For office use only

T1 ____________________

T2 ____________________

T3 ____________________

T4 ____________________

Team Control Number

2022

For office use only

F1 ____________________

F2 ____________________

F3 ____________________

F4 ____________________

2022

The International Mathematical Modeling Challenge (IM^{2}C) Summary Sheet

(Your team's summary should be included as the first page of your electronic submission.)

The delay of time, the most crucial factors in air transportation, always have a huge impact on the flight schedule, not only on the delayed flight, but it also affects on the overall timeline of the flight and widely affect to both passengers and air carriers, even though it is a really short period. The cause of time delay usually comes from the out-of-control process. Among the process of the flight, boarding and disembarking are the two procedures that require lots of time if there is no effective boarding method for that scenario.

In this report, the accounted problem will be solved by mathematical model and Monte Carlo method. Each type of method will be investigated to find the most effective method to queue the passengers to achieve the shortest boarding time. The mathematical model is constructed by the conditions and factors that will affect the boarding time such as the number of carry-on luggage, number of people in the flight. In some cases, the amount of people who do not follow the strategy also be included in this model. Subsequently, the Monte Carlo simulation is applied to simulate the data and conditions of the model to obtain the boarding time in each method. Additionally, greedy algorithm is introduced to determine the optimal passengers' sequence in the disembarking method, which can be proved that the greedy solution is feasible and correct.

Regarding this model and simulation, we found that the most effective way from the three provided boarding method; random method, section method, and WilMa/seat method is WilMa/seat method with around 14 minutes of boarding time (13 minutes in the ideal case). But there are also two new methods that the authors had recommended which are reverse pyramid method and WilMa/seat section method. In these five methods, it results in reverse pyramid method to be the most effective method with approximately 17 minutes of boarding time (14 minutes in the ideal case). Moreover, we also investigate the most effective method on the "Flying-Wing" plane and the "Two-Entrance, Two-Aisle" plane as well as considering the capacitance limitation when there is pandemic occurring.

Contents

1 Introduction

1.1 Introduction

In the airline industry, time is a crucial part to success. As the saying goes "Time is money". Airline industries tries to optimize each process time to gain the most profit by getting the most hour of flight per day and lose less on turn-around time, in other word, they aim to create the most effective method in each process. One of the most important process in the air line industry is the boarding and disembarking or leaving the aircraft. The time taken for the boarding/disembarking process are influenced by many factors. Aircraft typically has small aisle with the width that only 1 person could get through, meaning that if one person stop to store their luggage on the overhead bin, the line would come to a halt. Similar situation also occur when a person has to get up to make way for another person with the inner seat, for example, the passenger with the aisle seat has to get up and block the line of passengers to make way for the people with the middle seat and the window seats. Many prescribed methods for boarding and disembarking the aircraft are proposed but which one is the most effective time-wise.

In this paper, we will investigate the efficiency of each prescribed method in a common "Narrow-Body" passenger aircraft including the random method, the section method, and the seat method including new methods proposed by the authors using a mathematical model and Monte Carlo simulation considering the average carry-on luggage per person and the percentage of people not complying to the method. Comparing and analysing the time consumed by each method, the most effective method is proposed. The most effective method of other types of aircraft including the "Flying Wing" and the "Two-Entrance, Two Aisle" passenger aircraft is also proposed.

1.2 Problem Restatement

We were tasked with creating a plane boarding and disembarking method the most effective in real practice. The problem was separated into 5 parts:

1. Construct a mathematical model calculating time boarding and disembarking the airplane. The mathematical model should consider carry-on luggage, the number of passenger not following the prescribed boarding/disembarking method, as well as being able to apply to prescribed method of boarding and disembarking the aircraft.
2. Apply the mathematical model to the standard "narrow-body" aircraft illustrated in Figure 1.
    - a Compare the average time, the practical maximum (95^{th} percentile), and the practical minimum (5^{th} percentile) of the prescribed method
        - Random boarding
        - Boarding by section
        - Boarding by seat
    - b Analyze how the results varied with the percentage of people not following the prescribe method and the average number of carry-on luggage. And which method is the most effective.
    - c Consider the situation where the passengers carry more luggage than normal and stow all their carry-on luggage in the overhead bins. How does it affect the results?
    - d Describe two additional possible boarding methods. Explain and justify your recommended optimal boarding method.
    - e Explain and justify your optimal disembarking method.
3. Modify your model for the 2 type of airplane, "Flying Wing" type in Figure 2 and "Two-Entrance, Two Aisle" Figure 3, and recommend the optimal boarding and disembarking method for each types.
4. Due to the pandemics, capacity limitations can be applied. Will your optimal prescribed boarding/disembarking method changes at the capacity limitation of 70 %, 50%, and 30% of the available seat.
5. Write a 1-page mail to an airline executive discussing your results, recommendation, and rationale about the boarding/disembarking method in a non-mathematical way.

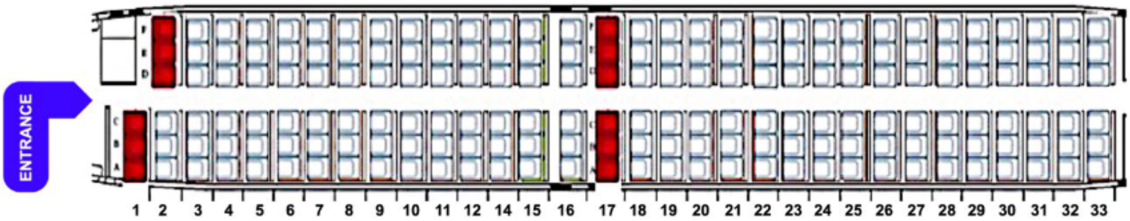


Figure 1: "Narrow-Body" Passenger Aircraft

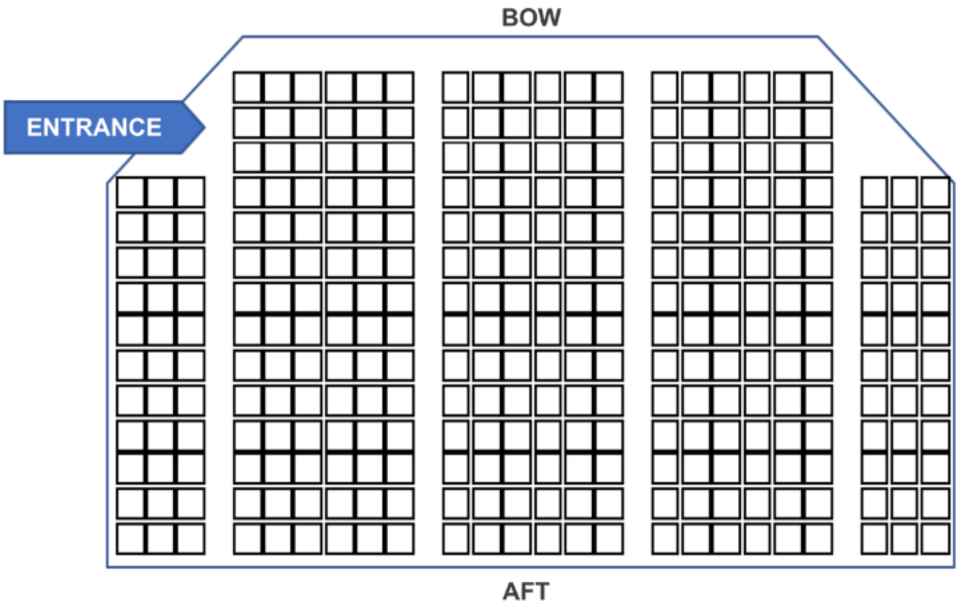


Figure 2: "Flying Wing" Passenger Aircraft

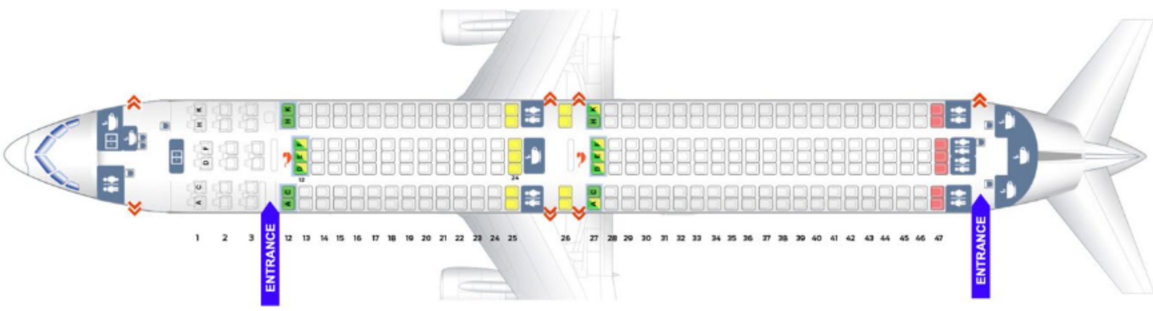


Figure 3: "Two-Entrance, Two Aisle" Passenger Aircraft

2 Definitions and Assumptions

2.1 Definitions of Important Terms

- Monte Carlo Simulation

  The mathematical and computational method for simulating and estimating the possible outcomes under the conditions with the stochastic variables.

- Boarding Method

  The method of generating the seat position for each passenger from passenger number 1 to n.

- Disembarking Method

  The method of planning a queue for each passenger to leave the aircraft.

- Greedy Algorithm

  The problem-solving algorithm that selects the current optimal choice and gives the optimal answer at the overall result.

- Interference

  The blockage of the aisle due to a passenger storing their luggage and the passenger waiting for the previously passenger to make way for them.

2.2 Definitions of Variables

Variable	**Definition**
t_{walk}	Total time for walking from each row to the next row
t_{seat}	Total time for each passenger to sit at their seat
t_{store}	Total time for storing carry-on bags
$t_{collecting}$	Total time for collecting carry-on bags from the overhead bin
$t_{boarding}$	Total aircraft boarding time in t_{walk}
$t_{disembarking}$	Total aircraft disembarking time in t_{walk}
$n_{passenger}$	Total number of passengers in the aircraft
n_{bin}	Total number of luggage in the overhead bin
$state_i$	Passenger i's current state
$seatx_i$	Column of passenger i's seat
$seaty_i$	Row of passenger i's seat
$luggage_i$	Number of luggage of passenger i
$speed_i$	Passenger i's celerity
$posx_i$	Passenger i's current column
$posy_i$	Passenger i's current row
ent_i	Passenger i's entranced way (front/back) in two-entrance, two-aisle aircraft

2.3 General Assumptions and Justifications

Assumptions	Justifications
At the beginning, all the seat are vacant and the first passenger is boarding the plane.	This assumption is held to set the scale of the model and to simplify the model.
The aircraft aisles and row seats allow only one passenger to pass at a time.	The aisles and rows are too narrow for more than one person to enter.
The passenger can enter the aircraft row seat if and only if they are in front of that row.	The position of the aisle and the seat allows just one people to enter
Passengers' speed are grouped into 2 groups, slow group and normal group, slow people will take action longer than normal people twice	Due to the variety of the empirical speed of each passenger in , the assumption is set up to simplify the model

3 Task 1: Mathematical model for calculating boarding and disembarking time

3.1 Boarding Model

The model for calculating the time needed in boarding the airplane is used in a simulation. The simulation is a discrete-event model meaning nothing happen between each time step. The model's input will be the queue of passenger and their respective seat number, the number of their carry-on luggage, and their walking speed. The input will be generated by another program based on the prescribed method used. People not following the prescribed method will be simulated by the program called *sequence generator* which we will program in later task.

The Aircraft plan
At the beginning of the simulation ($t = 0$), the aircraft map is created as shown in Figure 4. The green cell represents the seat and the grey cell in the middle is the aisle. The blue cell is the passenger's entrance. The airplane will be split into 2 side: the left side where the column A,B,C are at and the left side where the column D,E,F are at. In 1 row, the 3 passengers each side will share the same overhead cabin. The number of luggage in the bin will be recorded.

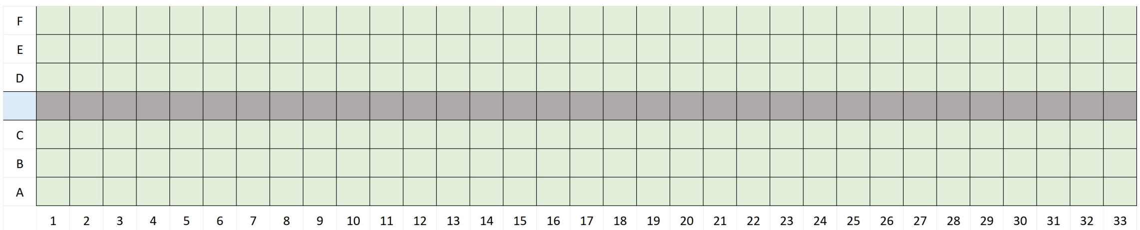


Figure 4: The generated aircraft plan

The Passenger

The passengers are represented in the program with an index from 1 to $n_{passenger}$. The index can be used to get the information such as followed:

- Their current *state* whether they are storing the luggage, walking down the aisle, waiting for the line to move, or seated
- $seatx, seaty$ represent the column (A,B,C,...) and the row (1,2,3,...) of the passenger's seat respectively.
- $posx, posy$ represent the column and the row the passenger is currently standing at at any point in time respectively.

