine = sumLine / (afterService[149] - arrivalTimes[0]);

            //Prints out the average length of the line at each minute the customers are in the bank
            System.out.println("Length of Line is: " + people[timeStamp-1]);

            //If the line exceeds 2 people, the counter increments by 1
            if (people[timeStamp-1] > 2)
            {
                lineOverflow += 1;
```

```
            }

        }
        //Prints out the number of customers that must wait 2 minutes or more
        System.out.println("The wait time is 2 minutes or more for " + waitOverflow + "
customers");
        //Prints out the number of minutes that the line exceeds 2 people
        System.out.println("The line exceeds 2 people for " + lineOverflow + " minutes");
        //Prints out the average wait time
        System.out.println("The average wait time is: " + averageWait);
        //Prints out the average length of the line during the time people are in the building
        System.out.println("The average length of the line is: " + averageLine);



    }
}
```

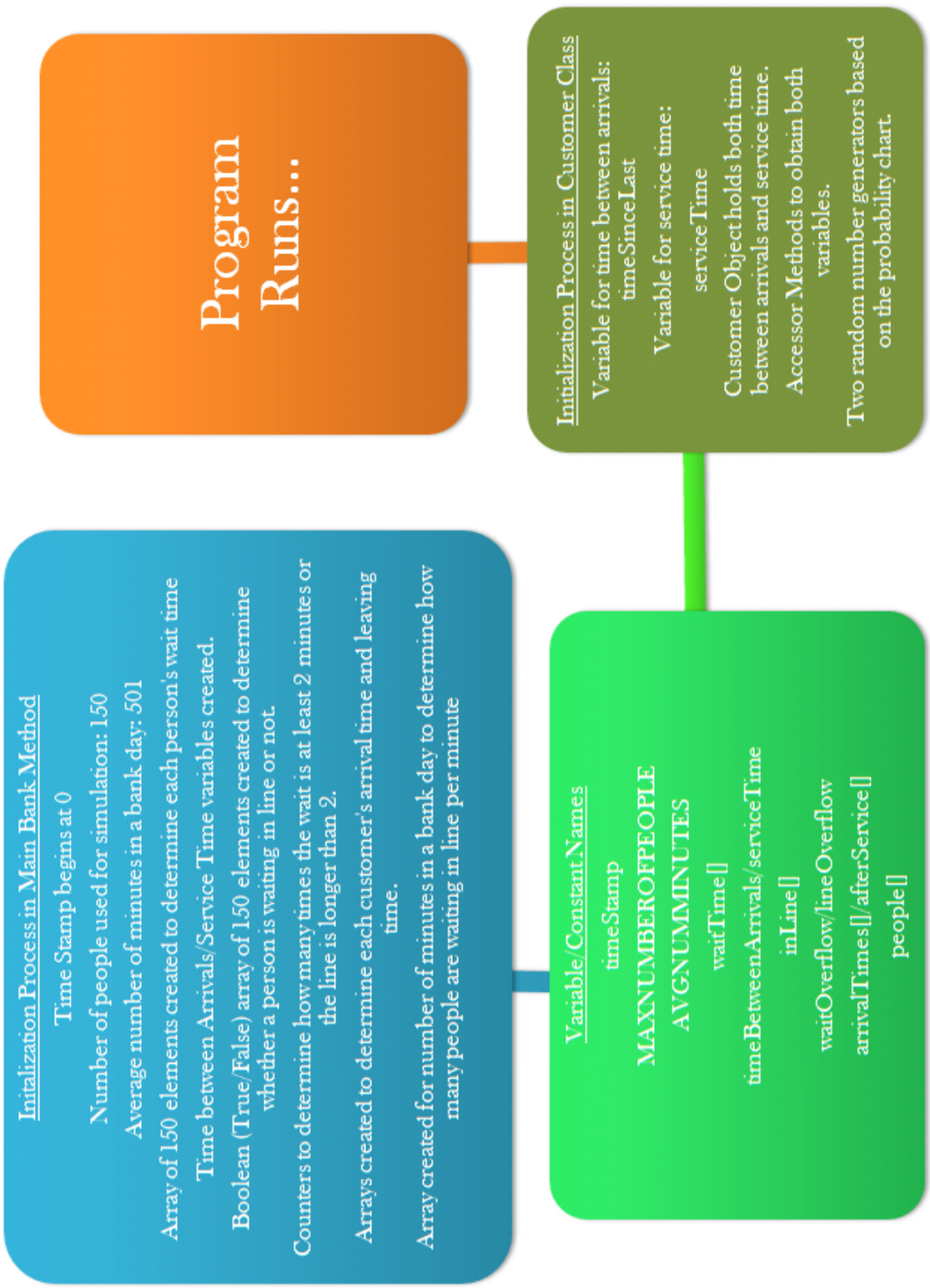
Method Class

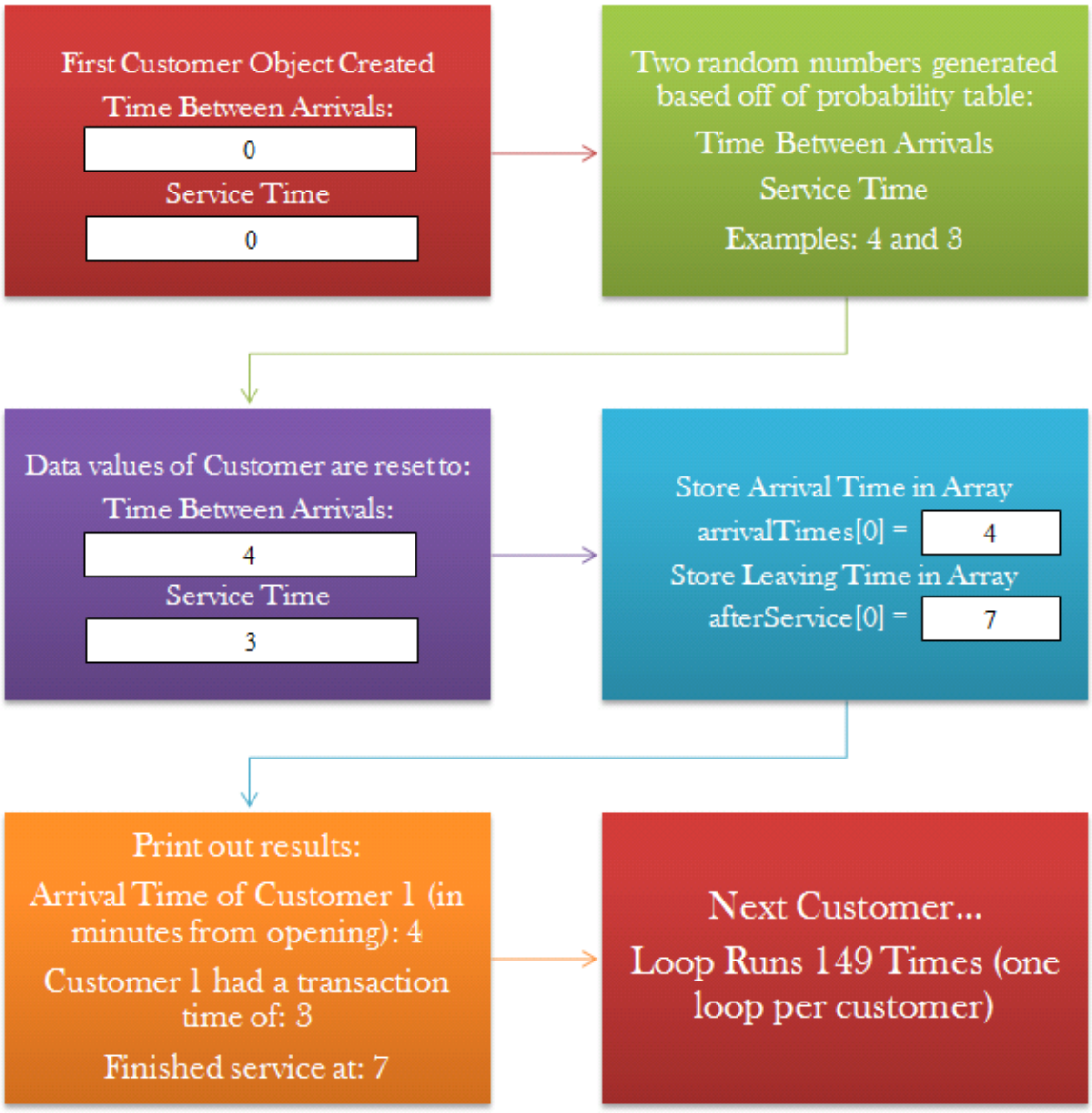
```
//This class allows storage of each Customer's unique time between arrival and service time, the
ability to keep track of each customer
//, and the calculation of arrival and service times
