For office use only		For office use only	
T1	_____	F1	_____
T2	_____	F2	_____
T3	_____	F3	_____
T4	_____	F4	_____

2014

17th Annual High School Mathematical Contest in Modeling (HiMCM)

Summary Sheet

(Please attach a copy of this page to your Solution Paper.)

Team Control Number: **4823**

Problem Chosen: **A**

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 today's global economy, efficiency counts. In economic systems, where large volumes of information are exchanged at lightning speed, the smallest inefficiencies can have drastic consequences for the productivity and success of the system as a whole. In logistics, corporations engineer their products' design, production and distribution with these small inefficiencies in mind, as an infinitesimal additional cost for one unit can blossom into the difference between profit and bankruptcy when the product is manufactured in large quantities. Similarly, on Wall Street, stockbrokers harness the power of supercomputers to make trades as quickly as possible and obtain an edge. Just as cost effects industry, a minimal temporal disadvantage for a trader can create a devastating financial loss. Now, more than ever, systems of this kind must be engineered with considerations in mind to eliminate the crippling effects of inefficiency.

Our group was tasked with optimizing a system of just this sort: commuter rail. Each day, corporations rely on commuter trains to bring their employees to the workplace, making commuter rail a crucial form of transportation. Specifically, we focused on the exit of the train station as an area for improvement, for this is traditionally a bottleneck for the speed at which commuters can proceed to work.

To model this situation, our group developed a computer simulation of the train station in Java. We created objects for people and staircases and made a simulator that based on parameters we entered, told us the average time that it takes one person to make if from their train seat, to street level. Using this data, we were able to isolate variables such as the number of train cars, car length, number of steps, staircase positioning and staircase carrying capacity and see their effects on average time. Noteworthy findings from our model include the fact that "seat to street" time is nearly a $1/\sqrt{K}$ relationship with k, the number of columns of individuals on a given staircase as well as linear with a number of the aforementioned variables. We manipulated the

positioning of a given number of staircases and found that if there are staircases near at the ends of a train, people may exit the station the fastest. Using assumptions about the train station and train itself we aligned our model with industry standards so we could finalize and backup our model with real data. The M8 train manufactured by Kawasaki nearly perfectly aligned with the problem statement, and Grand Central Station in New York City gave us a similar transportation hub to observe and compare. We believe that our model can be used by industry professionals to redesigning their train stations to make commuter's commutes faster and more enjoyable, improving the profitability and efficiency of industry in their city as well.

To the Director of Transportation,

Thank you for employing us to create a model to assist you in your endeavor to make exiting the central station more efficient. We designed a computer simulation that calculates the average train seat to street time for all of the commuters on a given train. It uses the various information about your trains and train station, including the length of your train cars, the number of train cars per train, the length of your platform, the number of staircases, the number of stairs per staircase, the person capacity (side by side) of one step, and the position of these staircases in relation to the platform. The simulation used two objects, 'person,' and 'staircase,' and a main simulator that created people based on the size of the train and staircases based on the inputs for positioning and quantity. Using this simulation, we were able to produce realistic data that demonstrated how these aforementioned factors affected the time from seat to street, and then create an accurate all encompassing model.

There are multiple important takeaways from this model. The first is that, as would be expected, the average time increases with the size of the train car, the number of train cars, and the number of steps on each staircase. We understand that it is not reasonable to expect that you be able to change these factors, so we looked at other, more malleable contributors. From our simulation and model, we figured out that by increasing the capacity of a single step for commuters travelling side by side up a staircase, you can significantly reduce the average travelling time, such as by widening the staircase. Also, increasing the number of staircases is effective in shortening travel time, but only to a point. Once the number of staircases begins to exceed the number of train exits, there is no decrease in travelling time. In order to optimize this

travelling time, we recommend that you increase that step capacity, and add staircases until they meet the number of train exits.

As purchasing a car becomes less and less appealing with soaring gas prices and global warming folks will become increasingly reliant on public transportation. If the number of commuters who make use of your train increases in the near future, and you find yourself having to make renovations on a limited budget, our model can be used to compute what the best way to do that is. We know how important customer satisfaction is to you, and we think our model can be instrumental in keeping that level of satisfaction high. We hope that you find this model helpful in creating a more efficient train station.

Best regards,

Team 4823

FROM SEAT TO STREET: OPTIMIZING TRAIN STATIONS

Table of Contents

PROBLEM STATEMENT

Trains arrive on one of two tracks at a central train station, with a middle platform between the two tracks. Each train car has a center aisle and there are two seats on one side and three seats on the other. To exit at a station, passengers exit the car and proceed to a staircase, which can accommodate two columns of commuters, that takes them to street level to exit the station. We are tasked to build a mathematical model to estimate the amount of time for a passenger to reach the street level, above the station, starting from the time to train pulls into the station. This is done with $\boldsymbol{n}$ cars per train, each with length $\boldsymbol{d}$. Platform length is represented by $\boldsymbol{p}$, and the number of stairs is $\boldsymbol{q}$. Using this model, we are to minimize the time traveled from a seat on a train car to the street level, provided the following 5 requirements:

1) All passengers on one fully occupied train are to exit.
2) All passengers on two fully occupied trains are to exit.
3) If we could redesign the location of the stairways along the platform, where would we place them?
4) How does time to exit to street level vary with s the number of stairways built?
5) How does the time vary as stairways can accommodate k (where $k > 1$) people?

ASSUMPTIONS AND RATIONALE

Assumption 1: Every passenger on the train has a seat. The problem states that all train cars are full, which we assumed to mean that every seat on the train is occupied and nobody is standing up, which allows us to use the number of seats on the train to calculate the number of people on the train. While standing may be common on trains like the Chicago 'L' or the New York Subway, on commuter trains it is infrequent.

Assumption 2: When walking up stairs, commuters maintain a distance of one step between each other. We know that in the instance of a crowded commute, people will be packed together tightly. However, it is not realistic to assume that commuters would be comfortable standing as close together as two steps are, so the next best way to then pack people onto the stairs is to assume they are one step apart. Our group tested climbing stairs at this distance, and determined that this was indeed a comfortable interval of separation that would certainly be suitable for commuters.

Assumption 3: Each train car has two exit aisles on either end of the car and has two doors total, one at either end of the car. These doors have a width of one meter, and there are no seats in the same row as the door. One meter is a reasonable space for one person to exit at a time. This assumption is essential when determining how the length of a train car determines the number of seats and therefore number of people that fit on a single train car. The difference in distance

between walking from the center aisle to the door on one side versus the other is the difference in a step and negligible, so for simplicity's sake we will assume that the doors are on the side of the train with two columns of seats (See Figure 5 in Appendix C).

Assumption 4: The distance from the back of one seat to the next one in front of it is one meter. The American Physical Therapy Association Commuter Rail Executive Committee[1] published the current standards for rail seating in 2010, finding that seating on trains should be designed to accommodate the 95th percentile sized male adequately. Scott Openshaw of Oregon State,[2] published ergonomics and design guidelines, finding that the 95th percentile male has a hip to knee length of 26.3 inches. Current standards suggest that 1.5 times the hip to knee length will suit this sized individual comfortably. This means the distance from one seat back to the next should be 39.45 inches, or 1.00203 meters (See Figure 5 in Appendix C).

Assumption 5: We will use the value that the National Institute of Technology[3] reports as the average time to walk the horizontal distance of a stair, when walking up stairs: .442 meters per second.

Assumption 6: Walking speeds on the platform determined for each person will be randomly selected from a normal distribution with mean 1.3005 and standard deviation f .1705. The American Public Transportation Association[4] reports that the average weekday commuter is 37 years old, which falls into the range that the Journal of Rehabilitation Research[5] finds has an average walking speed of 1.3005 meters per second (not on stairs) and a standard deviation of .1705 meters per second. The speed they walk at on the platform will be normally distributed relative to the average walking speed we found. Also, it is more reasonable to have varying walking speeds on the platform and not in the train or on the staircases because the platforms have much more space so that commuters can pass others walking at slower speeds, as opposed to having all commuters walk at the same speed in the train and on the staircase, where there is not room for passing.

Assumption 7: We assume our passengers are courteous and follow the exit order given in Figure 4 of Appendix B. When filing off of the train, commuters wait until the person before them in the exit order leaves, wait an additional two seconds to account for separation in the aisle, and then leave. When they begin to exit the train, each person will be queued at the edge of their row of seats on the aisle. We also assume that very person moves at .975 meters per second while

[1] http://www.apta.com/resources/standards/Documents/APTA-PR-CS-S-016-99.pdf
[2] http://oregonstate.edu/ehs/sites/default/files/pdf/ergo/ergonomicsanddesignreferenceguidewhitepaper.pdf
[3] http://easts.info/on-line/proceedings/vol9/PDF/P269.pdf
[4] http://www.apta.com/resources/statistics/Documents/transit_passenger_characteristics_text_5_29_2007.pdf
[5] http://www.rehab.research.va.gov/jour/93/30/2/pdf/oberg.pdf

exiting the train. This is ¾ the average walking speed of a commuter (.975 meters per second), which is reasonable since the train will be less spacious for people to travel at their own normal walking speeds. Moreover, passengers will file out with the aisle man in the 3-row going first, and then alternating from there. They will also use the exit in the train that is on their half of the train.

Assumption 8: The platform is longer than the train. If it weren't, people would have to walk on the tracks, which would be devastating.

Assumption 9: A person goes to the staircase nearest to where they exit the train. In a study published by the University of Washington[6] on crowd walking dynamics, the first thing people consider is the geographical end goal, second is their speed. Following this idea we figured out which staircase would be closest to an individual and simulated them exiting there, only taking into account the closeness of an exit and not the associated wait time.

Assumption 10: When the train pulls into the station, it stops evenly at the middle of the platform. In real life, it is random as to where the train stops relative to the platform and varies with the conductor. In order to maintain some sort of consistency, we had our train stop when its middle lines up with the middle of the platform, which simplifies our calculations but also reasonably accounts for the deviation between conductors who stop before this line or after this line (See Figure 6 Appendix D).

Assumptions about Dimensions:

- According to railsimulator.com[7], the M8 commuter railcar is 3.2 meters wide. The M8 made by Kawasaki is a great example to look to because it is a commuter rail with rows containing three seats on one side of a middle aisle, and two on the other.
- The platform is five meters in width. This makes sense because if the trains are 3.2 meters wide, the platform in the middle should be wider than one train itself, but with trains on either sides it is logical to have a platform substantially wider than an individual train
- There will be one meter between train cars. This makes sense because it is a short enough length for a serviceman to walk between cars, but it is long enough to properly bend as the train goes around a curve. In observing images of trains and considering their scale, we determined that this was a reasonable dimension for the car couplings.
- Stairs have a vertical length of .2 meters, and a horizontal length of .25 meters.
- The width of the wall of the train car is .3 meters.