- *speed* represents the type of the passenger whether he is a normal person or a slow person. Normal passenger has $speed = 1$ and slow passenger will has $speed = 2$ (The value of *speed* is not proportional to the real speed.)
- *luggage* is the number of carry-on luggage the passenger has ranging from 0 to 4.
- *ent* is represented the passenger's entrance way.

These are the conditions all passenger will obey upon walking down the aisle:

- All passenger will proceed down the aisle in the condition that each passenger cannot pass the aisle while there is someone blocking them and stop at the row their seat is located.
- If the row is empty, the passenger can immediately take his seat.
- Due to the narrow passage, the line in the aisle can come to a stall due to the interference in front of them. This interference might be due to the passenger in the front stopping to store their carry-on luggage in the overhead bin or the passenger already seated at the row another passenger is trying to get into moving out of the row to make way for the other passenger moving into the seat closer to the window seat than him, for example, if the passenger seating at the middle seat came before the window seat, the passenger already seated must get up and make way for the passenger seated at the window seat.

The Time Simulation

The simulation will calculate the time require for all the passenger to get seated. The simulation will evolve 1 time step at a time. The time require for a normal passenger to move from 1 row to the next is around 2 seconds, so we set out time step to be 2 second per time step. The time for other action will be calculated as a unit of time step. The final total time step will be converted to seconds to calculate the total boarding time. The type of people also affect the boarding time. As the assumption has stated, there will be 2 type of people, the normal type and the slow type.

The time calculated in the simulation includes the time the traveller is traversing down the aisle, the time the passenger uses to store the luggage and the time the passenger move into his seat. The time each passenger uses in traversing from row to row is equal to each passenger's *speed*, the slow passenger having 2 and normal passenger having 1.

The time used in storing luggage for each people is different. In real practice, there are group of people who is slower than normal as we have assume and they take double the time normal people need to do such action as we have established. The factor affecting the luggage storing time include: the speed of the passenger, the number of luggage the passenger has, and the number of luggage already in the overhead bin. The number of luggage already in the overhead bin affect the time since the passenger has to make room and re-position the already existing luggage in the overhead bin to properly store the luggage. The equation for calculating the storing time from [7] is

$$t_{store} = ((n_{bin} + luggage_i) \cdot luggage_i/2) \cdot speed_i \tag{1}$$

t_{store} is the time to store the luggage
n_{bin} is the number of bag already in the overhead cabin
$luggage_i$ is the number of luggage the passenger i has
$speed_i$ is the time for the passenger i to walk from 1 row to the next

When the passenger walk to the row their seat is located, their state value switch from "In Aisle" to "Storing". This causes interference in the aisle and the passengers behind this passenger has to stop.

The next interference is the row interference. Some seat is unreachable if the seat before the it is occupied, for example, the passenger in the middle seat needs to get up in order for the passenger in the window seat to reach his seat. The model consider this fact and calculate the time consumed due to this interference. The model first check if the row is vacant or if the passenger can immediately go sit in the seat. The time used in getting into and out of the seat is equal to t_{seat} which is the equivalent of 4 time steps (8 seconds). If the row is blocked and interference occurred, the condition is spread into 2 conditions

1. If only 1 person needs to get up and make way for the passenger to enter, the interference will be resolved in 3 t_{seat}, one for the already seated passenger to get out, one for the waiting passenger to get in, and finally 1 for the passenger who got up to make way for the other passenger to take a seat.

2. If the window seat passenger arrived with the aisle and middle seat already seated, the interference will be resolved in 5 t_{seat}. First, the 2 passenger seated must get up and make way for the window seat, consuming 2 t_{seat}. Next, the window seat can proceed to his seat and the 2 can follow, consuming 3 t_{seat}

That concludes the mathematical model in the boarding procedure. The final time step will be calculated by converting the time step into second by multiplying 2. The summary for the time each action will take is listed in Table 1

Action	**Number of time step consumed for normal people**
Walking from 1 row to the next	1
Taking a seat or getting out of the seat	4
Luggage storing	Equation 1

Table 1: The time used in each action of boarding the aircraft

3.2 Disembarking model

The disembarking method is similar to the boarding method in terms of calculating time and simulating the scenario. We created a simulation to simulate the passenger disembarking the aircraft. The model's inputs are the same as boarding model which are the queue of leaving, the numbers of luggage, and speed of the passenger. The condition on interference and walking down the aisle is the same. All passenger will get up from the aisle seat and walk toward the exit. The passenger will get up onto the aisle if the aisle next to their seat. In case there are 2 passengers both the aisle seat in the same row, the passenger with the higher priority in the queue will get up first. If the seat next to any passenger in the direction of the aisle seat is vacant, the passenger will move to that seat to get closer to the aisle. The time for getting your luggage out of the overhead bin is the same as Equation 1. The simulation ends when all passenger left the aircraft.

4 Task 2: "Narrow-Body" Passenger Aircraft

4.1 Task 2a: The different type of boarding method

The sequence generator purpose is to generate the queue or sequence of passenger entering the aircraft. The sequence generator will designate each passenger in the queue with the seat the passenger is in, the passenger's speed, and the number of luggage each passenger has. These output is an array that will be used as input for the time calculation model. Each method require the modification of the sequence generator for each prescribed method.

4.1.1 Random Method

Random Boarding Method

The random boarding is as the name suggest, boarding without any plan or specific method. In this method, there will not be any people not following the prescribed method since every boarding method can be called a random method.

The Random Method Sequence generator

The sequence generator generates the random queuing sequence for the standard body airplane by following the algorithm listed below:

1. First we assigned the row to all passenger in the queue, an array created by the sequence generator. The i^{th} queue will receive $seat_y$ by the equation $seat_y = \lfloor i/C \rfloor + 1$ where C is the total column in the plane.

2. We will assign the $seat_x$ value by $seat_x = i \ mod \ C$. The value of $seat_x$ from 1 to 5 corresponds to A to E respectively and 0 represents column F.
3. The queue in the array are then shuffle randomly
4. The luggage proportion and the passenger's chance of being a slow person is input by the user and the number of luggage and the speed of each person is randomly assigned to each passenger.

4.1.2 Section Method

Section Boarding Method

The section method refers to the boarding method where the passenger is separated into section of the plane, the aft, the middle and the bow section. There are different section method, the most commonly used are the back-to-front and the front-to-back method. This method can reduce some of the interference due to the luggage storing process. The queue generator would first separate the passenger into group by their section and then randomize the queue in each group. As long as there are groups of people belonging to the same section of the plane close together, the method will be considered as the section method.

The Section Method Sequence generator

The sequence generator for the section method is programmed according to the algorithm listed below:

1. All passenger in the queue is given $seat_y$ value with an additional value sec to determine their section. The rows are separated into 3 equal sections: the front, the middle, and the back.
2. $seat_x$ value is assigned in the same algorithm as the random sequence generator.
3. We group the same section together and shuffle the people within the same section.
4. The groups are then ordered in the sequence corresponding to the section method currently used.
5. The luggage proportion and the passenger's chance of being a slow person is input by the user and the number of luggage and the speed of each person is randomly assigned to each passenger.

For analyzing the different the effect of the order of boarding the plane by section, the sequence generator generate 6 type of the section method, all 6 of the possible permutation of the 3 section. The name of 6 method would be called using the letter F, M,B which represent front, middle, and back respectively to indicate the order of the section boarding. BMF method means the order of boarding is back, middle, and front.

4.1.3 Seat/WilMA Method

Seat/WilMA Boarding Method

The seat method, nicked name the WilMA (Window-Middle-and-Aisle seat boarding method), refers to the boarding method which separate the passenger on the plane according to their seat type: the window seat first, the middle seat next, and the aisle seat last. This method will reduce the occurrence of the row interference significantly. The sequence generator will create a queue for the input of the time calculating model by similar method as the section method, but this time, instead of using the section, we separate the passenger based on their seat type.

The WilMA Method Sequence generator

The sequence generator for the WilMA method create a sequence and group the people in each type of seat together and arrange them in this order: the window seat (seat A,F), the middle seat (seat B,E), and the aisle seat (seat C,D). The sequence generator algorithm is listed below:

1. The $seat_x$ is assigned to all passenger, starting with the column of window seats ,F and A respectively, to the first $n_{passenger}/3$ passengers, then provide the column of middle seats, E and B respectively, to the next $n_{passenger}/3$ passengers, lastly, assign the column of aisle seats, D and C respectively, to the last $n_{passenger}/3$ passengers. In each type of seats, the additional value sec is used to determine each passenger's section.

2. We account $seat_y$ of the passengers in each column by the equation $seat_y = i \ mod \ C + 1$ where C is the total column in the plane.
3. The passenger in the same type of seats are shuffled randomly.
4. The luggage proportion and the passenger's chance of being a slow person is input by the user and the number of luggage and the speed of each person is randomly assigned to each passenger.

4.1.4 The Time Calculation Process

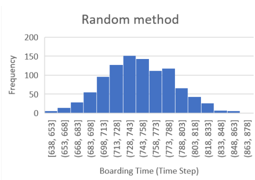
The time for each method is calculated using the sequence generator and the time simulation model. The process is split up into 2 parts: the queue generation process and the time calculation method. The sequence generation creates the input for the time model. For the average, the 5^{th} and the 95^{th} percentile, we use the ***Monte Carlo Method***, using the simulated result from 1000 loops, each loop having different permutation of the queue in each group, e.g., the queue of the front section in the section method can be shuffled. In this sub-task, we will assume that the passenger all follow the prescribed method and none is carrying any carry-on luggage. Those 2 variable will be further analyze in the next sub-task. All passenger has a 20% chance of being a slow person.

4.1.5 Result and Discussion

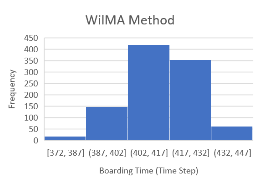
The random, WilMA, and the different seat configuration are shown in Figure 5 and Table 2 shows the different method's time, 5^{th} percentile, 95^{th} percentile and the sample's standard deviation of the boarding time. The airplane used is 33 rows long, and 6 columns wide as illustrated in Figure 4

Method	Average Time (s)	5^{th} percentile	95^{th} percentile	Standard Deviation
Random	1495.072	1368	1628	79.01016
WilMA	828.622	788.1	868	23.77929
FMB	1872.608	1706	2012	92.58584
FBM	1744.462	1612	1902	96.99599
MFB	1740.766	1602	1874	88.72055
MBF	1786.416	1658	1932	86.37991
BFM	1670.974	1534	1832	90.62156
BMF	1590.016	1468	1728	76.27535

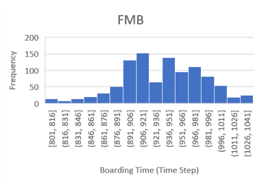
Table 2: The statistical value of each method



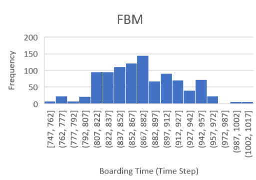
(a) Random method



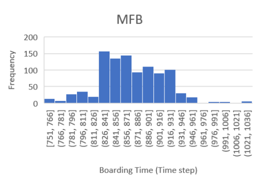
(b) WilMA method



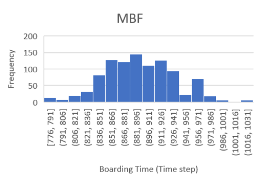
(c) FMB method



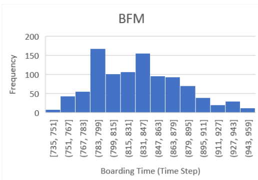
(d) FBM method



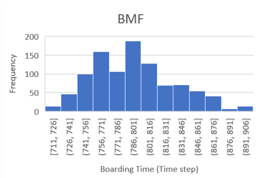
(e) MFB method



(f) MBF method



(g) BFM method



(h) BMF method

Figure 5: Histogram of boarding time of different method

All the histogram resemble the bell-shaped distribution. The method with the lowest average time is the WilMA method. This is to be expected since the model have no row interference, the most time consuming method of all interference. All the section method, however, consumed more time than the random method.

This might indicate the ineffectiveness of the section method. All the method except the WilMA method has high SD, especially the section method. The practical minimum and the practical maximum of the WilMA method is also the minimum. In the case of everyone abiding the boarding method and no carry-on luggage, the WilMA is the most optimal method of the 3.