public class Customer
{
    //Instantiate instance fields, one for the arrival time, the other for the service time
    private int timeSinceLast;
    private int serviceTime;

    //Default Constructor (initalizes each instance field to zero)
    public Customer()
    {
        timeSinceLast = 0;
        serviceTime = 0;
    }

    //Constructor that initalizes each instance field according to a random arrival and service
time
    public Customer(int timeBetween, int serviceLength)
```

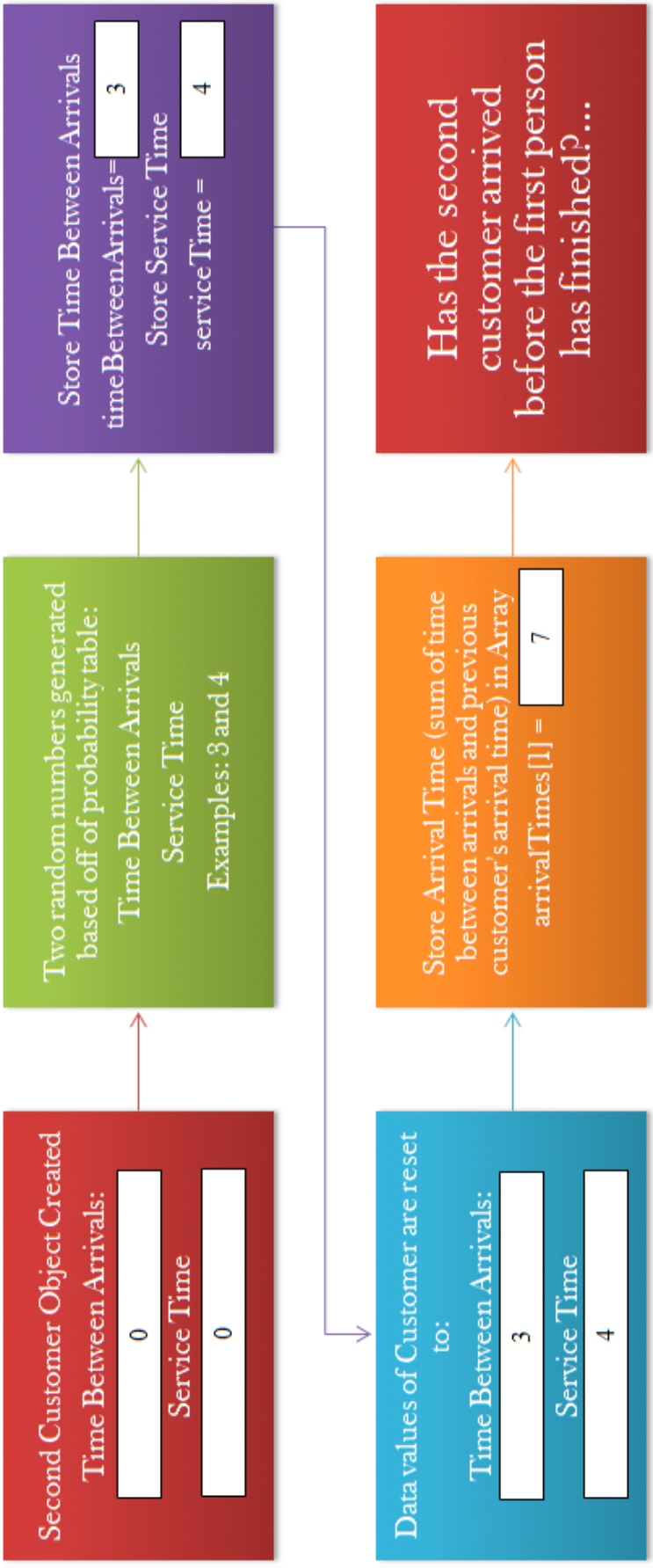
```
{
   timeSinceLast = timeBetween;
   serviceTime = serviceLength;
}

//Access method to obtain arrival time