[6] http://grail.cs.washington.edu/projects/crowd-flows/continuum-crowds.pdf
[7] http://www.railsimulator.com/resources/222694/41958e0221e7c4e8a54b88fb68615e83/M8%20Manual.pdf

VARIABLES

We are given four variables in the problem:

n - the number of cars on the train

d - the length of one car in meters

p - the length of the platform in meters

q - the number of steps on a staircase to get to street level

k - the number of people (side by side) that a single step can accomodate

In order to better the model the central train station, we needed to create more variables though, because the four above don't begin to account for the number of staircases, or staircase positioning. Therefore, we added:

v - an individual's walking speed in meters per second

s - the number of staircases

z - the positioning of a given staircase, anywhere between 0 and ***p***

T - the average time it takes all of the people on the train to get from their seat to the street in seconds, and the variable we are solving for

COMPUTER SIMULATION ALGORITHM

The model that we created is a computer simulation written in java that returns the average time that it takes a passenger to reach the street from when the train stopped at the station. The model accepts different variables for ***n*, *d*, *p*, *q*, *s*, *k*,** and the **number of trains.** These values are then used in conjunction to calculate an average exit time.

The calculation of this average time depends on calculating the exit time for each and every passenger. The number of passengers can be calculated based upon the dimensions of the M8 train by kawasaki, and the number of cars (***n***). The distance between seats on the M8 train is 1 meter, and since there are 2 exit aisles at each end of the train car, the number of rows of seats per car will be $\lfloor d-2 \rfloor$. This is because if there is a non integer length for another row of seats to be placed, the seats will simply not be created. This value can be multiplied by 5 in order to find the total number of passengers per car, since each row has exactly 5 seats in it. To find the total number of passengers on the train, that value is multiplied is multiplied by ***n***, yielding the equation $n * 5 * \lfloor d-2 \rfloor$.

Using the aforementioned value, a one dimensional array containing elements of class "person" is created with a length of $n * 5 * \lfloor d-2 \rfloor$. Each element of this array has a specific

position on the train which is defined by which ***train-car*** they are on, their specific ***row***, and what we called their "***column***" within their row. Each "person" is able to keep track of the time it takes them to do each specific step, which will be accessed later on. The time of the journey can be divided into three unique portions: one, time spent getting off of the train; two, time spent walking from the train to the stairs; three, time spent from arrival at the staircase to arriving at the street.

Time spent getting off of the train can be split even further into two separate sections: one, the time spent waiting to begin walking; two, time spent walking off of the train. The first time can be determined based upon each passenger's specific position. Firstly, whether a passenger exits off of the front exit or the back exit depends upon their position within the train car. If the ***row*** of a passenger is greater than $\lfloor \boldsymbol{d} - 2 \rfloor / 2$ then the passenger goes to the back exit, otherwise they will use the front exit. The equation for the time spent waiting depends upon which exit the person goes to. Each person's wait time is explicitly given by a formula due to the defined order which each passenger exits, shown in *Appendix B*. This formula is of the form (people before them)*(individual wait time) + (reaction). Reaction is the time between when the train stops and the first person exits, this is assumed to be 2 seconds. Additionally, the individual wait time which is the time that it takes between when a person begins walking and the person behind them begins walking is also assumed to be 2 seconds. For those exiting out of the front exit, the number of people before them is (***row***-1)*5+***column***-1. For those using the back exit, this value is ($\lfloor \boldsymbol{d} - 2 \rfloor$-***row)****5+***column***-1. Thus if someone is in row > $\lfloor \boldsymbol{d} - 2 \rfloor / 2$ their wait time is (($\lfloor \boldsymbol{d} - 2 \rfloor$-***row)****5+***column***-1)* 2 + 2. Otherwise, their wait time will be ((***row***-1)*5+***column***-1)*2 + 2.

To determine the time spent walking off of the train, the distance they walk from the aisle to the door can be calculated depending upon which exit they use. If they go out of the front exit, the distance is ***row*** + 1.43. For the back exit, this value is $\lfloor \boldsymbol{d} - 2 \rfloor$ - ***row*** + 2.77. Both of these distances are outlined in *Appendix C.* These distances can be divided by the train walking speed which is assumed to be .975 m/s in order to determine the time it takes to walk off of the train. By adding these two times together, the total time spent getting off of the train can be determined.

Once each person gets off of the train, they begin walking to the nearest staircase. Which staircase is nearest is determined by comparing the distance from the front end of the platform of where each person exits the train with the distance from the front end of the platform of each staircase. The distance where a person exits if its from the front can be given by the equation:

$$\frac{\boldsymbol{p} - (\boldsymbol{n} * \boldsymbol{d}) + \big(coupling\ length * (\boldsymbol{n} - 1)\big) + (2 * \boldsymbol{n} * wall\ width)}{2} + (\boldsymbol{n} - 1) * (\boldsymbol{d} + chain + 2 * wall\ width) + wall\ width$$

If they are exiting from the back exit, it is the same, however ***d*** is also added to the distance. In that above equation, coupling length which is the distance between train cars is 1 meter, and the

wall width is .3 meters. After determining which exit is the closest, the person begins walking in a straight line towards the stairs. The distance that they travel can be given by the pythagorean theorem. It is simply the square root of the absolute value of the difference in distances from the front end squared plus the half of the width of the platform squared. In this case, the width of the platform is 5 meters. This distance that the person must travel can be expressed symbolically as $\sqrt{(xdistance\ of\ where\ exited - xdistance\ of\ closest\ staircase)^2 + \left(\frac{platform\ width}{2}\right)^2}$. To see a diagram of this distance, see *Appendix D.*

One property of the class "person" is that each instance has a unique walking speed. These speeds follow are attributed randomly in accordance with a normal distribution with a mean at 1.3005 m/s and a standard deviation of .1705. In order to get the time that it takes each person to travel to whichever exit is the nearest, the previously determined distance is simply divided by the randomly determined speed.

The third and final section of the time it takes is the time from when a person arrives at the stairs to when they reach the top. The way that this works is that every single passenger is sorted into an cue of which staircase they are going up ordered by how long after arrival of the train they arrived at the staircase. The time that it takes can be broken down further into two sections. The first is the wait time. This can be determined by making an array of the time when each person going up this staircase began climbing the stairs. The first $\boldsymbol{k}$ people to arrive at the stairs will have a wait time of zero. Any person who then arrives 2* time per stair seconds after the person $\boldsymbol{k}$ in front of them in the cue will have no wait time. Whenever someone begins to walk up the stairs that time is saved into the array. Any person for whom the person $\boldsymbol{k}$ in front of them began walking up the stairs more than 2 *time per stair seconds after they arrive will have to wait until the person $\boldsymbol{k}$ in front of them begins walking + 2 * time per stair to begin walking up the stairs. This wait time calculated can then be added to the time that it takes to walk up the stairs which is $\boldsymbol{q}$ * time per stair.

The total time for each person is:

1. If they exit from the front

$$\left((\boldsymbol{row} - 1) * 5 + \boldsymbol{column} - 1\right) * 2 \ + \ 2 + \frac{\boldsymbol{row} \ + \ 1.43}{.975} +$$

$$\sqrt{(xdistance\ of\ where\ exited - xdistance\ of\ closest\ staircase)^2 + \left(\frac{platform\ width}{2}\right)^2}$$ / random velocity ($\mu = 1.3$ s = .17)

+recursive wait time (array) + $\boldsymbol{q} * time\ per\ step$

2. If they exit from the back

$$(\lfloor d-2\rfloor - \boldsymbol{row}) * 5 + \boldsymbol{column} - 1) * \ 2 \ + \ 2 + \frac{\lfloor \boldsymbol{d}-2\rfloor - \boldsymbol{row} \ + \ 2.77}{.975} +$$

$$\sqrt{(xdistance\ of\ where\ exited - xdistance\ of\ closest\ staircase)^2 + \left(\frac{platform\ width}{2}\right)^2}$$ / random velocity (μ = 1.3 s = .17) +recursive wait time (array) + $\boldsymbol{q} * time\ per\ step$.

In order to get the average time that it takes to get out of the station which is defined by all of these variables, using a for loop, the sum of each passenger's exit time is determined, which is divided by the number of passengers in order to get the average exit time. There is a larger encompassing for loop which can run this simulation multiple times for each set of parameters in order to get a large amount of data.

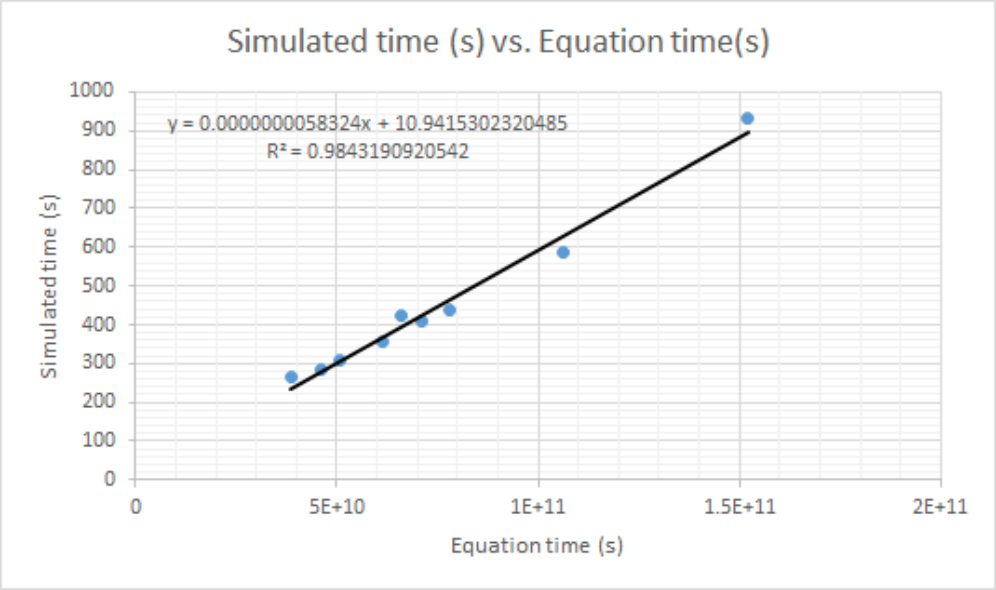
Additionally, the locations of the staircases are manipulatable, so the closest staircase distance can be altered by changing both the location of these staircases, the amount of them. Their ***k*** can also be manipulated which will affect the array wait time. The number of trains that arrive simultaneously can be changed to two, where all of the passengers on the second train are simply added to the array of passengers

To see the code, see *Appendix E.*

MODEL DESIGN, JUSTIFICATIONS, AND CONCLUSIONS

Requirement 1: *One fully occupied train's passengers to exit the train, and ascend the stairs to reach the street access level of the station.*

For this part of the problem we made our model generate an average time for the scenario in which there is one stairway (***s*** = 1) in the middle of platform (which aligns with the middle of the arriving train. This constant stairwell functions in conjunction with alterations of ***n***, ***d***, ***k***, ***q***.
A standard for all of our trials was ***n*=15**, ***d*=20**, ***p*=400**, ***q*=50**, ***s*=1**, and ***k*=2**. These values allowed for sufficient manipulation, up or down. After finding individual **F(*n*) = 36.10*n*+4.74**, **F(*d*) = 26.04*d*+19.54** , **F(q) = .55*q*+515.67** and **F(*k*) = 813.01*k*$^{-0.53}$** equations we determined that T ~ A_1*F(***n***)*F(***d***)*F(***q***)*F(***k***) with A_1 as some constant. In order to calculate the value of A_1, we graphed the values our simulation gave us for a given set of parameters on the y-axis and the value that multiplying the four functions above gave us on the x-axis. This formed an accurate line, with an R^2 value of .984. A_1 is the slope of this line, **$5.8324 * 10^{-9} = A_1$.** See figure below for graph. It should be noted that the variable p does not play a role in a our model for exit time, because p is known to be longer than the length of the train, but the staircases will not pass the ends of the train so people's walking times are negligibly affected by it.