4.2 Task 2b: Analyzing the Model: Sensitivity

4.2.1 Procedure

A simple sensitivity analysis will be done on each boarding method. The two variable we'll be testing is the percentage of people not complying with the boarding procedure and the average luggage of the passenger.

The Passenger not following the boarding method

The passenger not complying with the boarding method is defined as the passenger who is standing in the wrong queue. Each boarding method has its queue whether it be the section boarding where the front section come first or the WilMA method where the window seat come first. The percentage of passenger not complying with the boarding method will only be analyzed on the WilMA method and the BMF method. The random method won't be analyzed on this variable since the method has no designed queue so even if you shuffle the random method up, the queue in the shuffled random method would still be considered random method. We only chose the BMF method or the back-to-front method because out of all the section method, the back-to-front method is the most effective and all the section method has similar standard deviation and the data is distributed evenly. With these similar characteristic, I choose only the most effective section method, the back-to-front method.

The sequence generator created the passenger not following the boarding method by switching people places in the ordered queue created by the sequence generator. The switched pair must come from a different block in the queue. For example, the program would switch the a random person from the front section with a random person from the middle section in the back-to-front method. For each switch, the people who do not followed the boarding method increased by 2. The process continued until the switched passenger reached the input percentage. The percentage analyzed are 20%,40%, and 60%. The average boarding time of 1000 loops are compared between each percentage to evaluate the sensitivity of the 2 methods to the percentage of people not following the method.

The average carry-on luggage

The average carry-on luggage will be analyzed at 3 values: 50, 100, and 150. The 3 boarding methods we are analyzing are the Random method, the back-to-front method, and the WilMA method. The proportions of people who carry 0,1,2,3,and 4 pieces of luggage in each case will be evaluated by poisson distribution. The amount of luggage that each passenger will carry is randomly chosen according to the proportions from poisson distribution. After the simulation, it shows that the random method is the most effective strategy to deal with the larger amount of luggage.

In the process of random assignment of the luggage for each passenger, the sequence generator is used to contribute this process, starting with assigning the seat of the passenger, then shuffle the passenger regarding type of boarding method to make the queue. After that the program will randomly grouped the passengers into 5 groups. The first group that we assigned will be the group of passengers with 0 luggage then 1,2,3,4 luggage respectively, the members in each group are respect to the proportions in each case which are 50,99,149,and 198 average carry-on luggage.

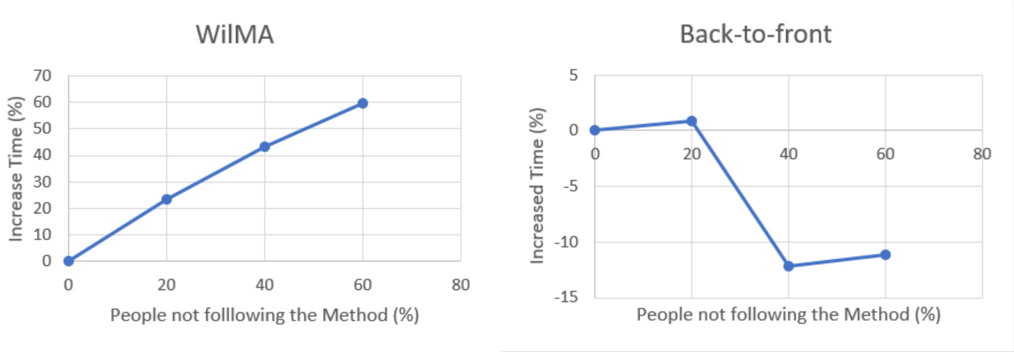
4.2.2 Result and Discussion

Percentage of passenger not following the Boarding Method: Sensitivity Analysis

The line graph representing each boarding method increases in percent at different number of people not following the boarding method is plotted in Figure 6. The 2 models showed different order of sensitivity. For the WilMA method,there is high sensitivity of up to 20 percent at only 20 percentage of people not following the method. This indicates the effectiveness of this method only occur when all the passenger follow the rule. The back-to-front method, however, showed low sensitivity toward people not following the method. The negative change or the decrease in the boarding time might be the queue being more random and the time decreases since the random model average time is lower than the section method.

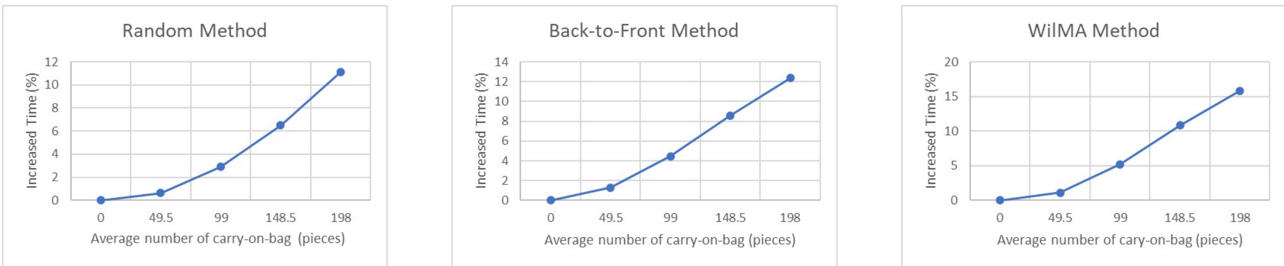
The average carry-on luggage: Sensitivity Analysis

The line graph of each method in Figure:6 shows the relationship between the increased time and average number of carry-on bag. All of them show the graph that has similar characteristic but with different amplitude which means that the sensitivity of the random method which has the lowest amplitude is the lowest.



(a) WilMA Method (b) BMF method

Figure 6: Sensitivity analysis on the percentage of people not following the method



(a) Random Method (b) BMF Method (c) WilMA Method

Figure 7: Sensitivity Analysis on the factors of average number of luggage per flight

The 3 methods showed similar sensitivity toward the average luggage per flight. All 3 methods peaked at approximately 15 percent, this might indicate that the storing luggage process does not significantly affect the boarding time of the 3 methods.

4.2.3 Conclusion

From both sensitivity analysis, we conclude that the WilMA method is the most effective among the 3 methods. Even though the WilMA method has high sensitivity toward the people not following the method, its raw data is still less than both the random method and the back-to-front method. The luggage has similar effect on the boarding time of the 3 methods. So from analyzing the sensitivity, the WilMA is the most effective method.

4.3 Task 2c: More than average number of luggage

Average passenger carry carry-on luggage per person, so the average number of luggage is about 1 to 1.5. In this task scenario, the passengers are carrying more luggage than usual. We will assume that the range of luggage stored in the overhead cabin is 0 to 4 in this scenarios. We will investigate the effect of the number of increased luggage on the 3 boarding methods.

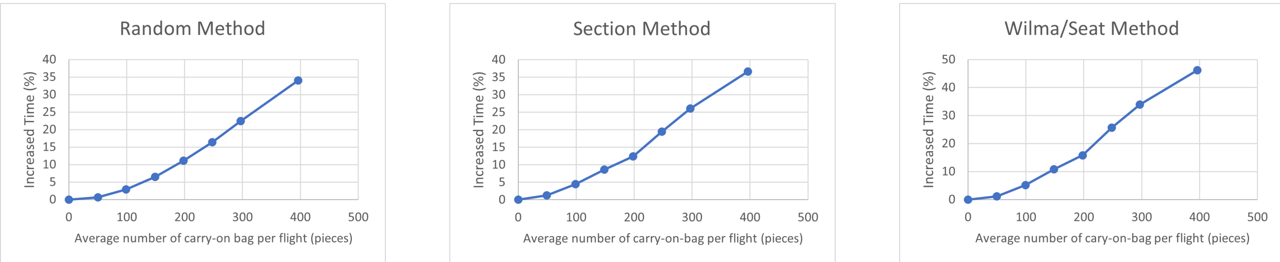
4.3.1 Procedure

We will do the sensitivity analysis on the average luggage number again, only this time, the additional cases of average number of luggage per flight are added which are 248,297,and 396 pieces per flight which we consider to be the case of "more than average number of luggage". There are also new groups of the passengers who carry 5 and 6 carry-on bags; the proportions of each group of passengers are the result from the poisson distribution.

The method of the assignment of these proportions is similar to the previous sensitivity analysis. After the simulation, we compared and discussed the result of boarding time from random method, back-to-front method, and WilMa method by plotting the graph between boarding time and increased time in each case of average number of luggage per flight.

4.3.2 Result

According to the previous results, the graphs show that the amount of the carry-on luggage has a significant effect on the boarding time, more luggage, more required time of boarding. And from the result in this part, the graphs represent that the relationship between carry-on luggage and the boarding time is not a linear relationship. The amount of the carry-on luggage affects to the rate of increased time according to the bigger slope of the graph while the number of carry-on bag is increased.



(a) Random Method (b) BMF Method (c) WilMA Method

Figure 8: Sensitivity Analysis on the factors of average number of luggage per flight

4.4 Task 2d: What's the Optimal Boarding Method

4.4.1 2 Additional Boarding Method

The 2 additional boarding method we proposed are the reverse pyramid method and back-to-front WilMA method.

Reverse Pyramid Method

The reverse pyramid method referred to the method derived from (reference). In the context of the narrow body passenger aircraft with 33 rows and 6 columns, the boarding scheme is illustrated in Figure 9. The people with seat the same color can switched between each other and the method would continued to be called the reverse pyramid method.

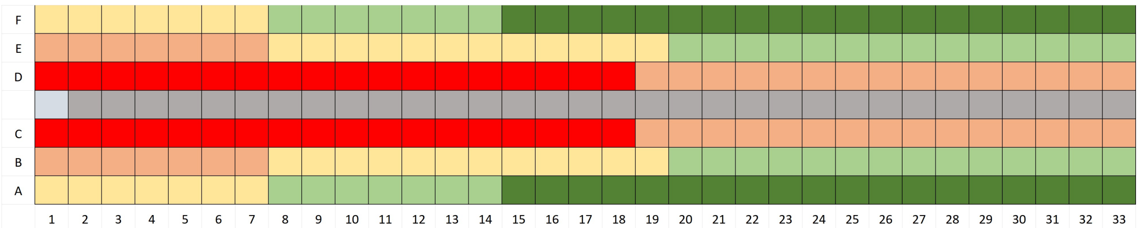
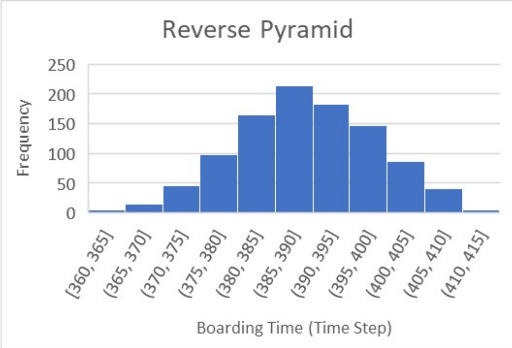


Figure 9: Boarding Scheme for the pyramid method. Green board first, Red board last

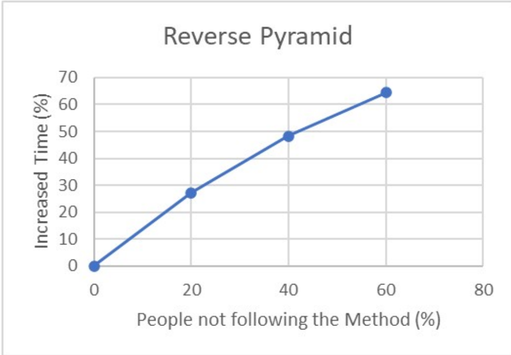
This method could theoretically reduce the occurrence of both the number of row interference and the luggage storing interference. The average time of the method when every passenger follow the boarding method and no passenger is carrying any luggage and the sensitivity analysis of the method toward the percentage of people not following the method is shown in the Figure 10 analyzed from 1000 loops from the shuffled pyramid method with all passenger following the method and no luggage.

Back-to-front WilMA method

This method is the result of combining the WilMA method together with the most effective section method. This method can resolve the row interference problem. The section method is used to reduce the effect of the



(a) The distribution of the reverse pyramid method from 1000 loops



(b) Sensitivity Analysis of people not following the method

Figure 10: The Reverse Pyramid Method Analysis

luggage interference. The scheme for the boarding method of the back-to-front WilMA method is shown in Figure 11. The airplane is separated into 3 blocks like the section method. The window seat of each section will be the first of the block to enter followed by the middle, and the aisle seat.