public int getArrivalTime()
{
   return timeSinceLast;
}

//Accessor method to obtain service time
public int getServiceTime()
{
   return serviceTime;
}

//Returns a random time between arrivals based on the data chart provided by Problem B
public int randomDelayTime()
{
   int arrivalTime;
   //Generates a random number
   double arrival = Math.random();

   //Returns the time (probability based on the random number)
   if (arrival <= 0.10)
      return arrivalTime = 0;
   else if (arrival <= 0.25)
      return arrivalTime = 1;
   else if (arrival <= 0.35)
      return arrivalTime = 2;
   else if (arrival <= 0.7)
      return arrivalTime = 3;
   else if (arrival <= 0.95)
      return arrivalTime = 4;
   else
```

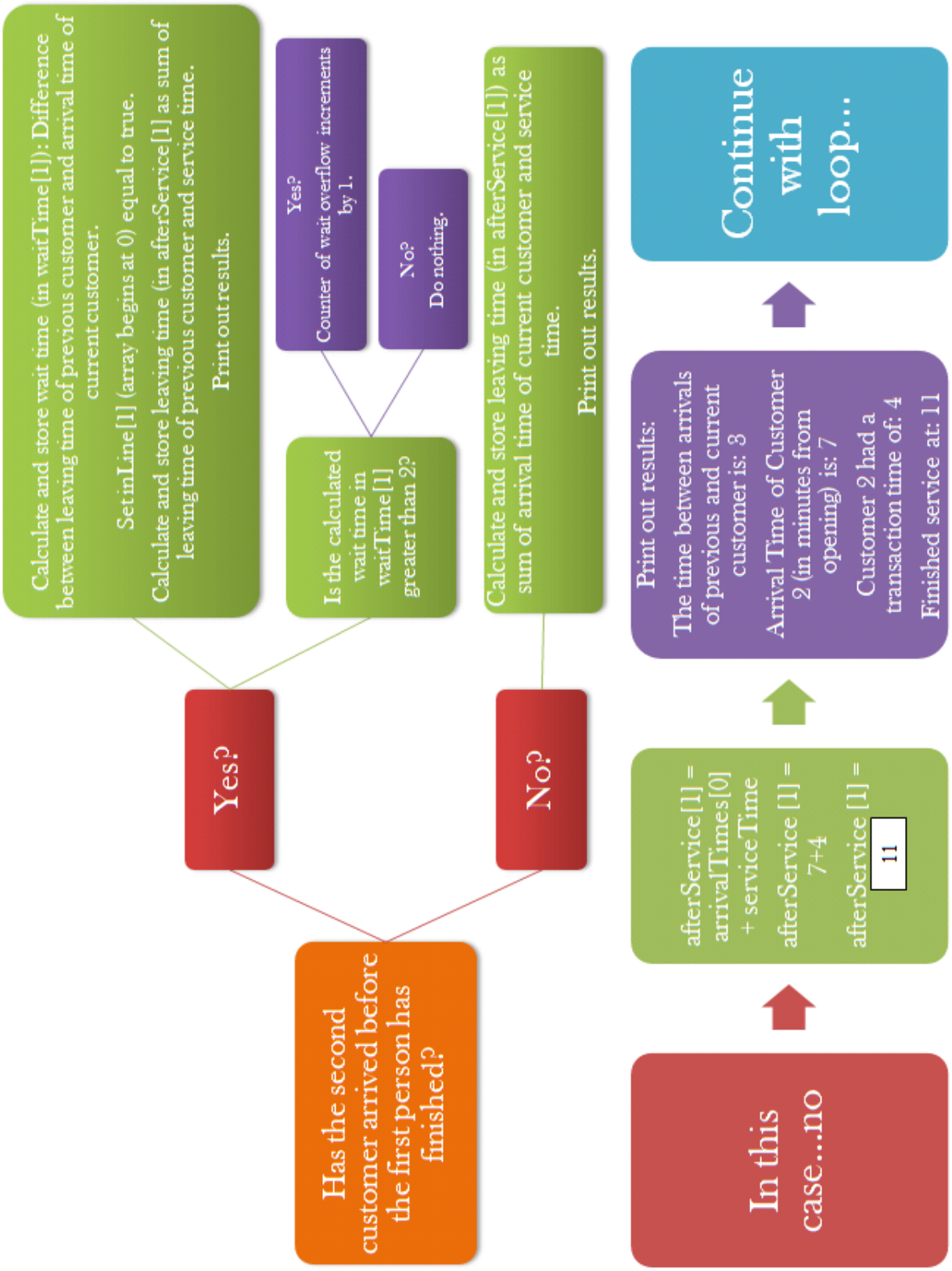
```
            return arrivalTime = 5;
    }

    //Returns a random service time based on the data chart provided by Problem B
    public int randomServiceTime()
    {
        int serviceTime;
        //Generates a random number
        double service = Math.random();

        //Returns the time (probability based on the random number)
        if (service <= 0.35)
            return serviceTime = 1;
        else if (service <= 0.60)
            return serviceTime = 2;
        else if (service <= 1)
            return serviceTime = 3;
        else
            return serviceTime = 4;
    }

}