$T(n,d,q,k) = (36.10n+4.74) * (6.04d+19.54) * (.55q+515.67) * (813.01k^{-0.53}) * 5.8324 * 10^{-9}$

Requirement 1 asks us to minimize the time traveled for the average commuter, but obviously to minimize this, one could say each parameter should be as small as possible. For instance if there were 1 train car (***n***), it were 10 meters long (***d***), the station were only 5 steps below street level (***q***) and the stairways could accommodate 10 people side by side (***k***) seat to street would be incredibly fast. However, the working model above, which comes very close to the simulation (R^2 of .984) is useful for the person in charge of creating the best possible train station because the director of transportation can evaluate how much proposed changes would change commute time. He could evaluate if the changes would ultimately be cost efficient.

Requirement 2: *Two fully occupied trains' passengers (all exit onto a common platform) to exit the trains and ascend the stairs to reach the street access level of the station.*

In the computer model that we created, the situation in which two fully occupied trains pulled into the station can be modelled by simply adding all of the passengers of the second train to the array of people. In the computer model the two trains have identical ***d*** and ***n*** in order to fully understand the impact of having a second train upon the average time. It is reasonable to assume that this second train is identical to the first one because typically specific transit authorities will use one specific type of train or a few types with almost identical dimensions. Additionally, had the second train not been identical, than it would be impossible to make a function for the data gathered because a certain ***T*** for one train would have an unreasonably large amount of possible corresponding ***T***s for the 2 train simulation because there are so many different possible combinations of ***n*** and ***d*** which would yield unique values of ***T***.

The linear model that was yielded is rather accurate, however due to the huge amount of variables that are involved in running each simulation to get the most accurate possible prediction for the time that it takes when there are two trains the recommended manner would be to simply run a computer simulation with your specific parameters using the attached code in *Appendix E*. Yet, the linear model (*figure 2.1)* was still able to yield an R^2 value of .915. This means that 91.5% of the variability that is observed in the data is explained by the model. This value is rather impressive provided that there are so many variables and that all of these can simply be explained by a ratio.

The way that the time for two trains works is that the first two "sections" measured (outlined in the algorithm of the computer model) are not at all affected by the fact that there is a second train. This is due to the fact that firstly, the time to get off of a train cannot possibly be affected by people on an entirely different train, and secondly, since each person walks directly to the nearest staircase, it is geometrically impossible for their paths to cross since it would entail going away from the most efficient path to cross over into the other half of the platform. Thus, the only change in time would be experienced at the stairs. Although the time for if two trains appeared simultaneously can be predicted with a degree of accuracy by simply using the equation Time for 2 trains = 1.8854(Time for 1 Train) - 73.033 it is not actually the best way to predict the change in the exit time if a second train appeared without a computer simulation

The best way to predict the way the time would change would be to simply change the time spent at the stairs. The relationship between the stair time for 2 and 1 train can be expressed by the equation Stair Time for 2 Trains = 2.1007(Stair Time for 1 Train)+ 31.3 (*figure 2.2)*. This equation is a highly accurate model, with an R^2 value of .9883, meaning that an extraordinary amount of the variance in the time spent at the stairs can be expressed this way.

Thus the best way to express the way that the average exit time would vary would be to replace the "stair time" from your previous situation with 2.1007* stair time + 31.3. This would yield a highly accurate model for how long it takes if there are 2 simultaneously arriving trains. On the other hand, there is still a passable model (*figure 2.1)* which can produce an acceptable prediction for the exit time if the only data that you have is the average total exit time.

figure 2.1

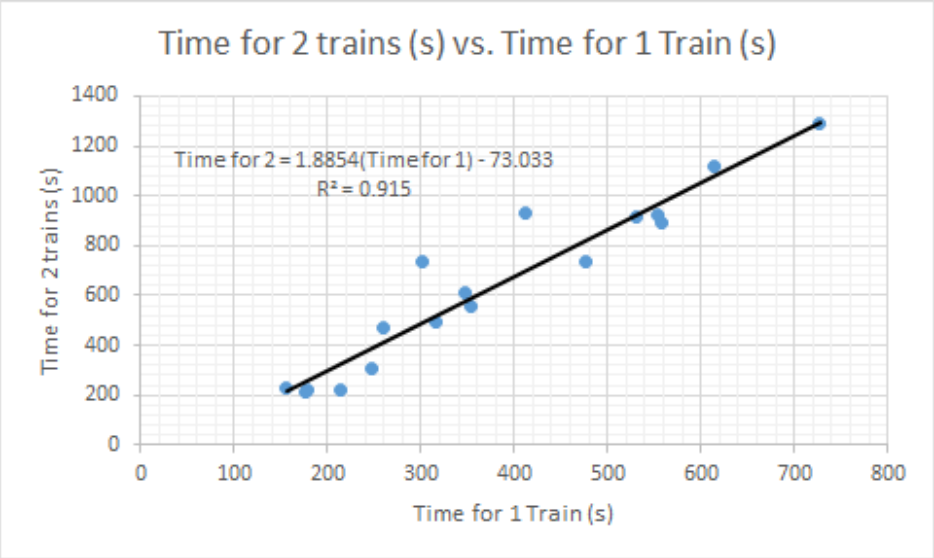
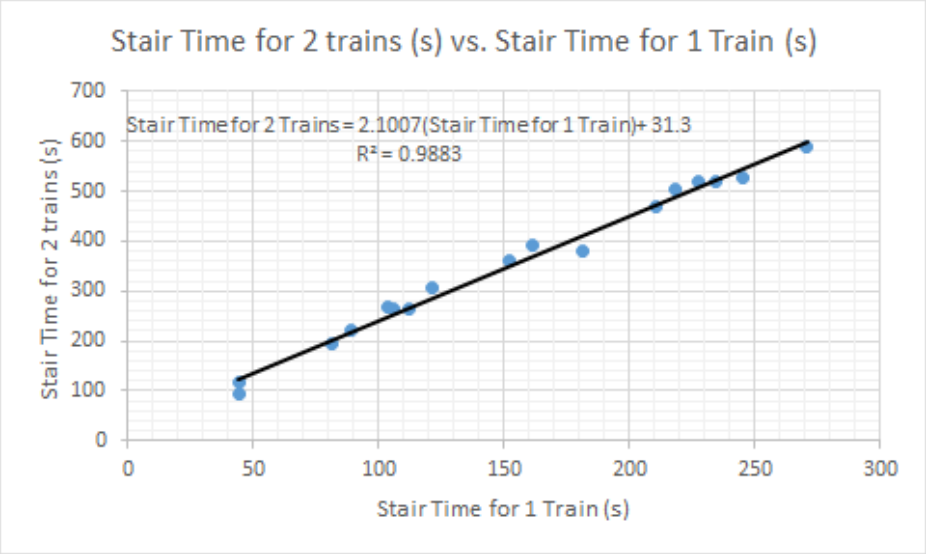
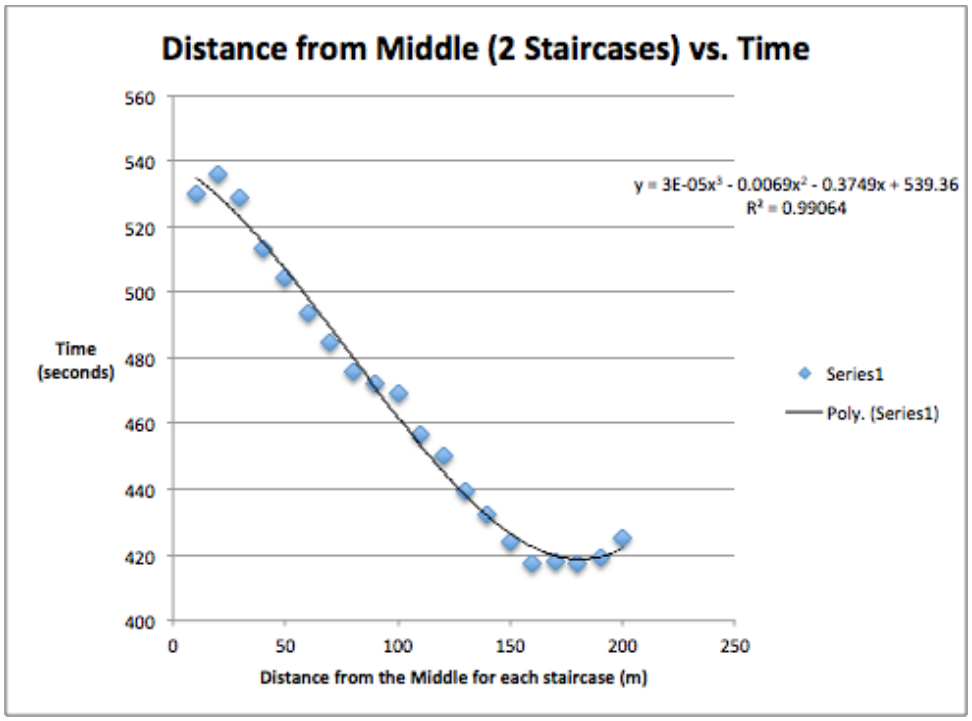


figure 2.2



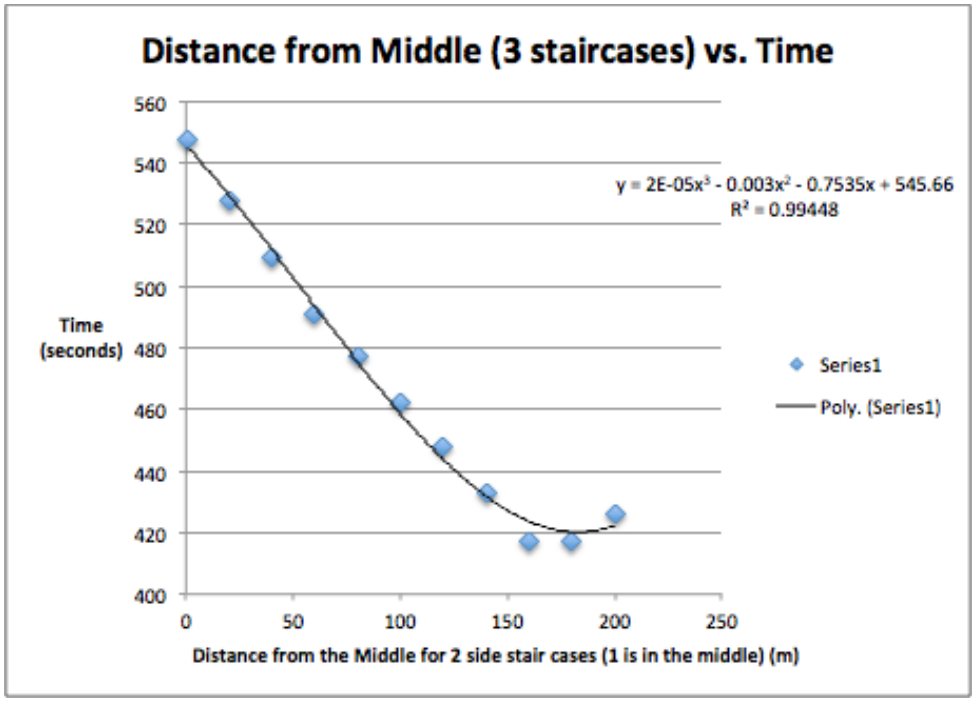
Requirement 3: *If we could redesign the location of the stairways along the platform, where should these stairways be placed to minimize the time for one or two trains' passengers to exit the station?*