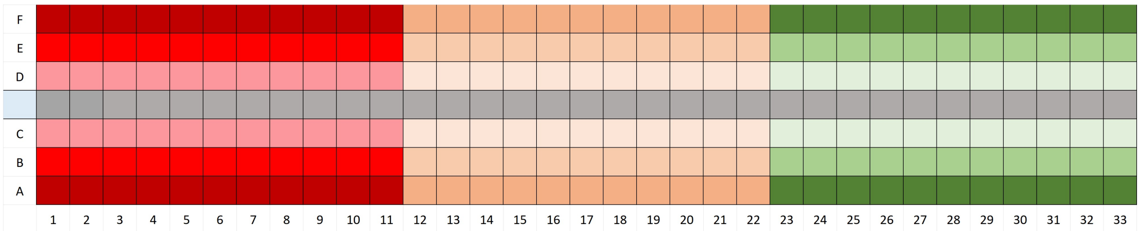
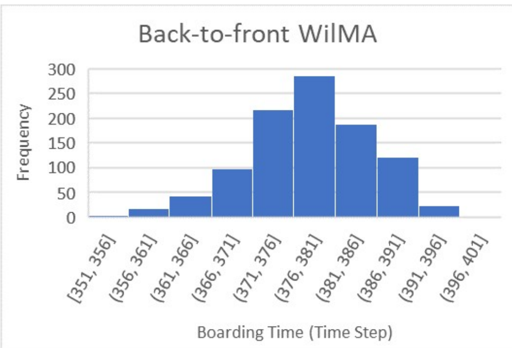
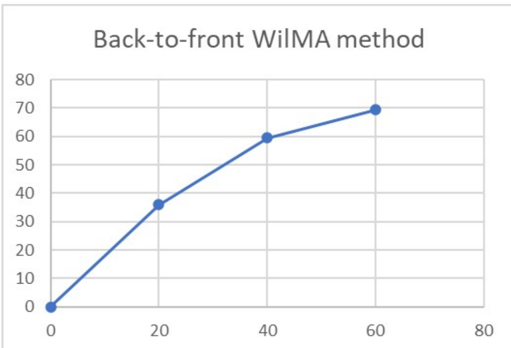


Figure 11: Back-to-Front WilMA method boarding scheme

The method is simulated for 1000 combinations of the back-to-front WilMA method with no luggage and all people following the method by shuffling the passenger in the 9 group, e.g. the window seat of the back section, the middle seat of the middle section, etc. After the 1000 loops are finished, the distributions of the boarding time is represented as a histogram in Figure 12a and the sensitivity analysis on the passenger not following the method is done and represented as a line graph in Figure 12b. The people not following the boarding method are made by switching people between the 9 group of passenger.



(a) The distribution of the back-to-front WilMA method from 1000 loops



(b) Sensitivity Analysis of people not following the method

Figure 12: The back-to-front WilMA Method Analysis

Lastly, the statistical value of the 1000 loops of both the back-to-front WilMA method and the reverse pyramid method is recorded in Table 3.

Method	Average Time (s)	5^{th} percentile	95^{th} percentile	Standard Deviation
Reverse Pyramid	779.324	748	810	18.58
Back-to-Front WilMA	756.836	730	780	14.93

Table 3: The Statistics for the reverse pyramid method and the back-to-front WilMA method

4.4.2 The Optimal Boarding Method

The optimal boarding method is the method that consume the least time and is less sensitive to percentage of people not following the method. The average time for all 5 methods is compared when all passenger follow the method and when 20 percent of the passenger does not. Both have an average of 50 bags per flight. The result is in Table 4.

Method	Time with all passenger following the method (s)	Time with 20% of the passenger not following the method (s)
Random	1539.86	1539.86
WilMA	873.616	1112.2
Back-to-Front	1664.008	1528.872
Reverse Pyramid	823.078	1024.622
Back-to-Front WilMA	807.836	1066.804

Table 4: The time used in each method in different parameter

The result of the show the back-to-front WilMA method has the best average time with reverse pyramid followed closely behind. However, when 20 percent of the passenger does not follow the boarding method, the back-to-front WilMA method consumed more time. So considering the average time and the sensitivity toward percentage of the passenger not following the method, the reverse pyramid is the optimal method, since the average time for both the method is close but the Reverse Pyramid Method is less sensitive toward the percentage of people not following the method.

4.5 Task 2e: What is the Optimal Disembarking Method

4.5.1 Greedy Solution

In the disembarking method, the authors propose the idea of greedy algorithm that selects the optimal sequence of passenger to leave the aircraft. The greedy solution is determined by selecting the minimum distance from each seat to the exit. If the distance is equal, the minimum total time using for collecting the luggage and move from the seat to the aisle of each passenger is considered. Hence, the greedy algorithm can be done by the following steps.

- Select the passenger who sit at the nearest seat from the exit.
- If there are more than one passenger with the smallest distance from the exit to their seat, select the passenger who has the lowest total time of collecting their luggage and moving to the aisle.

After the greedy algorithm selects the order of passengers, the Monte Carlo method is used to simulate the disembarking process. The passengers move to any empty aisle row according to the greedy conditions, and then walk to the exit by their speed. However, to ensure that this algorithm minimizes the total disembarking time, the next thing to concern is proving the correctness of greedy solution.

4.5.2 Correctness Proof

The important part of using greedy algorithm is to prove that it is the optimum. Thus, the "greedy stay ahead" techniques, which is the simple way to prove the correctness of greedy algorithm, is introduced.

First, defined $X = \{x_1, x_2, \ldots, x_k\}$ as the order of leaving passenger; x_1 and x_k is the first one and the last who leave the aircraft respectively. Let $f(x_i)$ is the time that $x_1, \ldots, x_i$ use for departing from the aircraft, $A = \{i_1, \ldots, i_n\}$ be the leaving passenger sequence selected by the greedy algorithm, $O = \{j_1, \ldots, j_n\}$ be an optimal solution; $f(j_n)$ is the minimum.

Assume that the solution from greedy algorithm is not the optimum for contradiction; $f(i_n) > f(j_n)$. Then, the next step is to prove that for all $k \leq n$, $f(i_k) \leq f(j_k)$ by mathematical induction.

Proof. Let $P(k)$ be the statement $f(i_k) \leq f(j_k)$, where $k \in \mathbf{N} \leq n_{passenger}$.
Basic step: If $k = 1$.
The passenger who is nearest to the exit and has the lowest time stowing their bags and getting up from the seat is the fastest person to leave the aircraft obviously.
Inductive step: Suppose that $P(t)$ is true and $P(t+1)$ is false for some $t > 1$.
It means $f(i_k) \leq f(j_k)$ and $f(i_{k+1}) \geq f(j_{k+1})$. Since, i_{k+1} has the smallest distance to the exit, it must be faster than other who has a seat next to i_{k+1}, so j_{k+1} must be at the same $seat_y$ with i_{k+1}. Despite that fact, i_{k+1} can store their luggage faster than j_{k+1}, thus, $f(i_{k+1}) \leq f(j_{k+1})$, which is contradiction. Therefore, $P(k+1)$ is true.
By mathematical induction, $P(k)$ is true for $k \in \mathbf{N} \leq n_{passenger}$. □

Finally, $f(i_n) \leq f(j_n)$ contradicts with $f(i_n) > f(j_n)$. Therefore, the greedy solution is the optimal solution.

4.5.3 Monte Carlo Simulation

The authors program the Monte Carlo simulation for 2 different disembarking methods including the greedy algorithm and the reverse WilMa method. The latter method is modified from the WilMa boarding method; the passengers at the aisle seat, the middle seat, and the window seat leave the aircraft respectively. Both method are simulated on the condition that the average luggage of each passenger is 0.5 and no one breaking the method rules. The result is shown in the Table 5

Method	**Average Time (s)**	5^{th} **percentile**	95^{th} **percentile**	**Standard Deviation**
Greedy Algorithm	1053.578	1008	1104	156.978
Modified-WilMa	1553.38	1468	1640	53.276

Table 5: The statistics of the Disembarking Method

5 Task 3: Different Aircraft Model and their optimal Boarding/Disembarking methods

5.1 Types of aircraft and their advantages

In this task, we were tasked with proposing the optimal method for the boarding/disembarking process of the aircraft. The 2 other aircraft have different characteristic than the standard "Narrow-Body" Passenger Aircraft. In this subsection, the authors will discuss on the characteristics and the advantage it has over the standard passenger aircraft.

"Flying Wing" Passenger Aircraft

The "Flying Wing" Passenger Aircraft is an aircraft is a relatively short passenger aircraft with a wide body. The generated model of the aircraft is 14 rows long and 24 column wide except the first 3 row where it is 18 columns wide. The aircraft contains 4 aisle, making the boarding method more effective than the standard aircraft since more aisle mean that there are multiple lines of passenger in the 4 aisles filling up the aircraft. This is advantages since an interference will not stall the only line of passenger in the aircraft but 1 of 4. The authors will assume that the passenger must enter through the side of the aisle only and the passenger must walk pass the aisle in front of row 1 to access each aisle.

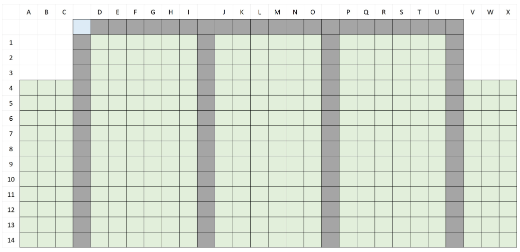
"Two-Entrance, Two-Aisle" Passenger Aircraft

The "Two-Entrance, Two-Aisle" Passenger Aircraft is an aircraft is similar to the standard passenger aircraft. The different is the wideness of the body, one addition aisle in the body, and the additional entrance located at the back of the aircraft. The additional aisle and entrance gave significant advantages over the standard model. For the additional aisle, the advantage is already discussed on in the "Flying-Wing" passenger aircraft. The additional entrance make it easier for passenger in the back row to reach their respective seat using the back entrance.

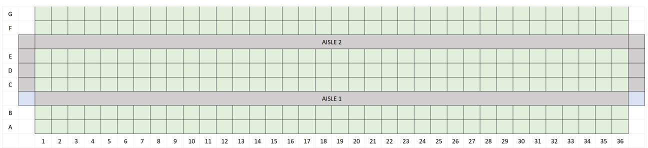
The methods proposed will utilized the additional characteristic with their advantages to lower the occurrence of interference as much as possible.

5.2 The Model modification

The model for the 2 type of aircraft are modified to fit the new aircraft. The time calculation is all the same as the "Narrow-Body" model along with the condition for each interference. The modified characteristic of the model is the map generated for aircraft and the walking path for each aircraft. The model will still take an input from the sequence generator calculate the time as the simulation time step. The generated map for both of the aircraft is illustrated in Figure 13.



(a) "Flying-Wing" Passenger Aircraft



(b) "Two-Entrance, Two-Aisle" Passenger Aircraft

Figure 13: The Map Generation

5.3 The Boarding/Disembarking Method for "Flying Wing" Passenger Aircraft

5.3.1 Boarding Method

The boarding method for "Flying Wing" Passenger Aircraft is the derived from the WilMA method for the standard model. The methods start with separating the aircraft into 4 blocks illustrated in Figure 14 named Block A to D from left to right. Since each block now looks like a standard narrow-body method, the aircraft is boarded using the WilMA method for each block. So the boarding scheme becomes WilMA method for block D, C, B, and A respectively. This method can lower the occurrence of row interference. The Monte Carlo method is used in with the condition of 0.5 average luggage per person or 99 pieces of luggage per each flight, and 20 percent chance each passenger has of being slow. Sensitivity analysis for the number of passenger not following the method is also done.

The statistics result is shown in Table 6. The distribution of all the boarding method in the flying wing model is illustrate as a histogram in Figure 15a and the sensitivity analysis toward percentage of people not following the method is in Figure 15b.

Method	Average Time (s)	5^{th} percentile	95^{th} percentile	Standard Deviation
Flying Wing	1701.876	1614	1788	54.142

Table 6: The statistics of the Flying Wing Method

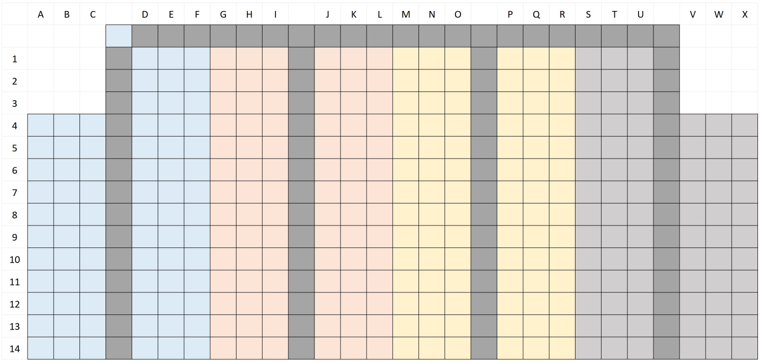
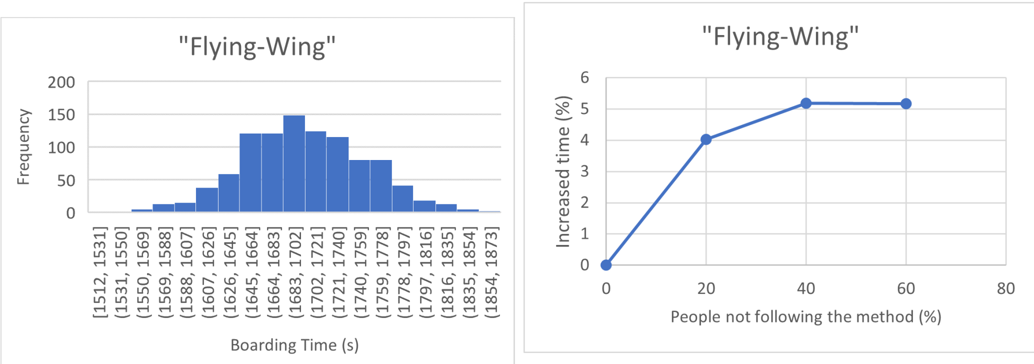


Figure 14: The separated block in the Flying Wing Passenger Aircraft boarding method. The separated blocks are colored differently.