```

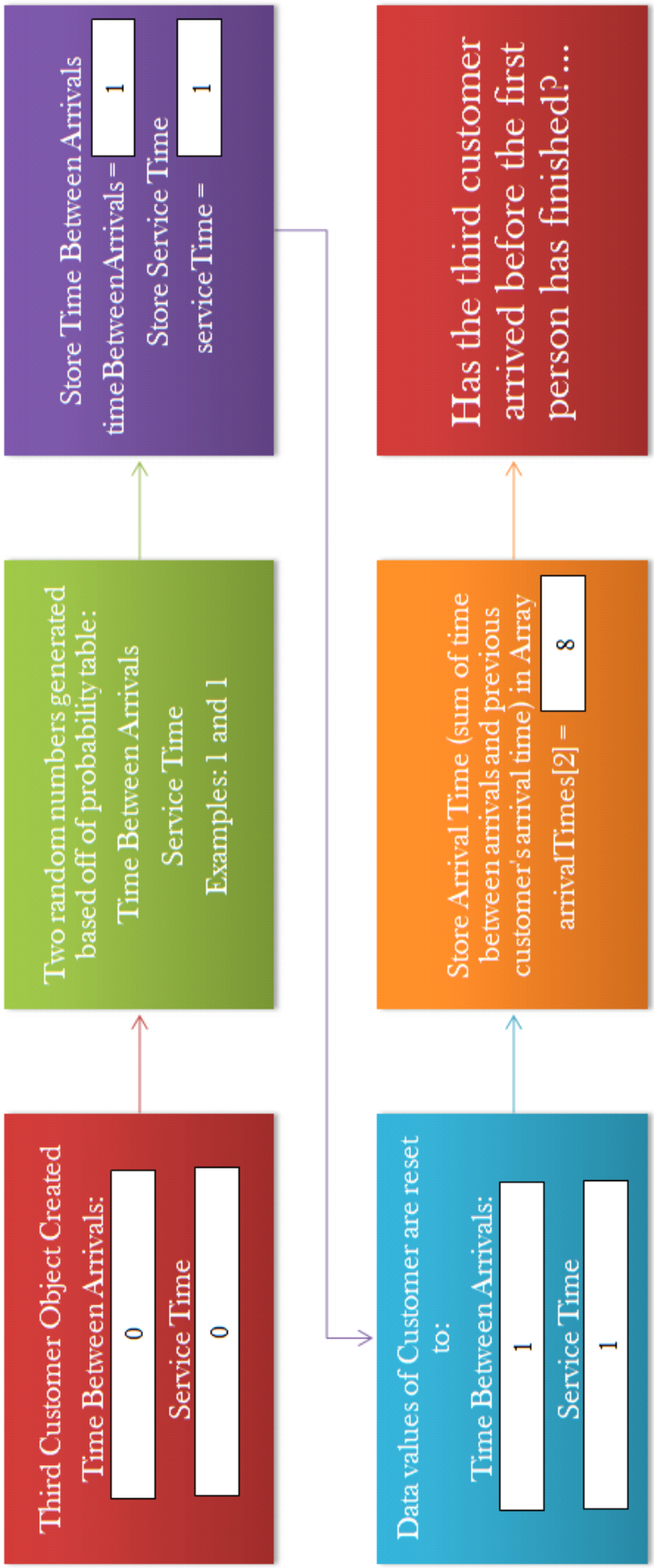
Appendix B. Program Flow Charts

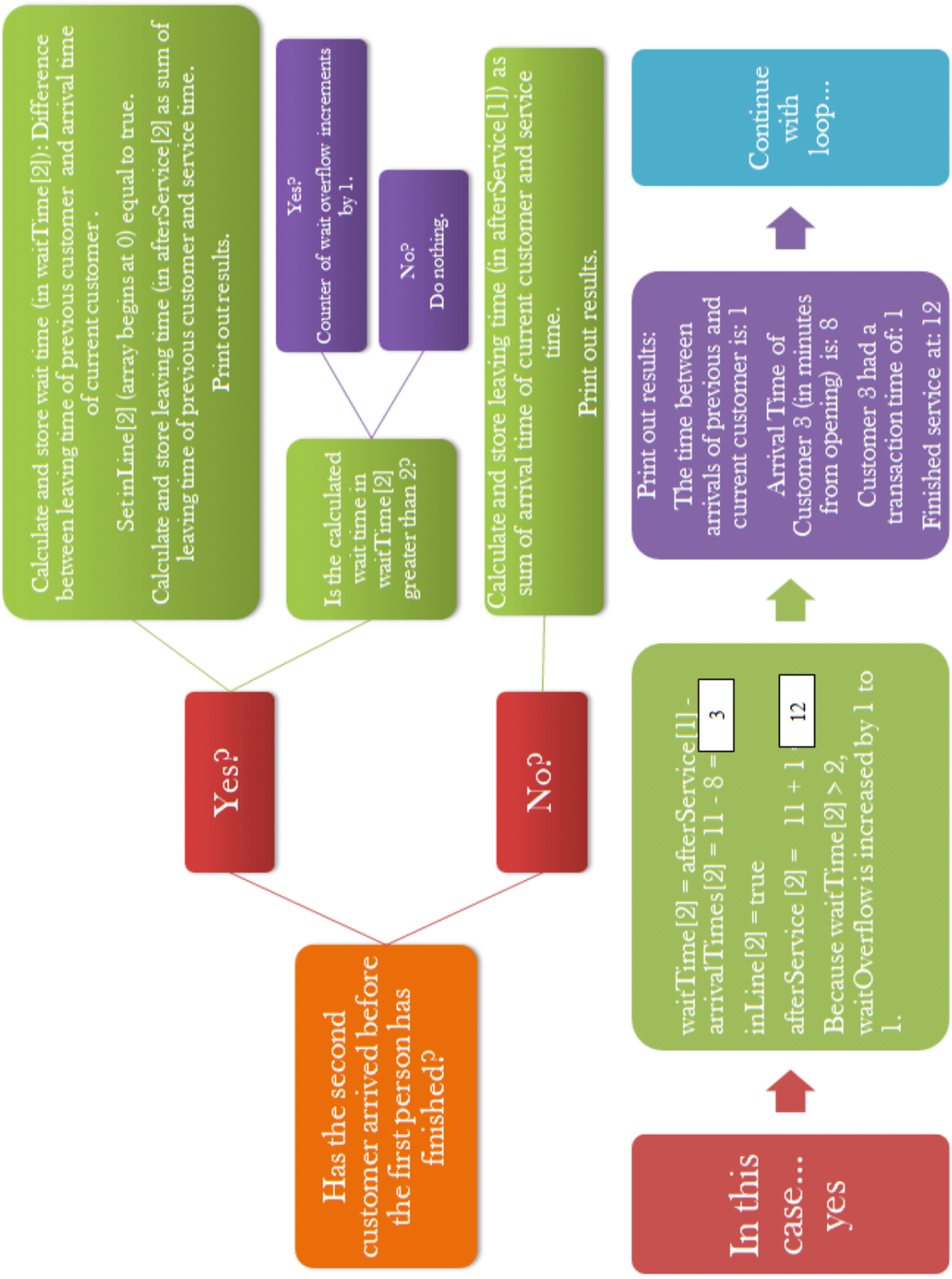


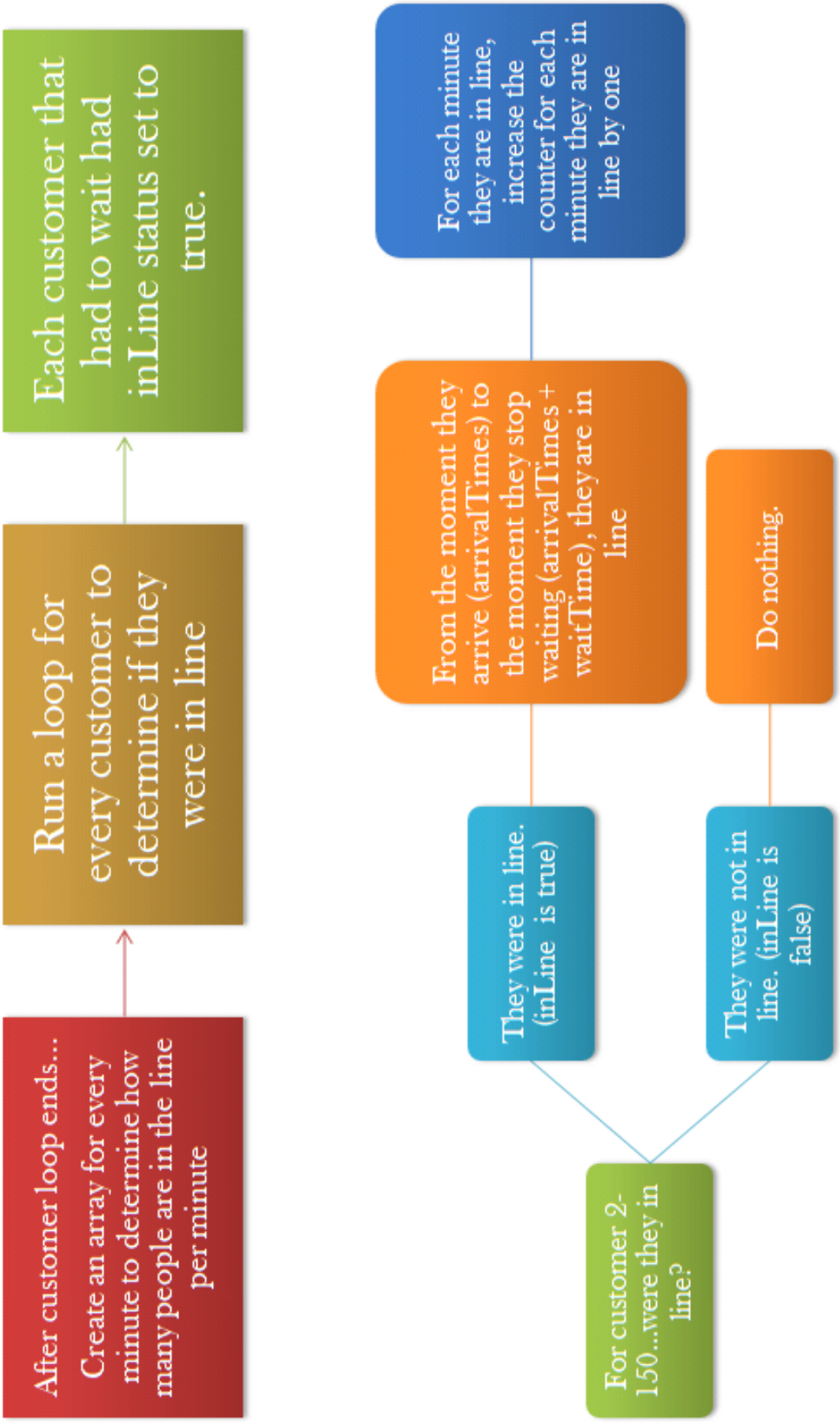

First Customer Object Created
Time Between Arrivals:
0
Service Time
0
Two random numbers generated based off of probability table:
Time Between Arrivals
Service Time
Examples: 4 and 3
Data values of Customer are reset to:
Time Between Arrivals:
4
Service Time
3
Store Arrival Time in Array
arrivalTimes[0] = 4
Store Leaving Time in Array
afterService[0] = 7
Print out results:
Arrival Time of Customer 1 (in minutes from opening): 4
Customer 1 had a transaction time of: 3
Finished service at: 7
Next Customer...