Testing for best placement of 2 stairways:



Let's call Distance from the Middle ***m*** **(**See Appendix D). The resulting function from this data is a third degree polynomial, ***T(m)***, which means that there is an ideal point at which the time will be at a minimum, a point that can be found by finding when the derivative of this function, $T' = 3x10^{-5} * m^2 - .0138m - .3749$, is zero. When this derivative is zero, ***m*****=178.88**, or the two staircases are located 178.88 meters away from the center of the platform on opposite ends. This is very close to locating the staircases at the end of the platform, the ends being 200 meters from the center of the platform, so it seems as though the ideal location for the staircases is at each end of the platform.

For 3 stairways:



In this case ***T(m)*** is very similar, different only in the placement of the third staircase in the middle of the platform. We performed the same analysis to get the value of *m* that gives us the minimum value for ***T***, where $T'(m) = 6x10^{-5} * m^2 - .006m - .7535$, and ***T'(m)*** is equal to zero. The result was that the minimum occurs at ***m*=172.71**, or that the time is minimized when one of the three staircases is in the middle of the platform and the other two are located at opposite ends of the platform, 172.71 meters from the center of the platform. This is very close to the value of ***m*** when there were two staircases, which makes sense since the arrangement are nearly identical save the one staircase in the middle.

Our initial thought when we saw how close ***m*** was to the end of the platform was that the ideal location would actually be at the end of the platform, and the only reason that our data did not explicitly show this phenomenon was because our use of actual data was causing inaccuracy. However, after finding ***T(m)*** when there are three staircases had a minimum in nearly the same location, it is not as apparent that the ideal location is on either end of the platform. If we had more time, we would take more data points in order to find a more accurate average either confirming our findings or challenging them. Also, we decided to make the two staircases mirror each other across the middle of the platform because it is fairly intuitive that this would yield the most efficient times. However, we never confirmed this, and had we had more time, we would have tested different arrangements that we not symmetrical in order to either confirm or challenge our assumption.

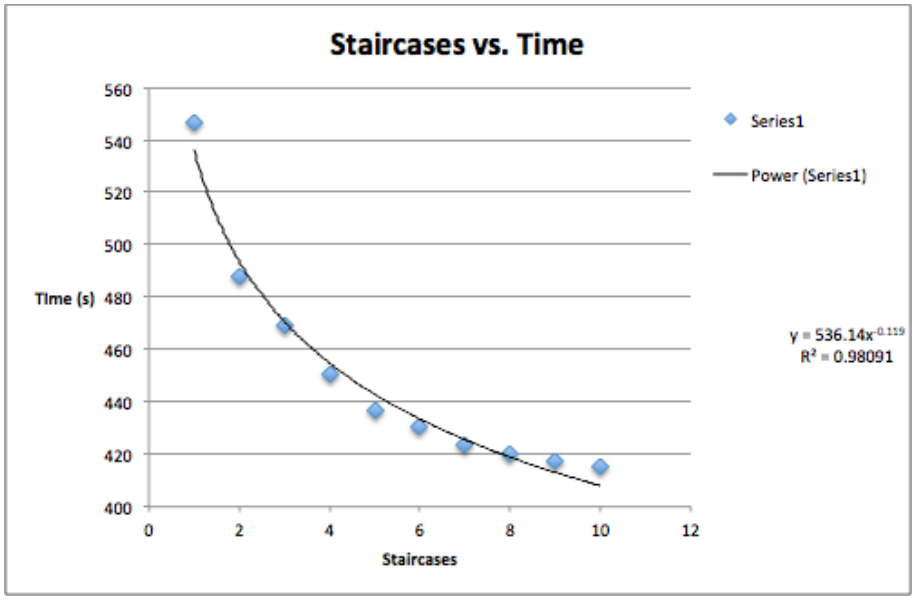
Requirement 4: *How does the time to street level vary with the number s of stairways that one builds?*

To isolate how number of staircases affects the seat to street exit time for a train passenger we looked at how placing s staircases symmetrically across the platform of length p affects exit time in our simulation.
In simulation code we used a for loop to assign each staircase we were dealing with a value along the platform (a number 0-p, with 0 and p being the ends of the platforms). The second parameter in a 'Staircase' is the location along the platform.

```
for(int qw = 0; qw < numberStaircases; qw++)
{
      exits[qw] = new Staircase(k, (qw +1) * p /(1 +numberStaircases));
}
```

For instance given 2 staircases, this for loop assigns location (⅓)P to the first staircase, and (⅔) to the second staircase. This gave us a systematic way to evaluate times as the number of staircases increased.

Again, we cannot minimize this function, because as more stairways are added, the average travel time would decrease because the whole platform becomes less congested. The graph would truly display asymptotic behavior in a scenario in which everyone had no wait times at $\boldsymbol{T = 536.14\, S^{-.119}}$ makes sense because the change from one to two staircases would clear up a substantial amount of traffic by getting rid of the one staircase in the middle and adding two on the sides, whereas the change from 2 to 3 staircases in the model simply adds a third staircase between the existing 2.

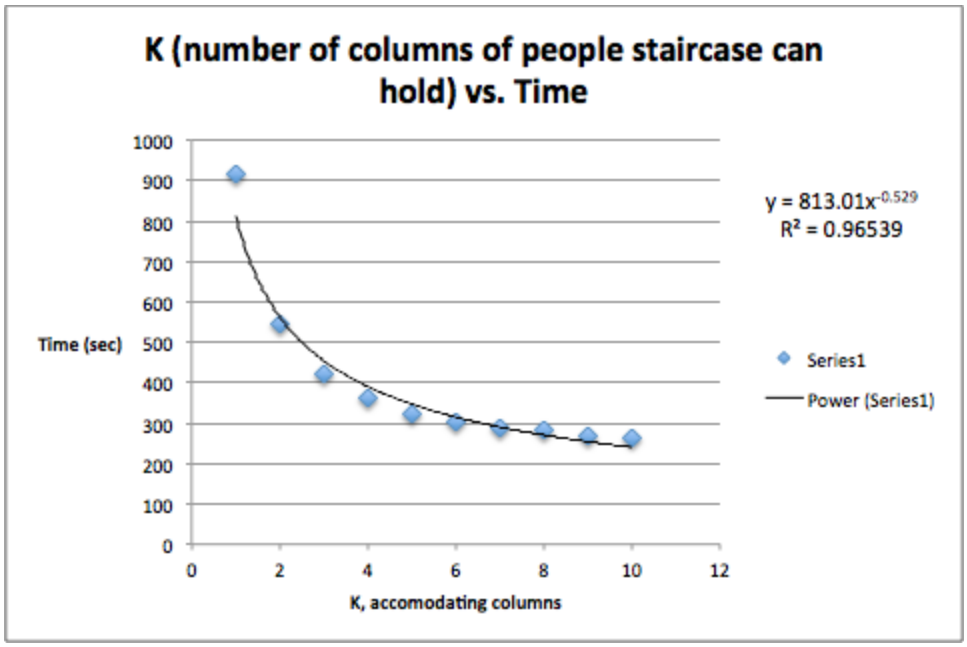


Similarly to how we found a model for time in terms of all variables excluding s, we made one for time that includes the number of staircases, evenly distributed by creating another graph of model values versus simulation values and using the slope as our constant. In this case, 1.715238 * 10^{-11}. Unlike our initial model, this graph had a y-intercept of -213.8, which tells us we must subtract this off of our model.

The model, including number of evenly distributed staircases:

$$T(n,d,q,k,s) = (1.715238 * 10^{-11})\frac{(36.095n + 4.7382)(26.043\,d + 19.536)(0.5485q + 515.67))(813.04)(535.14)}{(k^{0.529})(s^{0.119})} - 213.8$$

Requirement 5: *How does the time vary if the stairways can accommodate k people, k an integer greater than one?*



K is equal to the number of people that one stair can accommodate, or, equivalently, the number of columns of people exiting at once. If all variables (***n***, ***d***, ***q***, ***p***, ***s***) except ***k*** are held constant, the variation of the time in seconds it takes to leave the station is in accordance with the equation:

$$T = C * k^{-0.529}$$

Where C is some constant determined by the other variables. This is nearly an inverse square root relationship, from which it is clear that the difference in the total exit time decreases as ***k*** increases. This logical relationship stems from the fact that the rate of exit depends on the number of people that each stair can fit at once, Thus, if ***k*** is increased from one to two, the rate should increase much more than if it was increased, say, from ten to eleven. Consequentially, the exit time varies most when ***k*** is close to 1.

DISCUSSION

Model Testing

Real World Scenario: As mentioned before, the train described in the problem bears a striking resemblance to the M8 train designed by Kawasaki which is currently in use by the Metro-North's Railroad New Haven Line which runs between New York City and Connecticut. The problem tells us to look at a "central" station, that is nexus for commuters. Coincidentally, the New Haven Line runs through Grand Central Station in New York City as well as other large train hubs. The M8 commuter train is 25.9 meters in length, and commonly is run with 10 train cars. According to the Metro Transit Authority[8], Grand Central Station's New Haven Line terminal is 310.9 meters in length and the New York Times[9] reports that it is 13.7 meters underground. Additionally, in the MTA's own model of grand central station New Haven Line[10] traffic flow, the staircases can accommodate 3 people, and there are 6 staircases to a platform.

Testing:
n = 10 cars
d = 25.9 m
p = 310.9 m
q = 69 steps

s = 6 staircases
k = 3 persons stair accommodation
z(positioning of staircases) = even distribution, meaning one staircase at 1/7*P, the next at 2/7*P, etc.