(a) Boarding Time Distribution.

(b) Sensitivity Analysis Toward the People not following the method.

Figure 15: Flying Wing Method

5.3.2 Disembarking Method

In the flying wing passenger aircraft, the greedy solution also works well with this type of aircraft. According to the figure 14, the airplane is separated to 4 section and each section has the structure like the narrow-aisle aircraft. Consequently, the greedy algorithm can be done separately to each block of the airplane. Note that at the top-right of the black section and the top-left seat in the blue section that have no seat do not affect to the greedy solution as it does not vary in the greedy algorithm conditions.

5.4 The Boarding/Disembarking Method for "Two-Entrance, Two-Aisle" Passenger Aircraft

5.4.1 Boarding Method

The airplane is split in half, the front and the back. The front half of the airplane is entered from the front entrance and the back half enters from the bag. This is to utilize both the 2 entrance and let the 2 entrance hold equal passenger by splitting it in half. After separating the passenger into 2 group, there will be ordering within the group according to their seat column. The order will be as followed:

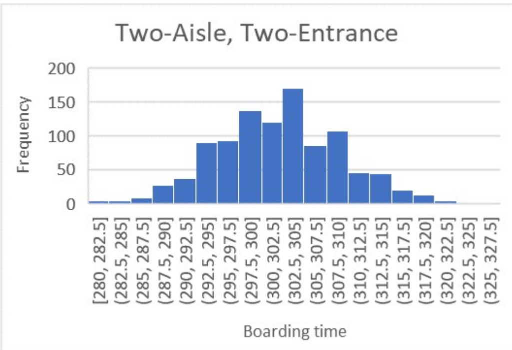
1. Seat A and G
2. Seat B and F
3. Seat C and D
4. Seat E

This grouping was inspired from the WilMA method. The order of the seat is done to filled the seat from the outside to the inside to avoid the row interference. Seat column A,B,C will enter from aisle 1 and seat column D,E,F,G will enter from the aisle 2. The 4 group by column from both the front and the back can shuffle between each other in their own respective group. Time simulation is done to find the average boarding time of this method and the sensitivity analysis on the people not following the method is done to evaluate the effectiveness of this method. The average number of carry-on luggage of all simulation will be 0.5 and all the passenger can have a 20% chance of being a slow passenger. Monte Carlo method is used in the method analysis. The sensitivity analysis for the people not following the method is also done

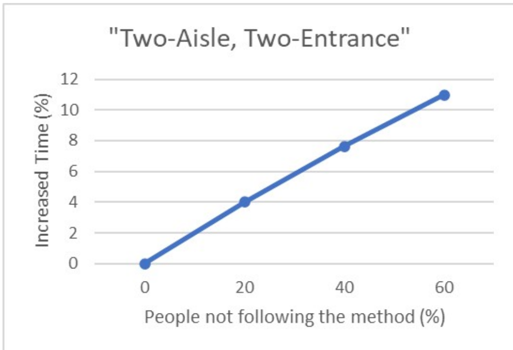
The statistics result is shown in Table 7. The distribution of all the boarding method in the two-entrances, two-aisles model is illustrate as a histogram in Figure 16a and the sensitivity analysis toward percentage of people not following the method is in Figure 16b.

Method	**Average Time (s)**	**5^{th} percentile**	**95^{th} percentile**	**Standard Deviation**
Two-Aisle Two-Entrance	604.618	582	630	14.062

Table 7: The statistics of the Two-Entrance, Two-Aisle Method



(a) Boarding Time Distribution.



(b) Sensitivity Analysis Toward the People not following the method.

Figure 16: Two-Entrance, Two-Aisle Method

The distribution shows low standard deviation and a low sensitivity toward the percentage of people not following the method.

5.4.2 Disembarking Method

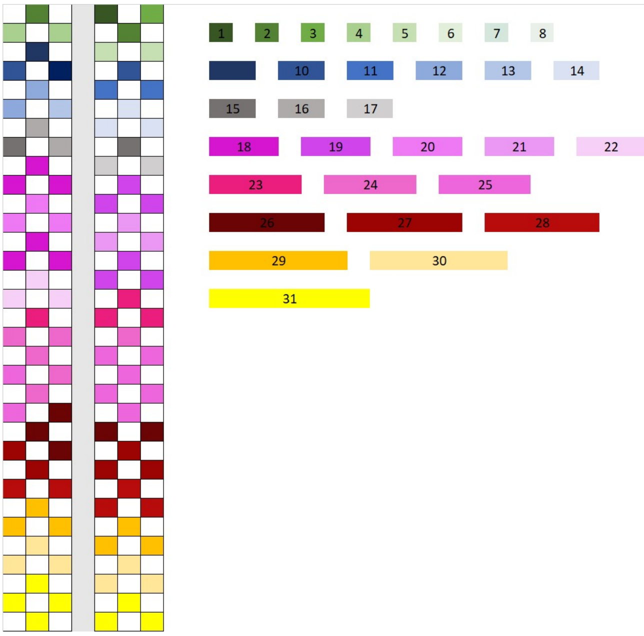
In the two-entrance, two-aisle passenger aircraft, as well as the previous aircraft, the aforementioned greedy algorithm works with this type of aircraft. Like the boarding method, the aircraft is split in half, and each of them is considered as a same structure plane but has an opposite walk way. Then, each section is separated in half again, but this time the size of new sections are not equal. By without loss of generality, the 2 sections are row A to row C and row D to row G. So, the greedy algorithm can be done separately on these 4 sections.

6 Task 4: Capacity Limitation

6.1 Boarding Method

According to the pandemic situation, the passenger capacity needs to be reduced to decrease the risk of spreading disease. Furthermore, avoiding seats that are close to each other is another factor that should be considered for the same reason while reducing the capacity. We also assume that there is no people who not followed the method because everyone should follow the measures to avoid spreading disease. However, when there is a limitation of the capacity and the rules to avoid sitting next to each other, those proposed method will not work at all because the process is very fixed to the position of the seat, and it will change most of the processes and lose the main concept of those methods away. Therefore, in this part, we are going to propose

new methods for each percentage of passengers.
In the case of 30% capacity of narrow body plane, to avoid pandemic and use the luggage bin efficiently, we should avoid people sitting on the same side of the row. Furthermore, about 2 out of 3 of all seats should not be used which will last 33% of seats of middle seat, which we will get the allocation like the picture below. Since the objective is the seat capacity should not be over 30% so we need to emit some more seats. To keep it fast, we should eliminate the far seat. Lastly, we will use the back-front method cause this method can deal with problems with only vertically seat. In the case of 50% of narrow body plane, we use method from[9] which have been proved that it is second fastest among other capacity limit method. However we choose this method because people is sit more separate than the fastest one. The sequence and seat position is shown for 50% is shown in figure 17. In case of 70% capacity limit we will empty all of the most middle seat to make people far from each other and reach 70%. The seat that have middle seat is going to be the one that near entrances. Lastly we use back-front method for boarding



(a) Seat assignment to different groups using regular pattern for 50% capacity

Figure 17: Best capacity limit seat allocation [9]

For the method that we proposed for flying wing aircraft is derived Wilma. However, when the allocation changes, some of the process needs to be adapted to the scenario. Recap that in flying wing aircraft, our proposed method depends on section separating like in the suggested method and repetition of our method to each aisle. Hence in 30% capacity, we are going to use back-front method with keep only middle seat of each section and repeat it in every aisle. In case of 50% capacity, we use same seat assignment as 50% narrow body which is alternate between 1 middle seat or 1 aisle seat and 1 window seat and will use back-front method to board. In case of 70% we are going to use seat assignment which most of them don't have middle seat but some of it need to be filled to get 70% instead of 66% and then use back-front method to fill up.
The boarding method for the 2 entrances key concept is that we separate the aircraft into two exact sections that have the same number of rows inside. We already prove how good it is in terms of avoiding interference. However, the sequence is strict with the column which it would be change a lot when the capacity limit is reduced. Hence, we are going to keep only the key concept. Furthermore, as there are two entrances, we should balance the number of passengers to each section equally. For 30% capacity limit, the middle section are going to be used only middle seat and then separate the seat that left into two side section. Furthermore, we are going to use back-front to approach each side of entrance. In case of 50% capacity limit, the middle section are going to be alternate like figure 17 and share the left seats to side section and use back-front method to approach from in side. In case of 70% capacity limit, the middle section won't be used and share left seats to side section and then use back-front method to approach from each side.

6.2 Disembarking Method

According to the greedy solution, $f(i_k)$ is the minimum for all $k \in \mathbf{N} \leq n_{passenger}$. If the passenger number is limited, the greedy solution also give the optimal solution that is $f(i_x)$, where x is the total number of passengers after considering the capacity limitation. To be more clearly, the algorithm does not vary on the number of people; it only considers the distance from the exit and collecting time, which will not be changed although the passenger number is decreased. Since the greedy algorithm can work normally on all three types of aircraft even though the passenger capacitance is limited. Therefore, the disembarking method when limiting the capacity of the passenger can be effectively done by the greedy algorithm.

7 Strengths, Weaknesses, and Improvements of Model

Our aircraft seat map and sequence generator are created manually and tested using the simulation. We need to construct new sequences to simulate different methods which take lots of time but it's good because we can expand our method testing easily. Moreover, our model is more realistic than other mathematical models. It clarifies all the behavior of passengers and shows how each method works clearly and it is precise due to a lot of sample simulation. However, simulation consumes lots of time to simulate in each round. Moreover, with more time, we can simulate more rounds. This will cause a more precise result, time of boarding and disembarking in each method. For the future plan, we should use a more stochastic model to vary the number of total luggage to make it more realistic. Furthermore, our two-entrance plane seat map is not accurate, it is not the same as the problem. We make it simpler by changing some of the seats.

8 Conclusion

In this paper, the authors aim to optimize the boarding method and the disembarking method of the passenger aircraft.

In task 1, the authors successfully created a method of simulating the time it took for the boarding method and the disembarking method. The model consider the interference occurring during the process, the effect of increasing luggage in the overhead bin, and the speed of the passenger. The input for the boarding method is the queue entering the aircraft and the characteristic of each passenger, how much luggage they have and their speed. For the disembarking method, the model require the priority of the passenger leaving the aircraft and also each queue characteristic.

In task 2a, the sequence generator for the 3 prescribed method (Random, Section, Seat) were created for the boarding model and the Monte Carlo experiment using different permutations of each method was used to obtain the statistical value of each method.

In task 2b, basic sensitivity analysis were conducted on the variable of the average number of luggage per flight and the percentage of passenger not following the method. The data suggest that the WilMA method is the most effective method out of the 3 methods.

In task 2c, the effect of more average luggage per flight on the boarding time was investigated. The number of luggage showed significant effect on the boarding time and the relationship is shown as a graph in this section.

In task 2d, 2 additional methods were introduced: the reverse pyramid method and the back-to-front WilMA method. The 5 prescribed methods were then compared against each other to see which is the most optimum method. The result showed that the reverse WilMA method is the most optimum method due to its low average time and low sensitivity to percentage of people not following the prescribed method.

In task 2e, the greedy algorithm was used to create the priority input in the disembarking model. Mathematical Proof of its effectiveness is also done. The Monte Carlo Method for disembarking method was created to obtain the distribution and the statistical value of the model

In task 3, new boarding method along with its boarding time distribution and the sensitivity analysis of the percentage of people not following the method is proposed for the 2 other type of aircraft, the "Flying-Wing" and the "Two-Entrance, Two-Aisle" aircraft. The disembarking method uses the greedy algorithm since the aircraft can be seen as a standard airplane sticking together.

In task 4, the effect of the limit capacity on the optimum boarding and disembarking method was evaluated. At different limit capacity, there will be new seating plan. Due to the old boarding method being strict with the seating plan being full, new boarding methods are needed. Disembarking method however will remain the same since the capacity limit does not violate the proof of the greedy algorithm.