Loop Runs 149 Times (one loop per customer)

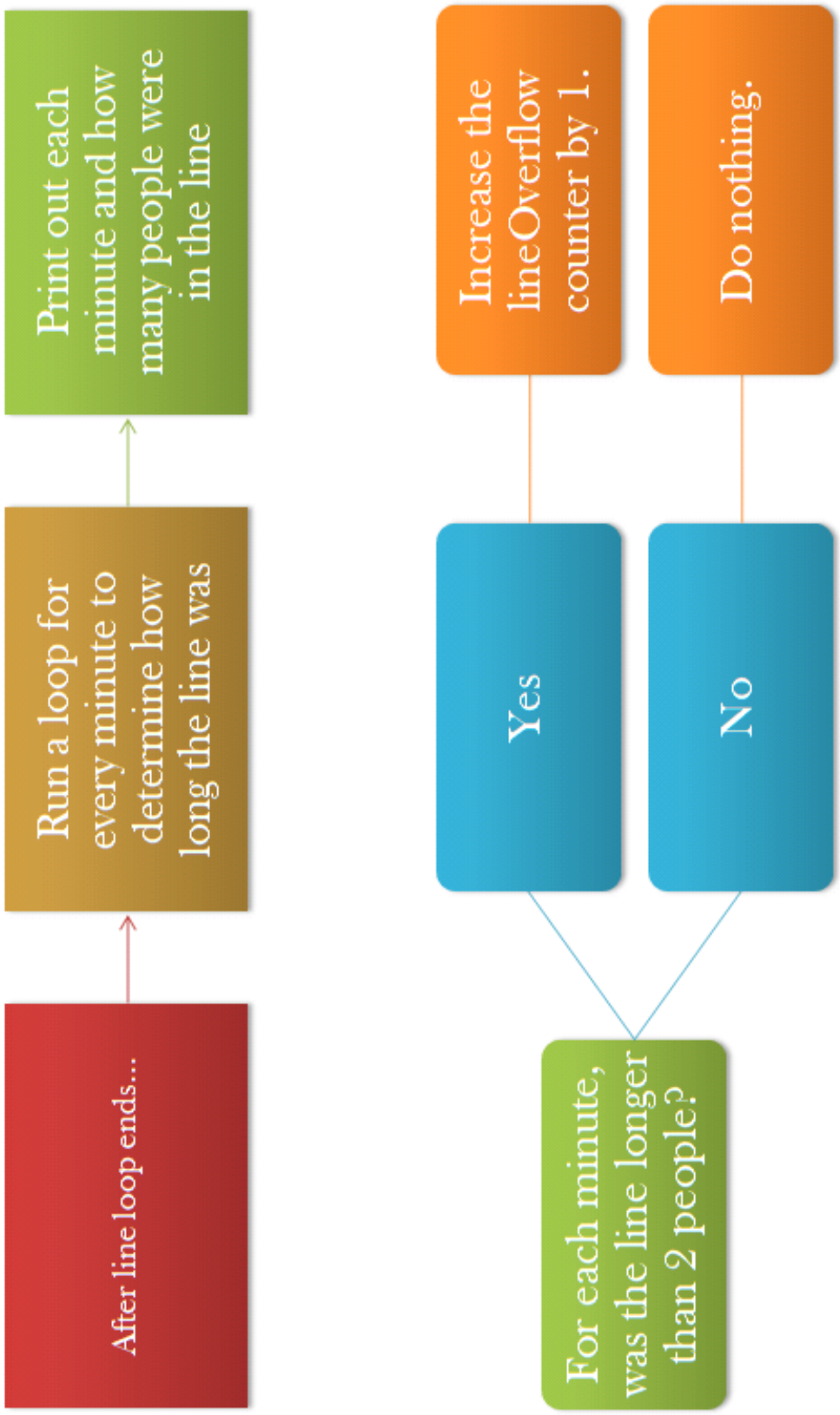

Second Customer Object Created
Time Between Arrivals:
0
Service Time
0
Two random numbers generated based off of probability table:
Time Between Arrivals
Service Time
Examples: 3 and 4
Store Time Between Arrivals
timeBetweenArrivals=
3
Store Service Time
serviceTime =
4
Data values of Customer are reset to:
Time Between Arrivals:
3
Service Time
4
Store Arrival Time (sum of time between arrivals and previous customer's arrival time) in Array
arrivalTimes[1] =
7
Has the second customer arrived before the first person has finished?...

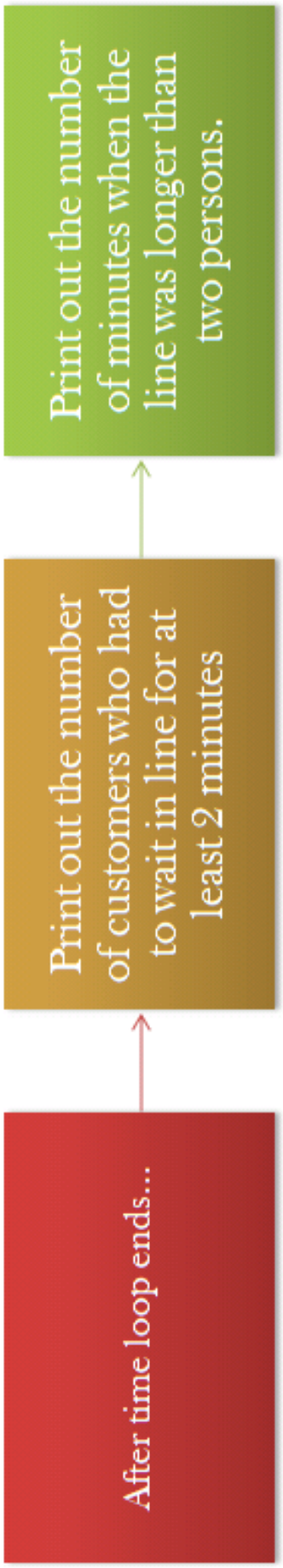

Has the second customer arrived before the first person has finished?