After plugging these parameters into our model that accounts for number of staircases (because Grand Central Station has six, not one):

$$T(n,d,q,k,s) = (1.715238 * 10^{-11})\frac{(36.095n + 4.7382)(26.043\ d + 19.536)(0.5485q + 515.67))(813.04)(535.14)}{(k^{0.529})(s^{0.119})} - 213.8$$

We yielded a time of 374 seconds, which is equivalent to 6 minutes and 14 seconds. This increased our confidence in the model, because according to the MTA[11], from the deepest point of Grand Central Station it takes approximately 10 minutes to exit at street level. The additional depth that must be covered on foot, as well as the great distances commuters must walk through once they've reached street level (Grand Central Station main room) indicate that for a normal level just over 6 minutes is likely accurate.

[8] http://web.mta.info/capital/esa_docs/esa_data_card.pdf
[9] http://www.nytimes.com/2013/01/20/nyregion/the-birth-of-grand-central-terminal-100-years-later.html?pagewanted=all
[10] http://www.cb5.org/cb5/projects/east_midtown_rezoning/MTA_East_Midtown_Presentation_-_October.pdf
[11] http://web.mta.info/capital/esa_docs/eafiles06/Appendix%20B%20Upper%20Level%20Loop%20Alternative%20Analysis.pdf

Weaknesses in the Model

Our computer simulation was not as efficient as it could have been. Running our simulation and gathering data took a long time due to the inefficiency of the program causing a slow production rate of reasonable data. There was a fairly wide range of values given by the program, but we took the wide range and converted it into an average in order to get a mean for each data point. However, if we had had more time, we would have gone back and made the code more streamlined and efficient.

The primary source of error in our model is pertaining to how a given person chooses the staircase he will use to exit. We assumed that when an individual exits, he will choose to seek out the staircase closest to him, and use it. However, this ignores the scenario where there are extreme wait times at one staircase, and low or no wait times at a different, but farther, staircase. In real life, if an individual were in a rush he would dart to that staircase, and take it. Similarly, if the weather is not temperate people may attempt to walk faster than normal adjusting the whole model. However, there is no possible way to understand what is going through the heads of our hypothetical commuter population.

Secondly, people will not be able to walk in straight lines to their staircase. In real life, people will most likely weave around slower traffic on their way to the staircase, and the path to the stairs will be slightly longer due to this phenomenon. However, it is reasonable to assume that the effect of this is mostly negligible, since no one will go too much out of their way. Nonetheless, it is a weakness in our model.

Another more unrealistic part of our simulation was that the staircases were treated like points, and not like three-dimensional objects. In our simulation, when a person reached a staircase, they did not have to walk around in order to get to the opening. Once they reached the staircase, they could either immediately begin walking up or wait until the person in front of them was sufficiently up the stairs. In real life, traffic can build up at staircases due to the fact that the opening to the staircase will only face one direction, and passengers who are not approaching the staircase from that direction will have to maneuver around the staircase in order to arrive at the opening. If this were a factor, we would have to account for the extra distance when someone is approaching from the opposite direction of the staircase opening, and when they would be let into the line forming at the entrance to the staircase. Had we had more time, we may have been able to account for this in the simulation, a factor that could potentially affect our results respective to number of staircases and the arrangement of the stair cases.

APPENDIX

A. Graphs

Figure 1:

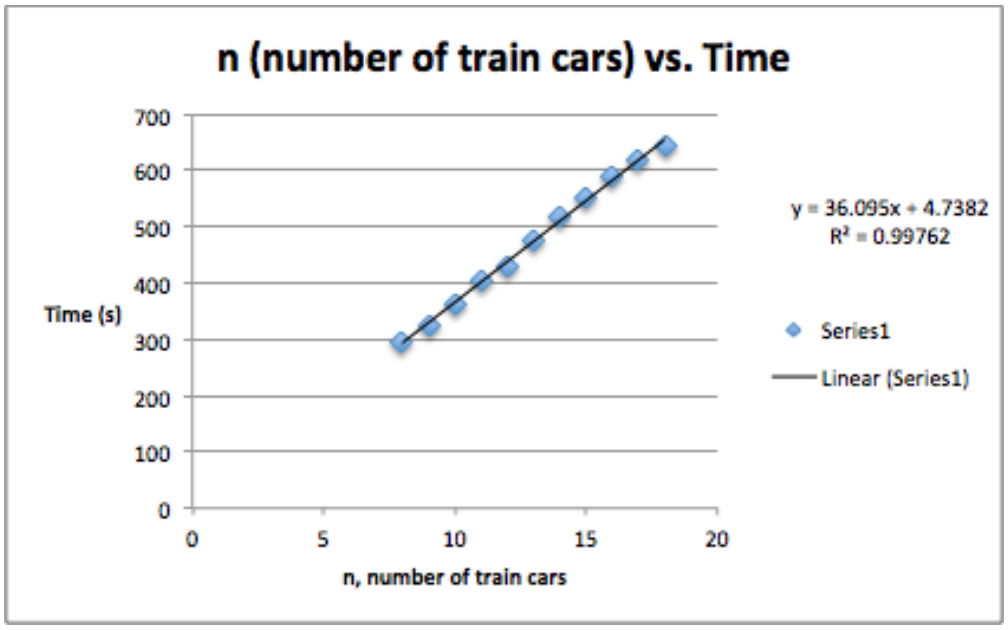


Figure 2:

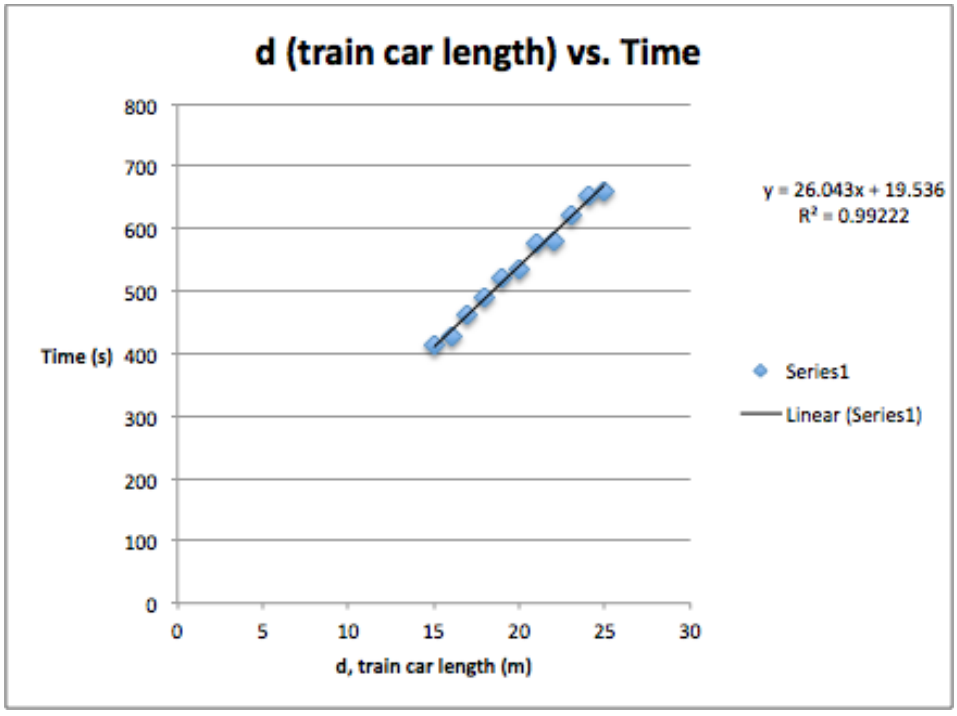
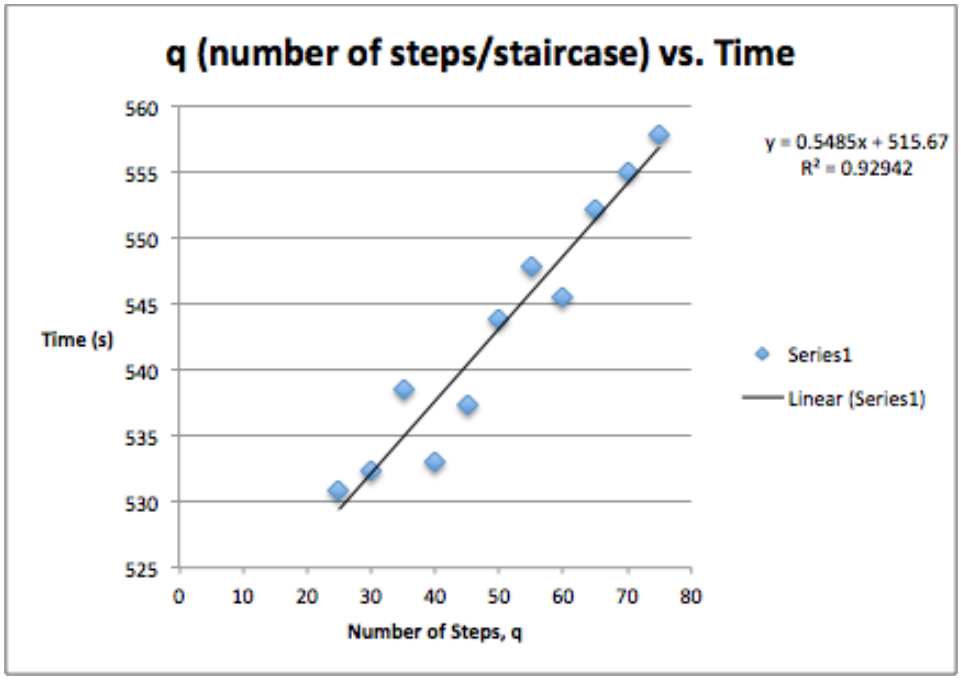
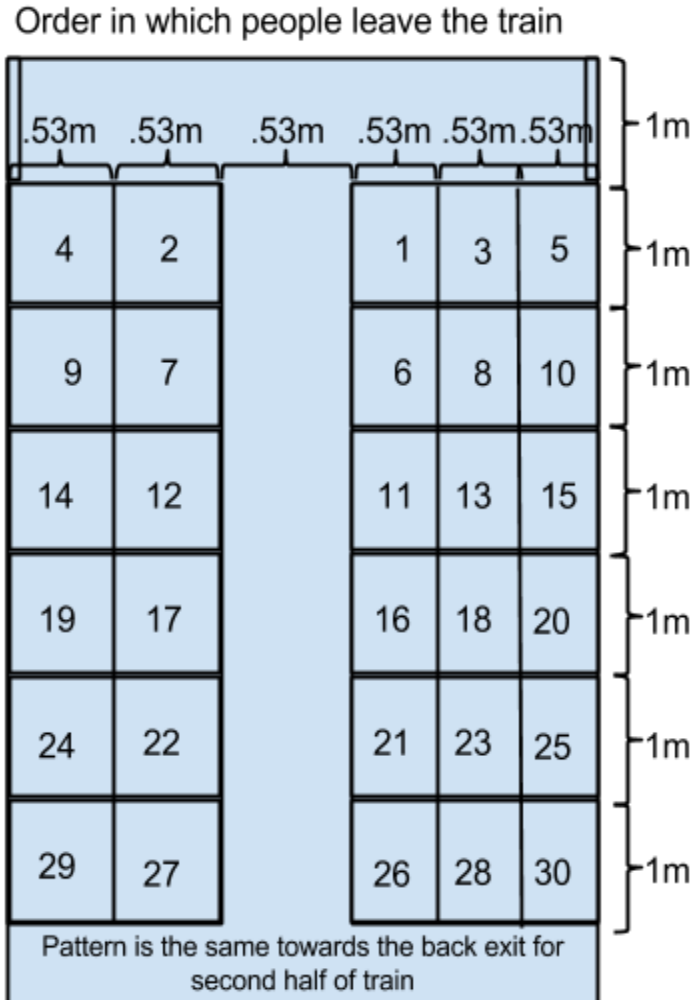