9 Task 5: A Letter to the Airline Executive

Dear Sir/Madam

Our group has derived a model to aid you in reducing the time required for both the Boarding Method and the Disembarking Method, the 2 process involving the passengers which most in airline industries try to reduce. Our group wrote up a simulation to simulate both process and calculate the time for both process and determine the optimal process for both process. Other than the standard "Narrow-Body" Passenger Aircraft, the evaluation was also conducted on different Passenger Aircraft Model like the "Flying Wing" and the "Two-Entrance, Two-Aisle" Aircraft.

The model we wrote up simulate all the action of the passenger in the plane from the action of walking, storing baggage, and even the action of the passenger getting up to give way for the passenger in the inner seat to reach his seat. In our model, the aisle can only fit 1 person, so a blockage in the aisle can occur if a passenger stop. Passengers will stop for 2 actions: to store the luggage and to wait for the previously seated passenger to make way for him if necessary. The passengers in the model are the normal type and the slow type who takes double the amount of time for each action. We assume the chance of a passenger being a slow person is 0.2. The luggage is all the same, and storing time would increases as more luggage is stored in the overhead bin.

We simulate different method commonly used in the boarding and the disembarking method for the "Narrow-Body" Passenger Aircraft including the Random Boarding Method, the Section Method where the passengers are boarded by sections(front, middle, back), the WilMA method where the passengers are boarded by their seat type (window, middle, and aisle), the Reverse Pyramid (a method proposed in previous literature), and finally the Back-to-Front WilMA method, a combination of the WilMA and the section method. The effect of the amount of luggage and the percentage of people not following the boarding method is also investigated.

The data from the results confirm that the most effective method for "Narrow Body" Passenger Aircraft is the Reverse-Pyramid method. With no passenger breaking the rule and no luggage, the average time for the reverse-pyramid method is around 13 minutes with a low standard deviation. Despite using more time than the Back-to-Front WilMA method, the model is less sensitive to the people not following the boarding method making it more effective in real practice. The sensitivity toward the average luggage per flight is the similar for all method.

Our group has also proposed the optimal disembarking method using the queue generated using the greedy algorithm. Basically, the program will find method using lesser and lesser time than the current method until there isn't any left. Unfortunately, this method is highly coordinated and the is very sensitive toward the percentage of people not following the method, but if you can control the queue of all passenger, this is the optimal method.

Boarding and Disembarking Method for the "Flying Wing" and the "Two-Entrance, Two-Aisle" Passenger Aircraft are also proposed. For the "Flying Wing" Aircraft, we proposed the a boarding method of separating the aircraft into blocks and viewing each block of seats as a normal "Narrow Body" Passenger Aircraft. Each block, from right to left, is filled using the WilMA method, assuming the Entrance is on the left of the aircraft. As for the "Two-Aisle,Two-Seat" Aircraft, the airplane is first separate into 2 sections, the front half that will enter using the front entrance and the back half that will enter using the back entrance. Each half is then separate into group according to the seat column they are in. The group entering the aircraft is in the following order: seat A and seat G, seat B and seat F, seat C and seat D, and lastly seat E. This method prevent the delay caused by the row blockage and utilized the 2 aisle and the ability to walk from the front and back of the aircraft to the fullest

In the midst of the pandemics, we acknowledge some of the airline's attempt at stopping the contagious diseases by using the limit capacity in some flight. Due to the change in the passenger's amount, new methods are required for the new optimal seating plan at different capacity limit. With new seating plan, some problem when the Aircraft is fully boarded is not present with the new seating plan so new method can be proposed without the interference you would have on a normal airplane. The disembarking method is the same since the greedy algorithm is not affected by the change in passenger number.

We sincerely hope our findings are somewhat useful, and can help your industry optimize the time required for each process. If you have any inquiry about our research, please feel free to reach out.

Yours Faithfully,
IMMC2022055

10 Bibliography

References

[1] Bachmat, E., Berend, D., Sapir, L., Skiena, S., Stolyarov, N. (2009). Analysis of Airplane Boarding Times. Operations Research, 57(2), 499–513. https://doi.org/10.1287/opre.1080.0630

[2] Bachmat, E., Khachaturov, V., Kuperman, R. (2013). Optimal back-to-front airplane boarding. Physical Review E, 87(6). https://doi.org/10.1103/physreve.87.062805

[3] Bazargan, M. (2007). A linear programming approach for aircraft boarding strategy. European Journal of Operational Research, 183(1), 394–411. https://doi.org/10.1016/j.ejor.2006.09.071

[4] Cotfas, L. A., Delcea, C., Milne, R. J., Salari, M. (2020). Evaluating Classical Airplane Boarding Methods Considering COVID-19 Flying Restrictions. Symmetry, 12(7), 1087. https://doi.org/10.3390/sym12071087

[5] Hutter, L., Jaehn, F., Neumann, S. (2019). Influencing factors on airplane boarding times. Omega, 87, 177–190. https://doi.org/10.1016/j.omega.2018.09.002

[6] Luo, L., Hong, S., Shang, S., Zhou, X., Yang, J., Pan, Y. (2021). Intelligent Boarding Modelling and Evaluation: A Simulation-Based Approach. Journal of Advanced Transportation, 2021, 1–12. https://doi.org/10.1155/2021/9973336

[7] Milne, R. J., Salari, M. (2016). Optimization of assigning passengers to seats on airplanes based on their carry-on luggage. Journal of Air Transport Management, 54, 104–110. https://doi.org/10.1016/j.jairtraman.2016.03.022

[8] Salari, M., Milne, R. J., Kattan, L. (2019). Airplane boarding optimization considering reserved seats and passengers' carry-on bags. OPSEARCH, 56(3), 806–823. https://doi.org/10.1007/s12597-019-00405-z

[9] Schultz, M., Soolaki, M. (2021). Analytical approach to solve the problem of aircraft passenger boarding during the coronavirus pandemic. Transportation research. Part C, Emerging technologies, 124, 102931. https://doi.org/10.1016/j.trc.2020.102931

[10] Steffen, J. H. (2008). Optimal boarding method for airline passengers. Journal of Air Transport Management, 14(3), 146–150. https://doi.org/10.1016/j.jairtraman.2008.03.003

11 Appendix

11.1 Code

All simulation code are written in C++ language with some messy part.

11.1.1 Sequence Generator

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

struct info
{
    int pos,posx;
    char seatx;
    int seaty;
    int luggage;
    int speed;
    char ent;
};

void kakshuffle(int *array, int n)
{
    if (n > 1)
    {
        for (int i = 1; i < n; i++)
        {
            int j = i + rand() / (RAND_MAX / (n - i) + 1);
            int tmp = array[j];
            array[j] = array[i];
            array[i] = tmp;
        }
    }
}

void shuffle(info *array, int n)
{
    if (n > 1)
    {
        for (int i = 1; i < n; i++)
        {
            int j = i + rand() / (RAND_MAX / (n - i) + 1);
            info tmp = array[j];
            array[j] = array[i];
            array[i] = tmp;
        }
    }
}

void shuffle_s(info *array, int m, int n)
{
    if (n > 1)
    {
        for (int i = m; i < n; i++)
        {
            int j = i + rand() / (RAND_MAX / (n - i) + 1);
            info tmp = array[j];
            array[j] = array[i];
            array[i] = tmp;
        }
    }
}

void information(info person[],int npassenger,int seatx)
{
    srand(time(NULL));
    for(int i=1;i<=npassenger;i++)
```

```
    {
        person[i].seaty = floor(i/seatx)+1;
        person[npassenger].seaty = i/seatx;
        if ((i%seatx)==1)
            person[i].seatx = 'A';
        else if ((i%seatx)==2)
            person[i].seatx = 'B';
        else if ((i%seatx)==3)
            person[i].seatx = 'C';
        else if ((i%seatx)==4)
            person[i].seatx = 'D';
        else if ((i%seatx)==5)
            person[i].seatx = 'E';
        else if ((i%seatx)==0)
            person[i].seatx = 'F';
        person[i].luggage = 0;
        int r = rand()%10+1;
        if(r<8)
            person[i].speed = 1;
        else
            person[i].speed = 2;
    }
}

void random(info person[],int npassenger,int seatx,int x)
{
    srand(x);
    for(int i=1;i<=npassenger;i++)
    {
        person[i].seaty = floor(i/seatx)+1;
        person[npassenger].seaty = i/seatx;
        if ((i%seatx)==1)
            person[i].seatx = 'A';
        else if ((i%seatx)==2)
            person[i].seatx = 'B';
        else if ((i%seatx)==3)
            person[i].seatx = 'C';
        else if ((i%seatx)==4)
            person[i].seatx = 'D';
        else if ((i%seatx)==5)
            person[i].seatx = 'E';
        else if ((i%seatx)==0)
            person[i].seatx = 'F';
        person[i].luggage = 0;
        int r = rand()%10+1;
        if(r<8)
            person[i].speed = 1;
        else
            person[i].speed = 2;
    }
    int ii,jj,kk;
    for(kk=1;kk<=npassenger;kk++)
    {
        int num[npassenger];
        for(ii=0;ii<npassenger;ii++)
        num[ii] = ii+1;
        kakshuffle(num,npassenger);
        int Prob0=60,Prob1=31,Prob2=8,Prob3=1,Prob4=0;
        for(jj=0;jj<npassenger;jj++)
        {
            if(jj<(Prob0*npassenger/100)) person[num[jj]].luggage = 0;
            else if((Prob0*npassenger/100)<=jj&&jj<((Prob0+Prob1)*npassenger/100)) person[num[
    jj]].luggage = 1;
            else if(((Prob0+Prob1)*npassenger/100)<=jj&&jj<(((Prob0+Prob1+Prob2)*npassenger
    /100))) person[num[jj]].luggage = 2;
            else if(((Prob0+Prob1+Prob2)*npassenger/100)<=jj&&jj<(((Prob0+Prob1+Prob2+Prob3)*
    npassenger/100))) person[num[jj]].luggage = 3;
            else if(((Prob0+Prob1+Prob2+Prob3)*npassenger/100)<=jj&&jj<(((Prob0+Prob1+Prob2+
    Prob3+Prob4)*npassenger/100))) person[num[jj]].luggage = 4;
            else person[num[jj]].luggage = 5;
```

```
        }
    }
    shuffle(person,npassenger);
}


void section(info person[],int npassenger,int seatx, int bad, int x, int y)
{
    int round=3;
    bad = bad/2;
    srand(x);
    int h = npassenger/(6*round);
    int g = 6*h;
    int sec[npassenger+5];
    for(int i=1;i<=npassenger;i++)
    {
        if (y==1)
        {
            if(i<=g) person[i].seaty = floor((i-1)/seatx)+1;
            else if((i>g)&&(i<=(2*g))) person[i].seaty = h+floor((i-g-1)/seatx)+1;
            else if(i>(2*g)) person[i].seaty = 2*h+floor((i-2*g-1)/seatx)+1;
        }
        else if (y==2)
        {
            if(i<=g) person[i].seaty = floor((i-1)/seatx)+1;
            else if((i>g)&&(i<=(2*g))) person[i].seaty = 2*h+floor((i-g-1)/seatx)+1;
            else if(i>(2*g)) person[i].seaty = h+floor((i-2*g-1)/seatx)+1;
        }
        else if (y==3)
        {
            if(i<=g) person[i].seaty = h+floor((i-1)/seatx)+1;
            else if(i>g&&i<=2*g) person[i].seaty = floor((i-g-1)/seatx)+1;
            else if(i>2*g) person[i].seaty = 2*h+floor((i-2*g-1)/seatx)+1;
        }
        else if (y==4)
        {
            if(i<=g) person[i].seaty = h+floor((i-1)/seatx)+1;
            else if(i>g&&i<=2*g) person[i].seaty = 2*h+floor((i-g-1)/seatx)+1;
            else if(i>2*g) person[i].seaty = floor((i-2*g-1)/seatx)+1;
        }
        else if (y==5)
        {
            if(i<=g) person[i].seaty = 2*h+floor((i-1)/seatx)+1;
            else if(i>g&&i<=2*g) person[i].seaty = floor((i-g-1)/seatx)+1;
            else if(i>2*g) person[i].seaty = h+floor((i-2*g-1)/seatx)+1;
        }
        else if (y==6)
        {
            if(i<=g) person[i].seaty = 2*h+floor((i-1)/seatx)+1;
            else if(i>h&&i<=2*g) person[i].seaty = h+floor((i-g-1)/seatx)+1;
            else if(i>2*g) person[i].seaty = floor((i-2*g-1)/seatx)+1;
        }
        if(i<=g) sec[i]=1;
        else if(i>h&&i<=2*g) sec[i]=2;
        else if(i>2*g) sec[i]=3;
        if ((i%seatx)==1)
            person[i].seatx = 'A';
        else if ((i%seatx)==2)
            person[i].seatx = 'B';
        else if ((i%seatx)==3)
            person[i].seatx = 'C';
        else if ((i%seatx)==4)
            person[i].seatx = 'D';
        else if ((i%seatx)==5)
            person[i].seatx = 'E';
        else if ((i%seatx)==0)
            person[i].seatx = 'F';
        person[i].luggage = 0;
        int r = rand()%10+1;
        if(r<8)
```