Yes?
Calculate and store wait time (in waitTime[1]): Difference between leaving time of previous customer and arrival time of current customer.
SetinLine[1] (array begins at 0) equal to true.
Calculate and store leaving time (in afterService[1] as sum of leaving time of previous customer and service time.
Print out results.
Is the calculated wait time in waitTime[1] greater than 2?
Yes?
Counter of wait overflow increments by 1.
No?
Do nothing.
No?
Calculate and store leaving time (in afterService[1]) as sum of arrival time of current customer and service time.
Print out results.
In this case...no
afterService[1] = arrivalTimes[0] + service Time
afterService [1] = 7+4
afterService [1] = 11
Print out results:
The time between arrivals of previous and current customer is: 3
Arrival Time of Customer 2 (in minutes from opening) is: 7
Customer 2 had a transaction time of: 4
Finished service at: 11
Continue with loop...


Third Customer Object Created
Time Between Arrivals:
0
Service Time
0
Two random numbers generated based off of probability table:
Time Between Arrivals
Service Time
Examples: 1 and 1
Store Time Between Arrivals
timeBetweenArrivals = 1
Store Service Time
serviceTime = 1
Data values of Customer are reset to:
Time Between Arrivals:
1
Service Time
1
Store Arrival Time (sum of time between arrivals and previous customer's arrival time) in Array
arrivalTimes[2] = 8
Has the third customer arrived before the first person has finished?...


Has the second customer arrived before the first person has finished?
Yes?
Calculate and store wait time (in waitTime[2]): Difference between leaving time of previous customer and arrival time of current customer.
Set inLine[2] (array begins at 0) equal to true.
Calculate and store leaving time (in afterService[2] as sum of leaving time of previous customer and service time.
Print out results.
Is the calculated wait time in waitTime[2] greater than 2?
Yes?
Counter of wait overflow increments by 1.
No?
Do nothing.
No?
Calculate and store leaving time (in afterService[1]) as sum of arrival time of current customer and service time.
Print out results.
In this case... yes
waitTime[2] = afterService[1] - arrivalTimes[2] = 11 - 8 = 3
inLine[2] = true
afterService [2] = 11 + 1 = 12
Because waitTime[2] > 2, waitOverflow is increased by 1 to 1.
Print out results:
The time between arrivals of previous and current customer is: 1
Arrival Time of Customer 3 (in minutes from opening) is: 8
Customer 3 had a transaction time of: 1
Finished service at: 12
Continue with loop...


After customer loop ends… Create an array for every minute to determine how many people are in the line per minute
Run a loop for every customer to determine if they were in line
Each customer that had to wait had inLine status set to true.
For customer 2-150…were they in line?
They were in line. (inLine is true)
They were not in line. (inLine is false)
From the moment they arrive (arrivalTimes) to the moment they stop waiting (arrivalTimes + waitTime), they are in line
Do nothing.
For each minute they are in line, increase the counter for each minute they are in line by one


After line loop ends…
Run a loop for every minute to determine how long the line was
Print out each minute and how many people were in the line
For each minute, was the line longer than 2 people?
Yes
No
Increase the lineOverflow counter by 1.
Do nothing.


After time loop ends…
Print out the number of customers who had to wait in line for at least 2 minutes
Print out the number of minutes when the line was longer than two persons.

Appendix C. Average Bank Hours

Computation for Average Bank Hours:

JPMorgan and Chase: $2,439,494,000- 37.826%

M-F: 9-6, Sat: 9-4

Average Hours per Day: [5(9) + 7]/6 = 8.667 hours

Bank of America $2,125,686,000- 32.961%

M-F: 9-6, Sat: 10-2

Average Hours per Day: [5(9) + 4]/6 = 8.167 hours

Citigroup Inc. $1,883,988,000- 29.213%

M-Th: 9-6, F: 9-5, Sat: 9-2

Average Hours per Day: [4(9) + 8 + 5]/6 = 8.167 hours

Weighted Average:

M-Th: .37826(9) + .32961(9) + .29213(9) = 9

F: .37826(9) + .32961(9) + .29213(8) = 8.709

Sat: .37826(7) + .32961(4) + .29213(5) = 5.427

.37826(8.667) + .32961(8.167) + .29213(8.167) = 8.356 hours ≈ 501 minutes

Appendix D: Third Program

This is a program we used to check our work. Results are shown below.

Trial	Customers who waited for 2+ minutes	Minutes that line had 2+ people
1	80	54
2	93	67
3	96	143
4	105	110
5	109	116
6	83	57
7	70	10
8	137	226
9	73	28
10	99	76
11	132	161
12	119	218
13	91	81
14	107	78
15	85	87
16	87	92
17	125	122
18	111	200
19	96	58
20	64	4