Figure 3:


q (number of steps/staircase) vs. Time
Time (s)
560
555
550
545
540
535
530
525
0
10
20
30
40
50
60
70
80
Number of Steps, q
y = 0.5485x + 515.67
R² = 0.92942
Series1
Linear (Series1)

B. *Diagram of Order of Leaving Train*

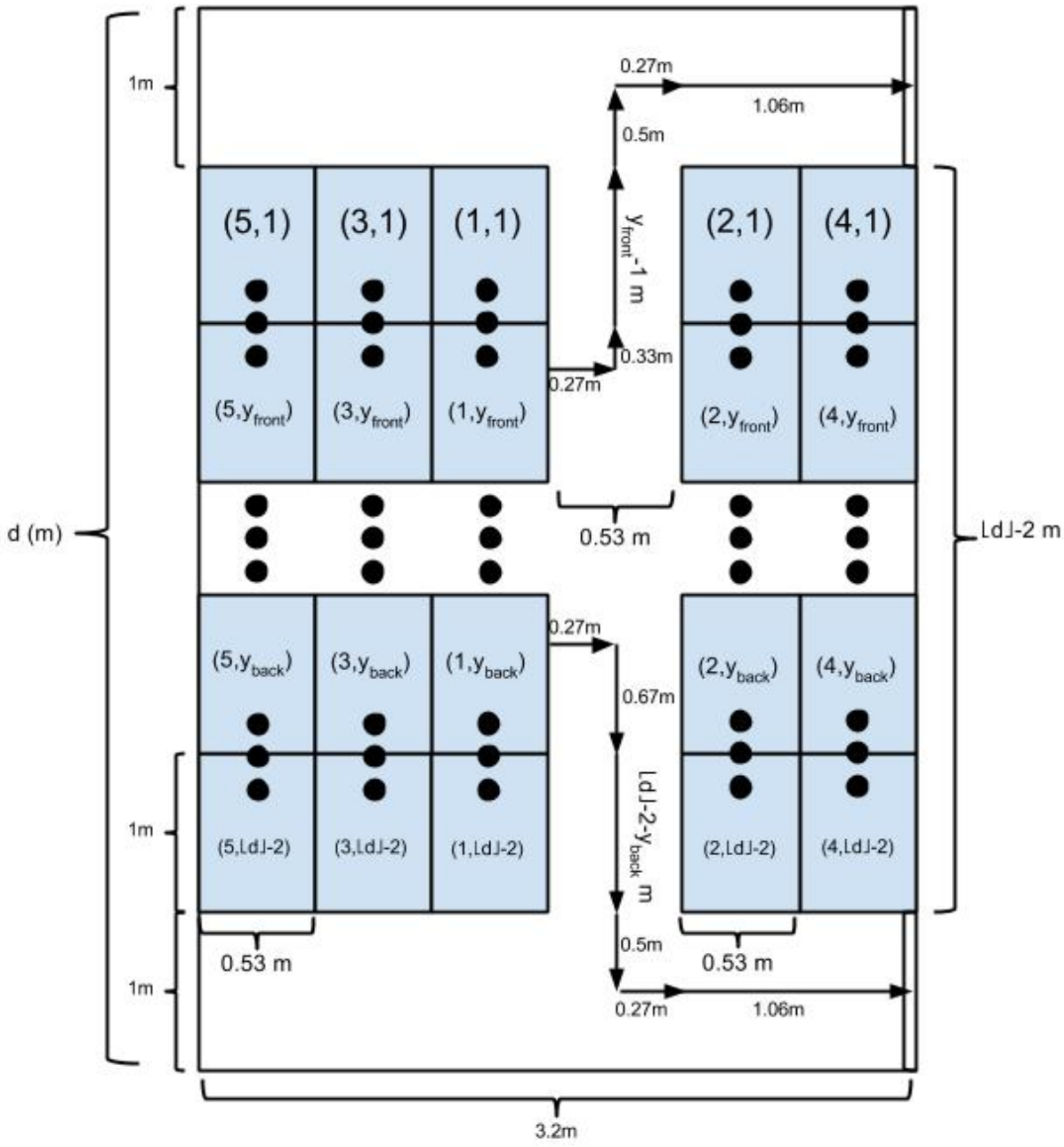
Figure 4:



In this diagram, the platform is on the left side and passengers exit out the doors on that side.

C. Diagram of Distances of Leaving Train

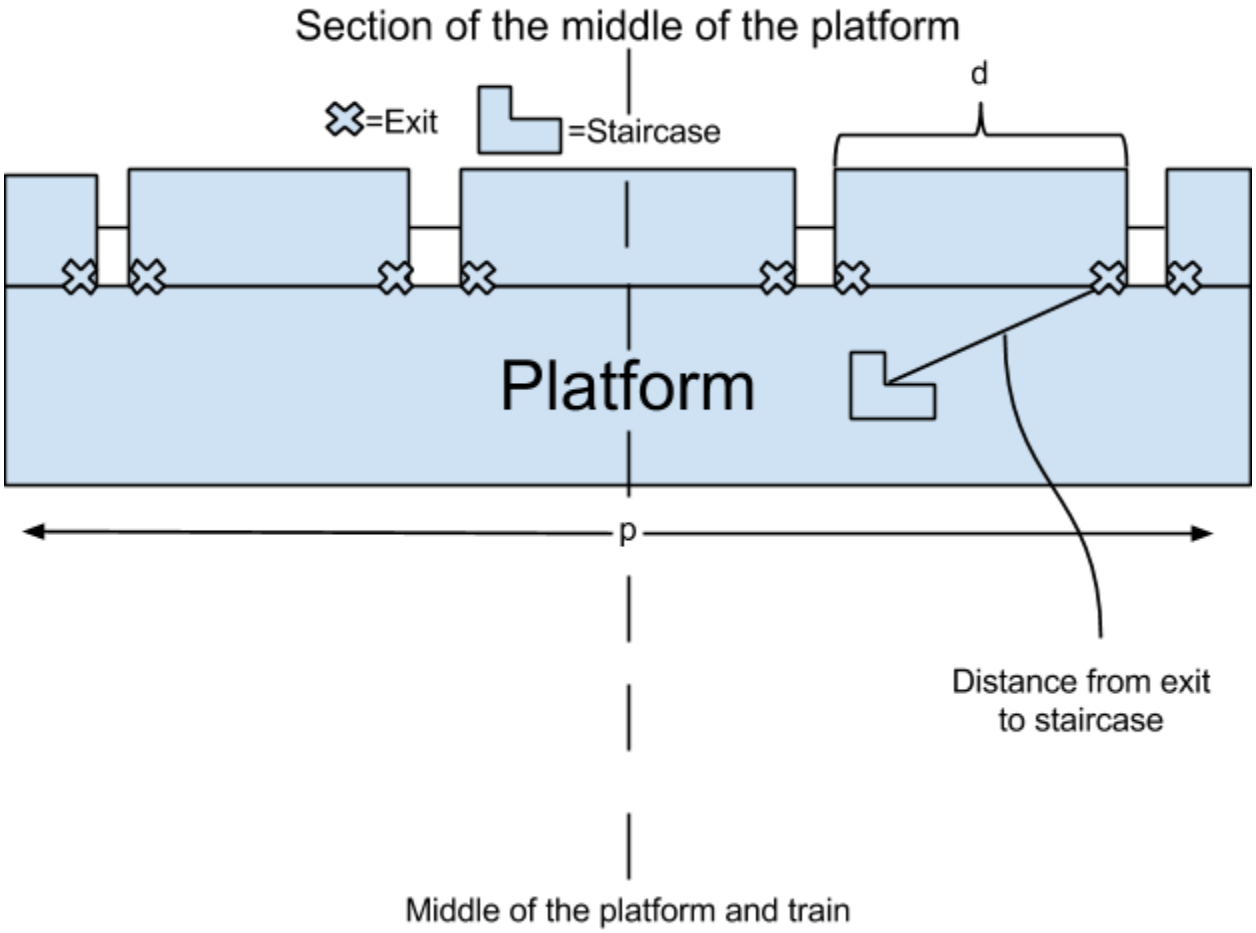
Figure 5:



In the diagram, the platform is on the right side and people exit out these doors. Each seat has some coordinate (x, y). x is the column that the seat is in, from 1 to 5, and the columns are numbered in the order that people leave. y is the row the seat is in, from 1 to ⌊d⌋-2.

D. Diagram of where the train pulls into the station

Figure 6:



E. Code for Simulation

```
import java.util.ArrayList;
public class exitSimulation
{

	public static void main(String[] args)
	{
		//All of the variables defining the situation are defined here. As well as those defining a second train
		int n, n2, q;
		double d, d2, p;
		//will be used to calculate average times
		double sumOfTimes = 0;

		//parameters for train station
		//p = length of the platform (m)
		p = 400;
		//q = number of stairs to get to street level
		q = 25;

		//parameters for train 1
		//n = number of train cars
		n = 18;
		//d = length in meters of the inside of each train car
		d = 23;


		int numberOfTrains = 1;

		//parameters for train 2.
		//Only applicable if numberOfTrains is 2.
		//n2 = the number of train cars in this second car
		n2 = n;
		//d2 = the length in meters of train cars for this train
		d2 = d;

		//k is the number of columns of people that can be walking up the stairs at any given time
		int k = 4;

		//the number of staircases can be defined
		int numberStaircases = 1;

		//clarify if on each side
		boolean equallyDistanced = true;
		boolean random = false;
		boolean arbitrarilyAssign = false;
		boolean completelySymmetrical = false;
		//defines the array of staircases
		Staircase[] exits = new Staircase[numberStaircases];

		//Based upon the parameters entered we can easily define the placement of the staircases

		//The following methods are different ways of creating and situating all of the staircases.
		if(numberStaircases == 1)
		{
			//When there is only one single stairway it is located in the middle
			exits[0] = new Staircase(k, p/2);
		}
```

```
		else if(completelySymmetrical)
		{
			//This method situates the staircases at constant increasing intervals away from the center in a symmetrical fashion where
			//all distances away are defined by the one that is farthest
			//The length that they are away can be varied

			//this distance, in meters is how far the farthest staircase is from center
			double distanceFromCenter = 200;

			//creates the number of staircase that is specified
			for(int l = 0; l < numberStaircases; l++)
			{
				//Places the staircases in a symmetrical manner
				exits[l] = new Staircase(k, ((p - (2 * d))/2) + ((2*d)/numberStaircases) * l);
			}
		}
		else if(random)
		{
			//This uses changeable values for the number of staircases and their capacity to create an array of object Staircase places the staircases randomly

			for(int i = 0; i < numberStaircases; i++)
			{
				exits[i] = new Staircase(p, k, 1);
			}
		}
		else if(equallyDistanced)
		{
			//in this every staircase is equally spaced from one another

			for(int qw = 0; qw < numberStaircases; qw++)
			{
				exits[qw] = new Staircase(k, (qw +1) * p /(1 +numberStaircases));
			}
		}
		else if(arbitrarilyAssign)
		{
			//every single value in here is altered as wanted/needed

			exits[0] = new Staircase(k, 300);
		}
		else
		{


			//This method creates completely variable distances for up to 3 pairs of staircases. Provided they are arranged symmetrically
			//generally make distanceFromCenter1 > distanceFromCenter2 > distanceFromCenter3
			double distanceFromCenter1 = 160;
			double distanceFromCenter2 = 100;
			double distanceFromCenter3 = 50;
			for(int i = 0; i < (numberStaircases/2); i++)
			{
				exits[i] = new Staircase(k, p/2 - d);
			}
			for(int i = (numberStaircases/2); i < numberStaircases; i++)
```

```
                {
                        exits[i] = new Staircase(k, p/2 +d);
                }
                if(numberStaircases % 2 == 1)
                {
                        exits[numberStaircases - 1] = new Staircase(k, p/2);
                }
        }


        //THIS BEGINS DEFINING THE PEOPLE ON THE TRAIN

                //This gets the number of passengers who would be riding in one train with n cars, where each car was
length d
                int numberOfPassengers = getNumberOfPassengers(n, d);

                //If there were two trains, this adds to the previous total number of passengers how many people you'd
expect to be on the second train
                if(numberOfTrains == 2)
                {
                        numberOfPassengers = getNumberOfPassengers(n2, d2) + numberOfPassengers;
                }

                //This creates an array of object "person" whose length is the total number of passengers determined above
                person[] allPassengers = new person[numberOfPassengers];


                //Based upon common train measurements and specifications, the length of the interior of a train can be
translated into the numberOfRowsPerTrain
                int numberOfRowsPerTrain = (int)(d - 2);

        //The variables above do not need to be altered so they do not need to be run again

    //this for loop runs the entire simulation 20 times so that data can be collected quickly and easily
    for(int xyz = 0; xyz < 20; xyz++)
    {

        //Will have a counter to assign values into the matrix
        int counter = 0;
        //Using 2 sets of 3 nested for loops, the entire array of passengers can easily be filled with unique passengers with their
own specific locations for both trains
        //goes through each train car, placing people in seats
        for(int trainCar = 1; trainCar <= n; trainCar++)
        {
                //goes through putting people in each and every row
                for(int row = 1; row <= numberOfRowsPerTrain; row++)
                {
                        //places people in all of the columns 1-5 for each row
                        for(int column = 1; column <= 5; column++)
                        {
                                //adds all these people into the array. Is noted that they are in the first train
                                allPassengers[counter] = new person(row, column, trainCar, n, q, d, p, exits);
                                counter++;
                        }
                }
        }
```

```
		//if there are two trains, more people must be added to the array of passengers
		if(numberOfTrains == 2)
		{
			//you begin placing people after all of the previous passengers of the first train
			counter = getNumberOfPassengers(n, d);

			//based upon the dimensions of this second train the same method as above is utilized to place passengers into specific locations
			numberOfRowsPerTrain = (int)(d2 -2);
			//goes through each and every train car adding people
			for(int trainCar = 1; trainCar < n2 + 1; trainCar++)
			{
				//uses the dimensions of this train to determine how many rows are in car, and places 5 people in each of these
				for(int row = 1; row < numberOfRowsPerTrain + 1; row++)
				{
					//for each train the number of columns is the same, so this isn't altered
					for(int column = 1; column < 6; column++)
					{
						//adds these people to the array noting they are in the second train
						allPassengers[counter] = new person(row, column, trainCar, n2, q, d2, p, exits);
						counter++;
					}
				}
			}
		}

		//now that all of the passengers have been defined, the staircase that they use can be determined by using arrayLists due to their "elastic" properties
		//(meaning their size is not defined)

		//create a tempArrayList that can be varied in order to be cued as is needed
		ArrayList<person> tempArrayList = new ArrayList<person>();

		//Goes through and does this for each of the staircases
		for(int i = 0; i < exits.length; i++)
		{
			//goes through each and every passenger so that their staircase used can be determined
			for(int f = 0; f< counter; f++)
			{
				//using a method defined in person, the staircase used is determined, and compared to the staircase from the first for loop
				if(allPassengers[f].getExitUsed()== i)
				{
					//if they are part of this exit, they are added to the arrayList
					tempArrayList.add(allPassengers[f]);
				}
			}
			//A method defined below is used to sort the tempArrayList of object person by the time that it takes each person to arrive at the staircase.
			//This creates a cue of people, where each has an order "in line" to begin walking up the stairs
			tempArrayList = sortByTime(tempArrayList);

			//defines some variables that will be needed later
			double waitTime, tstairs;
```

```
//this is the number of columns that can walk next to one another at a time (k)
int widthCapacity = exits[i].getCapacity();

//this is the number of people who can be on the stairs at one time
//implements the assumption that each person will give the person in front of them a buffer space of 1 step.
//if the capacity is not a whole number, the "floor" is used
int totalCapacity = widthCapacity * q/2;

//this value is the time that it takes for a person to go up each stair
double tPerStair = .565611;

//this value is the time that it takes for each person to walk from stair 1 to the street
//is simply the total stairs multiplied by the time that it takes to go up each stair
tstairs = q * tPerStair;

//an array is defined which has a size that is the same as the number of people going to this staircase
double[] timeStartedWalkingUpStairs = new double[tempArrayList.size()];

int counter1 = 0;
double compareTime;

//goes through each person in the cue to go up the stairs in the order that they arrived to the staircase
for(int t = 0; t < tempArrayList.size(); t++)
{
	//compareTime is the time after the train stopped that each person reaches the stairs
	compareTime = tempArrayList.get(t).getTotalTimeToReachStairs();

	//if they are among the first few people to get there, they will have no wait time
	//this if statement is necessary to avoid an "ArrayIndexOutOfBoundsException"

	if(t <= widthCapacity)
	{
		//sets their stair value to just the time it takes to get from stair 1 to street level, in other
words they
		//have a smooth walking line from train door to street
		//this is because they will not have to wait
		//defines this time in the person method
		tempArrayList.get(t).setStairTime(tstairs);

		//adds this person to the number of people on the stairs
		timeStartedWalkingUpStairs[t] = tempArrayList.get(t).getTotalTimeToReachStairs();
	}
	else if(compareTime > timeStartedWalkingUpStairs[t - widthCapacity] + tPerStair*2)
	{
		//Since the array timeStartedWalkingUpStairs is put in order, if the person who is the
number of possible columns of walkers in front of you arrived
		//early enough before you that they already have walked up two stairs then there will
be an open lane and you may begin walking without waiting


		//sets their stair value to just the time it takes to get from stair 1 to street level, in other
words they
		//have a smooth walking line from train door to street
		//defines the stairTime in the person method as this
		tempArrayList.get(t).setStairTime(tstairs);

		//adds this person to the number of people on the stairs
```

```
                                        timeStartedWalkingUpStairs[t] = tempArrayList.get(t).getTotalTimeToReachStairs();
                                    }
                                    else
                                    {
                                        //If the person who is the widthCapacity in front of you in the array has not yet begun
walking when you arrive at the stairs you will not be able to begin
                                        //walking until they have gotten up two stairs. This is due to the ordered nature of the
array and the assumption that there is no skipping in line

                                        //thus, your wait time is the time that the person widthCapacity in front of you began
walking + how long it takes them to get up two stairs
                                        waitTime = timeStartedWalkingUpStairs[t - widthCapacity] + tPerStair*2 -
compareTime;

                                        //the time you started walking up the stairs is added to this array so that ensuing people
can use your value to determine how long they must wait
                                        timeStartedWalkingUpStairs[t] = timeStartedWalkingUpStairs[t - widthCapacity] +
tPerStair*2;

                                        //defines the stairTime of the method as the wait time + the walk time
                                        tempArrayList.get(t).setStairTime(tstairs + waitTime);
                                    }
                                }
                            }

                            //this calculates the average time for all of the passengers from the "seat to the street"
                            double addThis = getAverageTimeForWholeTrain(allPassengers);
                            sumOfTimes = sumOfTimes + addThis;
                            System.out.println(addThis);
                            }
                    //prints the average time
                            System.out.println("average is");
                            System.out.println(sumOfTimes/20);
                            //prints out the model expectation without the scaling constant so that this constant can be experimentally determined
                            System.out.println("model expectation");
                            System.out.println(modelValue(n, d, p, q, k) *  0.0000000058324);
}

                    //gets the total trip time for each passenger and sums them all up to get, in seconds, the total amount of time spent in the station by
all passengers
                    public static double getTotalTimeForWholeTrain(person[] allPassengers)
                    {
                            double total = 0;
                            for(int j = 0; j < allPassengers.length; j++)
                            {
                                    total = total + allPassengers[j].getTotalTime();
                            }
                            return total;
                    }
                    //takes sum of time for whole train and divides it by the the number of passengers to get the average time in seconds to get from
"seat to street"
                    public static double getAverageTimeForWholeTrain(person[] allPassengers)
                    {