```
            person[i].speed = 1;
        else
            person[i].speed = 2;
    }
    int ii,jj,kk;
    for(kk=1;kk<=npassenger;kk++)
    {
        int num[npassenger];
        for(ii=0;ii<npassenger;ii++)
        num[ii] = ii+1;
        kakshuffle(num,npassenger);
        int Prob0=22,Prob1=33,Prob2=25,Prob3=13,Prob4=5;
        for(jj=0;jj<npassenger;jj++)
        {
            if(jj<(Prob0*npassenger/100)) person[num[jj]].luggage = 0;
            else if((Prob0*npassenger/100)<=jj&&jj<((Prob0+Prob1)*npassenger/100)) person[num[
    jj]].luggage = 1;
            else if(((Prob0+Prob1)*npassenger/100)<=jj&&jj<(((Prob0+Prob1+Prob2)*npassenger
    /100))) person[num[jj]].luggage = 2;
            else if(((Prob0+Prob1+Prob2)*npassenger/100)<=jj&&jj<(((Prob0+Prob1+Prob2+Prob3)*
    npassenger/100))) person[num[jj]].luggage = 3;
            else if(((Prob0+Prob1+Prob2+Prob3)*npassenger/100)<=jj&&jj<(((Prob0+Prob1+Prob2+
    Prob3+Prob4)*npassenger/100))) person[num[jj]].luggage = 4;
            else person[num[jj]].luggage = 5;
        }
    }
    int m = 1;
    int n = npassenger/round;
    while(m < npassenger)
    {
        shuffle_s(person,m,n);
        m = n+1;
        n = n+npassenger/round;
    }
    info arr;
    int a;
    int b;
    bool already[npassenger+5]={};
    int i = 0;
    while(i<bad*npassenger/100)
    {
        a=0;
        b=0;
        while(already[a]== true||a==0)
            a = (rand() % (npassenger - 1 + 1)) + 1;
        while(sec[b]==sec[a]||b==0||already[b]== true)
            b = (rand() % (npassenger - 1 + 1)) + 1;
        arr = person[a];
        person[a] = person[b];
        person[b] = arr;
        already[a] = true;
        already[b] = true;
        i++;
    }
}

void seat(info person[],int npassenger,int seatx, int bad,int x)
{
    bad = bad/2;
    srand(x);
    int n=npassenger/6;
    int sec[npassenger+5];
    for(int i=1;i<=npassenger;i++)
    {
        if (5*n<i&&i<=npassenger){
            person[i].seatx = 'C';
            person[i].seaty = (i-1)%(npassenger/seatx)+1;
            sec[i] = 1;
        }
        else if (4*n<i&&i<=5*n){
```

```
            person[i].seatx = 'D';
            person[i].seaty = (i-1)%(npassenger/seatx)+1;
            sec[i] = 1;
        }
        else if (3*n<i&&i<=4*n){
            person[i].seatx = 'B';
            person[i].seaty = (i-1)%(npassenger/seatx)+1;
            sec[i] = 2;
        }
        else if (2*n<i&&i<=3*n){
            person[i].seatx = 'E';
            person[i].seaty = (i-1)%(npassenger/seatx)+1;
            sec[i] = 2;
        }
        else if (1*n<i&&i<=2*n){
            person[i].seatx = 'A';
            person[i].seaty = (i-1)%(npassenger/seatx)+1;
            sec[i] = 3;
        }
        else if (0<i&&i<=1*n){
            person[i].seatx = 'F';
            person[i].seaty = (i-1)%(npassenger/seatx)+1;
            sec[i] = 3;
        }

        int r = rand()%10+1;
        if(r<8)
            person[i].speed = 1;
        else
            person[i].speed = 2;
    }
    int m = 1;
    int c = 2*npassenger/seatx;
    while(m < npassenger)
    {
        shuffle_s(person,m,c);
        m = c+1;
        c = c+2*npassenger/seatx;
    }
    int ii,jj,kk;
    for(kk=1;kk<=npassenger;kk++)
    {
        int num[npassenger];
        for(ii=0;ii<npassenger;ii++)
        num[ii] = ii+1;
        kakshuffle(num,npassenger);
        int Prob0=22,Prob1=33,Prob2=25,Prob3=13,Prob4=5;
        for(jj=0;jj<npassenger;jj++)
        {
            if(jj<(Prob0*npassenger/100)) person[num[jj]].luggage = 0;
            else if((Prob0*npassenger/100)<=jj&&jj<((Prob0+Prob1)*npassenger/100)) person[num[
    jj]].luggage = 1;
            else if(((Prob0+Prob1)*npassenger/100)<=jj&&jj<(((Prob0+Prob1+Prob2)*npassenger
    /100))) person[num[jj]].luggage = 2;
            else if(((Prob0+Prob1+Prob2)*npassenger/100)<=jj&&jj<(((Prob0+Prob1+Prob2+Prob3)*
    npassenger/100))) person[num[jj]].luggage = 3;
            else if(((Prob0+Prob1+Prob2+Prob3)*npassenger/100)<=jj&&jj<(((Prob0+Prob1+Prob2+
    Prob3+Prob4)*npassenger/100))) person[num[jj]].luggage = 4;
            else person[num[jj]].luggage = 5;
        }
    }
    info arr;
    int a;
    int b;
    bool already[npassenger+5]={};
    int i = 0;
    while(i<bad*npassenger/100)
    {
        a=0;
        b=0;
```

```
        while(already[a]== true||a==0)
            a = (rand() % (npassenger - 1 + 1)) + 1;
        while(sec[b]==sec[a]||b==0||already[b]== true)
            b = (rand() % (npassenger - 1 + 1)) + 1;
        arr = person[a];
        person[a] = person[b];
        person[b] = arr;
        already[a] = true;
        already[b] = true;
        i++;
    }
}

void repyramid(info person[],int bad,int x)
{
    int npassenger = 198;
    int seatx = 6;
    bad = bad/2;
    srand(x);
    int a=38,b=42,c=38,d=44,e=36,i=1;
    int j=0;
    int k=0;
    int sec[npassenger+5];
    while(i<=a)
    {
        person[i].seaty = j+15;
        person[i].seatx = 'A';
        person[i+1].seaty = j+15;
        person[i+1].seatx = 'F';
        i=i+2;
        j++;
        sec[i]=1;
    }
    j=0;
    k=0;
    while(i<=a+b)
    {
        if(i-(a+1)<14)
        {
            person[i].seaty = j+8;
            person[i].seatx = 'A';
            person[i+1].seaty = j+8;
            person[i+1].seatx = 'F';
            i=i+2;
            j++;
        }
        else
        {
            person[i].seaty = k+20;
            person[i].seatx = 'B';
            person[i+1].seaty = k+20;
            person[i+1].seatx = 'E';
            i=i+2;
            k++;
        }
        sec[i]=2;
    }
    j=0;
    k=0;
    while(i<=a+b+c)
    {
        if(i-(a+b+1)<14)
        {
            person[i].seaty = j+1;
            person[i].seatx = 'A';
            person[i+1].seaty = j+1;
            person[i+1].seatx = 'F';
            i=i+2;
            j++;
        }
```

```
        else
        {
            person[i].seaty = k+8;
            person[i].seatx = 'B';
            person[i+1].seaty = k+8;
            person[i+1].seatx = 'E';
            i=i+2;
            k++;
        }
        sec[i]=3;
    }
    j=0;
    k=0;
    while(i<=a+b+c+d)
    {
        if(i-(a+b+c+1)<14)
        {
            person[i].seaty = j+1;
            person[i].seatx = 'B';
            person[i+1].seaty = j+1;
            person[i+1].seatx = 'E';
            i=i+2;
            j++;
        }
        else
        {
            person[i].seaty = k+19;
            person[i].seatx = 'C';
            person[i+1].seaty = k+19;
            person[i+1].seatx = 'D';
            i=i+2;
            k++;
        }
        sec[i]=4;
    }
    j=0;
    k=0;
    while(i<=a+b+c+d+e)
    {
        person[i].seaty = j+1;
        person[i].seatx = 'C';
        person[i+1].seaty = j+1;
        person[i+1].seatx = 'D';
        i=i+2;
        j++;
        sec[i]=5;
    }
    shuffle_s(person,1,a);
    shuffle_s(person,a+1,a+b);
    shuffle_s(person,a+b+1,a+b+c);
    shuffle_s(person,a+b+c+1,a+b+c+d);
    shuffle_s(person,a+b+c+d+1,a+b+c+d+e);
    for(i=1;i<=npassenger;i++)
    {
        int r = rand()%10+1;
        if(r<8)
            person[i].speed = 1;
        else
            person[i].speed = 2;
    }
    int ii,jj,kk;
    for(kk=1;kk<=npassenger;kk++)
    {
        int num[npassenger];
        for(ii=0;ii<npassenger;ii++)
        num[ii] = ii+1;
        kakshuffle(num,npassenger);
        int Prob0=22,Prob1=33,Prob2=25,Prob3=13,Prob4=5;
        for(jj=0;jj<npassenger;jj++)
        {
```

```
            if(jj<(Prob0*npassenger/100)) person[num[jj]].luggage = 0;
            else if((Prob0*npassenger/100)<=jj&&jj<((Prob0+Prob1)*npassenger/100)) person[num[
    jj]].luggage = 1;
            else if(((Prob0+Prob1)*npassenger/100)<=jj&&jj<(((Prob0+Prob1+Prob2)*npassenger
    /100))) person[num[jj]].luggage = 2;
            else if(((Prob0+Prob1+Prob2)*npassenger/100)<=jj&&jj<(((Prob0+Prob1+Prob2+Prob3)*
    npassenger/100))) person[num[jj]].luggage = 3;
            else if(((Prob0+Prob1+Prob2+Prob3)*npassenger/100)<=jj&&jj<(((Prob0+Prob1+Prob2+
    Prob3+Prob4)*npassenger/100))) person[num[jj]].luggage = 4;
            else person[num[jj]].luggage = 5;
        }
    }
    info arr;
    int o;
    int p;
    bool already[npassenger+5]={};
    i = 0;
    while(i<bad*npassenger/100)
    {
        o=0;
        p=0;
        while(already[o]== true||o==0)
        o = (rand() % (npassenger - 1 + 1)) + 1;
        while(sec[p]==sec[o]||p==0||already[p]== true)
            p = (rand() % (npassenger - 1 + 1)) + 1;
        arr = person[o];
        person[o] = person[p];
        person[p] = arr;
        already[o] = true;
        already[p] = true;
        i++;
    }
}

void seatsection(info person[],int npassenger,int seatx, int bad, int x)
{
    bad = bad/2;
    srand(x);
    int round = 3;
    int n = npassenger/3;
    int p = npassenger/6;
    int m = n/6;
    int sec[npassenger];
    int i=0,j=0;
    while(j<3)
    {
        for(i=1;i<=n;i++)
        {
            if (5*m<i&&i<=n){
                person[j*n+i].seatx = 'C';
                person[j*n+i].seaty = p-(j*m+((i-1)%m));
                sec[j*n+i] = 3+3*j;
            }
            else if (4*m<i&&i<=5*m){
                person[j*n+i].seatx = 'D';
                person[j*n+i].seaty = p-(j*m+((i-1)%m));
                sec[j*n+i] = 3+3*j;
            }
            else if (3*m<i&&i<=4*m){
                person[j*n+i].seatx = 'B';
                person[j*n+i].seaty = p-(j*m+((i-1)%m));
                sec[j*n+i] = 2+3*j;
            }
            else if (2*m<i&&i<=3*m){
                person[j*n+i].seatx = 'E';
                person[j*n+i].seaty = p-(j*m+((i-1)%m));
                sec[j*n+i] = 2+3*j;
            }
            else if (1*m<i&&i<=2*m){
                person[j*n+i].seatx = 'A';
```