                            return getTotalTimeForWholeTrain(allPassengers) / allPassengers.length;
                    }
                    /*Using the parameters n and d, the number of train cars and length of train cars respectively, the total number of passengers can
be determined
```

```
	 *This number can be determined based on known measurements of existing train car interiors.*/
	public static int getNumberOfPassengers(int n, double d)
	{
		int passengersPerCar = 5 * (int)(d -2);
		return n * passengersPerCar;
	}
	//this method is able to sort the arrayList of people going to each specific staircase by the time it takes them to get there
	//in an ascending order. Is a variation upon bubble sort
	public static ArrayList<person> sortByTime(ArrayList<person> arrayToSort)
	{
		//goes through the whole array beginning from the end going downwards
		for(int g = arrayToSort.size()-1; g>0; g--)
		{
			//goes through the rest of the arraylist from start until one less than g
			for(int h = 0; h<g; h++)
			{
				double time1 = arrayToSort.get(g).getTotalTimeToReachStairs();
				double time2 = arrayToSort.get(h).getTotalTimeToReachStairs();
				person tempPerson = arrayToSort.get(g);

				//if the time2 is longer than time1, the two switch order
				if(time2>time1)
				{
					arrayToSort.set(g, arrayToSort.get(h));
					arrayToSort.set(h, tempPerson);
				}
			}
		}
		//the ascending sorted array is returned
		return arrayToSort;
	}
	//gets the expected time based upon the variables defined by trials using the simulation
	public static double modelValue(int n, double d, double p, int q, int k)
	{
		//experimental equation of t vs. n
		double nEq = (36.095 * n) + 4.7382;
		//experimental equation of t vs. d
		double dEq = (26.043 * d) + 19.536;
		//experimental equation of t vs. q
		double qEq = (.5485 * q) + 515.67;
		//experimental equation of t vs. k
		double kEq = (Math.pow(k, -.529)) * 813.04;

		//multiply all of them together and can use this value compared to the simulated average times to determine coefficient
of empirical equation
		return kEq * qEq * dEq * nEq;
	}
}
```

```
import java.util.Random;
public class person
{
	//here all the variables in which the parameters will be stored are defined
	private int row, column, car, n, q, theirExit;
	private double walkSpeed, d, p;
	private double totalTimeToStreet;
	private boolean frontExit;
	private Staircase[] exits;
	private double stairTime = 0;

	//this constructor passes in all of the applicable information that defines the actions of the person and saves these parameters to the variables previously initialized
	public person(int row1,int column1,int car1, int n1, int q1, double d1, double p1, Staircase[] exits1)
	{
		row = row1;
		column = column1;
		car = car1;
		n = n1;
		d = d1;
		p = p1;
		q = q1;
		exits = exits1;
	}

	//The following methods define the parameters of the person, and can return specific values

	//This returns the row within the car, where the furthest "row" to the front is 1
	public int getRow()
	{
		return row;
	}
	//This returns the specific seat within the car where columns 1&2 are the 2-seat side, & 3-5 are the 3-seat side
	public int getColumn()
	{
		return column;
	}
	//This returns the "car number" where the cars are numbered from 1 - n where 1 is the front car.
	public int getCar()
	{
		return car;
	}
	/*Using assumed values for walking speed and standard deviation, a double which is walking speed in m/s is returned.
	 *This velocity is determined based upon a normal distribution, where the mean is what is assumed.
	 */
	public double getWalkingSpeed()
	{
		//imports the random class from java, because each person will be assigned a random walking speed
		Random r = new Random();
		//stanDev is the assumed value for standard deviation of walking speed in m/s
		double stanDev = .1705;
		//meanSpeed is the assumed value for the average walking speed of a commuter in m/s
		double meanSpeed = 1.3005;
		/*The function "nextGaussian" gives pseudo-random floating values following a normal/Gaussian distribution
		 *These values have a mean of 0 and a standard deviation of 1. To use this method to approximate the distribution
		 *yielded by our assumptions we are able to multiply "nextGaussian" by stanDev and add meanSpeed
		 */
```

```
		walkSpeed = r.nextGaussian()*stanDev + meanSpeed;
		return walkSpeed;
	}

	//The following methods can be used together to determine the time in seconds that it takes to exit the train car

	/*This method returns the time that it takes for a person to exit the train car from the time when the car stops in the station.
	 *This is a function of their seat location, based on previously outlined assumptions about exit formations and procedures.
	 *Each person will be walking at the same speed as all others, as there is so little room for variance in walking speed.
	 */
	public double getTimeToLeaveTrainCar()
	{
		return getTimeBeforeWalkDownAisle() + getTrainWalkTime();
	}
	/*Determines the time before a passenger can begin to walk down the aisle and exit the train car. This is defined by their location
	 *within the train car. The pattern which they exit in is outlined in a diagram.
	 */
	public double getTimeBeforeWalkDownAisle()
	{
		int peopleBefore;
		double timeBefore;
		int indivWaitTime = 2; //variable for time it takes in between people leaving
		if(isFrontExit())
		{
			//the people who will exit before you can be calculated based upon your seat row and column
			peopleBefore = (row-1)*5+column-1;

			//by multiplying how long it takes for each person to start to leave by the number of people in front of a
person
			//the time it takes before they can begin walking can be calculated
			timeBefore = peopleBefore*indivWaitTime+2; //added two seconds for time for initial person to realize
train has stopped
		}
		else
		{
			//if the person will  be going out of the back exit, a different equation yields the number of people who leave
before them
			peopleBefore = (((int)d)-2-row)*5+column-1;

			//can use this to yield total wait time
			timeBefore = peopleBefore*indivWaitTime+2; //added two seconds for time for initial person to realize
train has stopped
		}
		return timeBefore;
	}

	/*Using information upon the location of the passenger's seat and the length of the train, the time that is spent walking
	 *down the aisle can be determined based upon assumptions about walking speed and interior configurations of the train.
	 */
	public double getTrainWalkTime()
	{
		//Due to the tight quarters of a train aisle, the walking speed can be expected to have decreased by 25% for all.
		//All people will walk at the same speed due to peer pressure to get out of the way
		double trainSpeedWalk = .975;

		double distanceWalked, timeWalked;
		//this is able to, based upon your location in the train determine how far you must walk before exiting the train car
```

```
		if(isFrontExit())
		{
			//if you are leaving out of the front exit this will be the distance walked. See diagram for exact path
			distanceWalked = row - 1 + .33 + .5 + .27 + .27 + 2*(.53);

			//to get the time this walk took, the distance can be divided by the velocity of the walker
			timeWalked = distanceWalked / trainSpeedWalk;
		}
		else
		{
			//These methods are essentially the same as for the front exit, however the distance is slightly different.
			//For more information on the distance see the diagram for the exact path.
			distanceWalked = (int)(d-2) - row + .67 + .5 + .27 + .27 + 2*(.53);
			timeWalked = distanceWalked / trainSpeedWalk;
		}
		//the total time that it takes to walk from the aisle to the door of the train can be returned
		return timeWalked;
	}
	//Based on the row the specific passenger is in, which exit they take can be calculated by proximity
	public boolean isFrontExit()
	{
		//if they are in the second half of the train, or in the middle aisle, they will leave out of the back exit
		if(row > ((int)(d-2)) /2)
		{
			return false;
		}
		else
		{
			//otherwise it is true that they will leave out the front exit
			return true;
		}
	}

	//The following methods can be used together to determine both which exit each person will go to and the time that walk takes.

	//This method can get the "x-value" or distance from the "front" of the platform for each person based upon which exit they use
	public double xValueOfExitUsed()
	{
		//The double chain is the length in meters of the train coupling or the distance between the individual train cars
		double chain = 1;

		//The double wallWidth is the width in meters of the wall at the ends of each train.
		double wallWidth = .3;

		//The distance from the front end of the platform can be calculated based on which exit is used

		//the xValue of the front exit is half of the difference of between the length of the platform and train. This is added to
the width of the wall
		//based upon which car they are in, the distance between front exits is multiplied.
		double xValue = ((p - (n*d) + (chain * (n -1)) + (2 * n * wallWidth))/2) + (n -1)*(d + chain + (2 *wallWidth)) +
wallWidth;
		if(!isFrontExit())
		{
			//if they are leaving from the back exit, they have the xValue of the front exit + the distance of the car
			xValue = xValue + d;
		}
		return xValue;
```

```
	}
	//Determines based upon which exit they use, which staircase is the closest, and the distance to this staircase
	public double xDistanceToTheirStaircase()
	{
		//it is established where upon the "number line" that is the platform the person begins
		double xValue = xValueOfExitUsed();

		//This variable will be a placeholder used to determine which exit is closest. It will be replaced whenever a closer exit is discovered.
		//It begins at p, because that is further than the farthest possible length between an exit and a staircase
		double changer = p;

		//this is also a placeholder that will improve efficiency and simplicity rather than recalculate difference each time
		double difference;

		//goes through the array of all of the exits
		for(int i = 0; i < exits.length; i++)
		{
			//calculates the distance along the x-axis from your exit to the specific staircase
			difference = xValue - exits[i].getPos();

			//if it is positive and closer than the previous closest difference, it replaces the previous distance
			if(difference < changer && difference > 0)
			{
				changer = difference;
			}
			else if(difference * -1 < changer && difference < 0)
			{
				//if it is negative and closer, it is made positive and replaces the previous distance
				changer = difference * -1;
			}
		}
		//the shortest distance is then returned
		return changer;
	}

	//determines the total time that it takes to walk from the train to the staircase
	public double getPlatformWalkTime()
	{
		//this changeable variable is an assumption for the width of the platform
		double platformWidth = 5;

		//using distances we know we are able to assume that each person walks in a straight line, and thus get their distance travelled by using the pythagorean theorem
		double totalDistance = (platformWidth / 2)*(platformWidth / 2) + (xDistanceToTheirStaircase())*(xDistanceToTheirStaircase());
		totalDistance = Math.sqrt(totalDistance);

		//by dividing this distance by their walking speed the total time that each person spends walking on the platform can be calculated
		return totalDistance / getWalkingSpeed();
	}

	//returns the specific exit used by a person
	public int getExitUsed()
	{
		double xValue = xValueOfExitUsed();
```

```
		//This variable will be a placeholder used to determine which exit is closest. It will be replaced whenever a closer exit is discovered.
		//It begins at p, because that is further than the farthest possible length between an exit and a staircase
		double changer = p;
		double difference;
		int theirExit1 = 0;

		//goes through all of the exits
		for(int i = 0; i < exits.length; i++)
		{
			//calculates the distance
			difference = xValue - exits[i].getPos();

			//keeps track of which exit is the closest
			if(difference < changer && difference > 0)
			{
				changer = difference;
				theirExit1 = i;
			}
			else if(difference * -1 < changer && difference < 0)
			{
				changer = difference * -1;
				theirExit1 = i;
			}
		}
		//ends by returning the closest exit
		return theirExit1;
	}

	//This method can determine the total time it takes up until getting to the stairs in order to determine the order in which people walk up the stairs
	public double getTotalTimeToReachStairs()
	{
		//sums up the time to get out of the train car and how long it takes to get to the stairs
		return getTimeToLeaveTrainCar() + getPlatformWalkTime();
	}
	//can easily return the total time it takes to reach the stairs
	public double getStairTime()
	{
		return stairTime;
	}
	//allows the "stair time" to be set from a main method
	public void setStairTime(double stairTime1)
	{
		stairTime = stairTime1;
	}

	//The following method combines all of the individual times that it takes to get the total time for this person to get to the street

	/*The total time to get to the street where the clock starts when the train stops can be expressed as the sum of
	 *the time it takes to leave the train car, the time it takes to walk down the platform, and the time to walk upstairs.
	 */
	public double getTotalTime()
	{
		return getTotalTimeToReachStairs() + getStairTime();
	}
}
```

public class Staircase

```
{
	//the only two characteristics of a staircase are its position and capacity
	private double pos;
	private int capacity;

	//if a staircase needs to be put in a random location, an arbitrary integer is added to the constructor
	public Staircase (double p, int y, int f)
	{
		//randomly positioned somewhere from 0 to p
		pos = Math.random()*p;
		//defines the capacity
		capacity = y;
	}
	//more commonly used constructor, both position and capacity are passed in
	public Staircase(int y, double pos1)
	{
		//define these two variables
		pos = pos1;
		capacity = y;
	}
	//returns the position of this staircase along the platform. Where 0 is the frontmost end of the platform
	public double getPos()
	{
		return pos;
	}
	//returns the saved capacity
	public int getCapacity()
	{
		return capacity;
	}
}
```