```
                person[j*n+i].seaty = p-(j*m+((i-1)%m));
                sec[j*n+i] = 1+3*j;
            }
            else if (0<i&&i<=1*m){
                person[j*n+i].seatx = 'F';
                person[j*n+i].seaty = p-(j*m+((i-1)%m));
                sec[j*n+i] = 1+3*j;
            }
        }
        j++;
    }
    int g = 1;
    int c = 2*m;
    while(g < npassenger)
    {
        shuffle_s(person,g,c);
        g = c+1;
        c = c+2*m;
    }
    int ii,jj,kk;
    for(kk=1;kk<=npassenger;kk++)
    {
        int num[npassenger];
        for(ii=0;ii<npassenger;ii++)
        num[ii] = ii+1;
        kakshuffle(num,npassenger);
        int Prob0=22,Prob1=33,Prob2=25,Prob3=13,Prob4=5;
        for(jj=0;jj<npassenger;jj++)
        {
            if(jj<(Prob0*npassenger/100)) person[num[jj]].luggage = 0;
            else if((Prob0*npassenger/100)<=jj&&jj<((Prob0+Prob1)*npassenger/100)) person[num[
    jj]].luggage = 1;
            else if(((Prob0+Prob1)*npassenger/100)<=jj&&jj<(((Prob0+Prob1+Prob2)*npassenger
    /100))) person[num[jj]].luggage = 2;
            else if(((Prob0+Prob1+Prob2)*npassenger/100)<=jj&&jj<(((Prob0+Prob1+Prob2+Prob3)*
    npassenger/100))) person[num[jj]].luggage = 3;
            else if(((Prob0+Prob1+Prob2+Prob3)*npassenger/100)<=jj&&jj<(((Prob0+Prob1+Prob2+
    Prob3+Prob4)*npassenger/100))) person[num[jj]].luggage = 4;
            else person[num[jj]].luggage = 5;
        }
    }
    for(i=1;i<=npassenger;i++)
    {
        int r = rand()%10+1;
            if(r<8)
                person[i].speed = 1;
            else
                person[i].speed = 2;
    }
    info arr;
    int o;
    int q;
    bool already[npassenger+5]={};
    i = 0;
    while(i<bad*npassenger/100)
    {
        o=0;
        q=0;
        int h = 0;
        while(already[o]== true||o==0)
        o = (rand() % (npassenger - 1 + 1)) + 1;
        while(sec[q]==sec[o]||q==0||already[q]== true)
        {
            q = (rand() % (npassenger - 1 + 1)) + 1;
            h++;
            if(h>10) break;
        }
        if(h<10)
        {
            arr = person[o];
```

```
            person[o] = person[q];
            person[q] = arr;
            already[o] = true;
            already[q] = true;
            i++;
        }
        else i++;
    }
}

void doubleaisle(info person[],int npassenger,int seatx,int seaty,int bad,int x)
{
    bad = bad/2;
    srand(x);
    int n=npassenger/7;
    for(int i=1;i<=npassenger;i++)
    {
        if (6*n<i&&i<=npassenger){
            person[i].seatx = 'E';
            person[i].seaty = (i-1)%(npassenger/seatx)+2;
        }
        else if (5*n<i&&i<=6*n){
            person[i].seatx = 'D';
            person[i].seaty = (i-1)%(npassenger/seatx)+2;
        }
        else if (4*n<i&&i<=5*n){
            person[i].seatx = 'C';
            person[i].seaty = (i-1)%(npassenger/seatx)+2;
        }
        else if (3*n<i&&i<=4*n){
            person[i].seatx = 'F';
            person[i].seaty = (i-1)%(npassenger/seatx)+2;
        }
        else if (2*n<i&&i<=3*n){
            person[i].seatx = 'B';
            person[i].seaty = (i-1)%(npassenger/seatx)+2;
        }
        else if (1*n<i&&i<=2*n){
            person[i].seatx = 'G';
            person[i].seaty = (i-1)%(npassenger/seatx)+2;
        }
        else if (0<i&&i<=1*n){
            person[i].seatx = 'A';
            person[i].seaty = (i-1)%(npassenger/seatx)+2;
        }
        int r = rand()%10+1;
        if(r<8)
            person[i].speed = 1;
        else
            person[i].speed = 2;
        if(person[i].seaty<=seaty/2+1)
            person[i].ent = 'F',person[i].posx=1;
        else
            person[i].ent = 'B',person[i].posx=seaty+2;
    }
    int ii,jj,kk;
    for(kk=1;kk<=npassenger;kk++)
    {
        int num[npassenger];
        for(ii=0;ii<npassenger;ii++)
        num[ii] = ii+1;
        kakshuffle(num,npassenger);
        int Prob0=60,Prob1=31,Prob2=8,Prob3=1,Prob4=0,Prob5=0;
        for(jj=0;jj<npassenger;jj++)
        {
            if(jj<(Prob0*npassenger/100)) person[num[jj]].luggage = 0;
            else if((Prob0*npassenger/100)<=jj&&jj<((Prob0+Prob1)*npassenger/100)) person[num[
    jj]].luggage = 1;
            else if(((Prob0+Prob1)*npassenger/100)<=jj&&jj<(((Prob0+Prob1+Prob2)*npassenger
    /100))) person[num[jj]].luggage = 2;
```

```
            else if(((Prob0+Prob1+Prob2)*npassenger/100)<=jj&&jj<(((Prob0+Prob1+Prob2+Prob3)*
    npassenger/100))) person[num[jj]].luggage = 3;
            else if(((Prob0+Prob1+Prob2+Prob3)*npassenger/100)<=jj&&jj<(((Prob0+Prob1+Prob2+
    Prob3+Prob4)*npassenger/100))) person[num[jj]].luggage = 4;
             else if(((Prob0+Prob1+Prob2+Prob3+Prob4)*npassenger/100)<=jj&&jj<(((Prob0+Prob1+
    Prob2+Prob3+Prob4+Prob5)*npassenger/100))) person[num[jj]].luggage = 5;
            else person[num[jj]].luggage = 6;
        }
    }
    int m = 1;
    int c = 2*npassenger/seatx;
    while(m+npassenger/seatx < npassenger)
    {
        shuffle_s(person,m,c);
        shuffle_s(person,m,c-npassenger/seatx);
        m = c+1;
        c = c+2*npassenger/seatx;
    }
    shuffle_s(person,m,c-npassenger/seatx);
    info arr;
    int a;
    int b;
    bool already[npassenger+5]={};
    int i = 0;
    while(i<bad*npassenger/100)
    {
        a=0;
        b=0;
        while(already[a]== true||a==0)
            a = (rand() % (npassenger - 1 + 1)) + 1;
        while(person[b].seatx==person[a].seatx||b==0||already[b]== true)
            b = (rand() % (npassenger - 1 + 1)) + 1;
        arr = person[a];
        person[a] = person[b];
        person[b] = arr;
        already[a] = true;
        already[b] = true;
        i++;
    }
}

void flying_wing(info person[],int bad,int x)
{
    bad=bad/2;
    srand(x);
    int i=1,j,sec[320],npassenger=318;
    j=0;
    while(j<25)
    {
        if(j/11==0) person[i].seatx='X';
        else person[i].seatx='S';
        sec[i]=1;
        j++;
        i++;
    }
    j=0;
    while(j<25)
    {
        if(j/11==0) person[i].seatx='W';
        else person[i].seatx='T';
        sec[i]=2;
        j++;
        i++;
    }
    j=0;
    while(j<25)
    {
        if(j/11==0) person[i].seatx='V';
        else person[i].seatx='U';
        sec[i]=3;
```

```
        j++;
        i++;
    }
    j=0;
    while(j<28)
    {
        if(j/14==0) person[i].seatx='R';
        else person[i].seatx='M';
        sec[i]=4;
        j++;
        i++;
    }
    j=0;
    while(j<28)
    {
        if(j/14==0) person[i].seatx='Q';
        else person[i].seatx='N';
        sec[i]=5;
        j++;
        i++;
    }
    j=0;
    while(j<28)
    {
        if(j/14==0) person[i].seatx='P';
        else person[i].seatx='O';
        sec[i]=6;
        j++;
        i++;
    }
    j=0;
    while(j<28)
    {
        if(j/14==0) person[i].seatx='L';
        else person[i].seatx='G';
        sec[i]=7;
        j++;
        i++;
    }
    j=0;
    while(j<28)
    {
        if(j/14==0) person[i].seatx='K';
        else person[i].seatx='H';
        sec[i]=8;
        j++;
        i++;
    }
    j=0;
    while(j<28)
    {
        if(j/14==0) person[i].seatx='J';
        else person[i].seatx='I';
        sec[i]=9;
        j++;
        i++;
    }
    j=0;
    while(j<25)
    {
        if(j/14==0) person[i].seatx='F';
        else person[i].seatx='A';
        sec[i]=10;
        j++;
        i++;
    }
    j=0;
    while(j<25)
    {
        if(j/14==0) person[i].seatx='E';
```

```
            else person[i].seatx='B';
            sec[i]=11;
            j++;
            i++;
        }
        j=0;
        while(j<25)
        {
            if(j/14==0) person[i].seatx='D';
            else
            {
                person[i].seatx='C';
            }
            sec[i]=12;
            j++;
            i++;
        }
        int k=0;
        i=1;
        for(k=1;k<=6;k++)
        {
            j=0;
            while(j<25)
            {
                if(j/11==0) person[i].seaty=j%11+1;
                else person[i].seaty=(j-11)%14+1;
                j++;
                i++;
            }
            k++;
        }
        k=0;
        for(k=1;k<=12;k++)
        {
            j=0;
            while(j<28)
            {
                if(j/14==0) person[i].seaty=j%14+1;
                else person[i].seaty=(j-14)%14+1;
                j++;
                i++;
            }
            k++;
        }
        k=0;
        for(k=1;k<=6;k++)
        {
            j=0;
            while(j<25)
            {
                if(j/14==0) person[i].seaty=j%14+1;
                else person[i].seaty=(j-14)%11+1;
                j++;
                i++;
            }
            k++;
        }
        int m = 1;
        int c = 25;
        while(m < 75)
        {
            shuffle_s(person,m,c);
            m = c+1;
            c = c+25;
        }
        c=c+3;
        while(m < 243)
        {
            shuffle_s(person,m,c);
            m = c+1;
```

```
        c = c+28;
    }
    c=c-3;
    while(m < 318)
    {
        shuffle_s(person,m,c);
        m = c+1;
        c = c+25;
    }
    for(i=0;i<=npassenger;i++)
    {
    int r = rand()%10+1;
    if(r<8)
        person[i].speed = 1;
    else
        person[i].speed = 2;
    }
    info arr;
    int a;
    int b;
    bool already[npassenger+5]={};
    i = 0;
    while(i<bad)
    {
        a = (rand() % (npassenger - 1 + 1)) + 1;
        b = (rand() % (npassenger - 1 + 1)) + 1;
        arr = person[a];
        person[a] = person[b];
        person[b] = arr;
        i++;
    }
}
```

11.1.2 Narrow-Body Passenger Aircraft

```
#include<bits/stdc++.h>
#include "Sequence_Generator.h"
using namespace std;
const int N=50,M=50,NXM=350,NS=1000;
info p[NXM];
bool arr[N][M];
double t_boarding,t_disembarking,t_seat=4,t_walk=2,wait[NXM];
int nPeople=198,nRows=33,bin[M][2];
double t_boarding_avg,t_disembarking_avg;
vector<double> t_boarding_acc,t_disembarking_acc;
map<int,string> state;
map<char,int> side,c2i;
set<int> s;

void boarding(){
    while(!s.empty()){
        t_boarding++;
        for(int i=1;i<=nPeople;i++){
            if(state[i]=="Not in Aisle"){
                if(arr[4][1]) break;
                state[i]="In Aisle";
                arr[4][1]=true;
                wait[i]=p[i].speed;
                p[i].pos=1;
                break;
            }
            else if(state[i]=="In Aisle"){
                // at thier seat
                if(p[i].pos==p[i].seaty){
                    state[i]="Storage";
                    // time step for storage
                    wait[i]=ceil((bin[p[i].seaty][side[p[i].seatx]]+p[i].luggage)*p[i].luggage
    *p[i].speed/2);
                    // no luggage, just wait for seating
                    if(wait[i]==0){
                        state[i]="Waiting";
```

```
                        bin[p[i].seaty][side[p[i].seatx]]+=p[i].luggage;
                        // if there are people sit before
                        if(p[i].seatx=='A' and (arr[2][p[i].seaty] and arr[3][p[i].seaty])){
                            wait[i]=floor(0.5+5*t_seat); continue; }
                        else if(p[i].seatx=='A' and (arr[2][p[i].seaty] or arr[3][p[i].seaty])){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='B' and arr[3][p[i].seaty]){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='E' and arr[5][p[i].seaty]){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='F' and (arr[5][p[i].seaty] and arr[6][p[i].seaty])){
                            wait[i]=floor(0.5+5*t_seat); continue; }
                        else if(p[i].seatx=='F' and (arr[5][p[i].seaty] or arr[6][p[i].seaty])){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        state[i]="Seated";
                        arr[4][p[i].seaty]=false;
                        arr[c2i[p[i].seatx]][p[i].seaty]=true;
                        p[i].pos=-1;
                        s.erase(i);
                    }
                    continue;
                }
                wait[i]--;
                if(wait[i]==0){
                    // there is space in front of them
                    if(!arr[4][p[i].pos+1]){
                        arr[4][p[i].pos]=false;
                        arr[4][++p[i].pos]=true;
                        wait[i]=p[i].speed;
                    }
                    else wait[i]=1;
                }
            }
            else if(state[i]=="Storage"){
                wait[i]--;
                if(wait[i]==0){
                    state[i]="Waiting";
                    bin[p[i].seaty][side[p[i].seatx]]+=p[i].luggage;
                    // if there are people sit before
                    if(p[i].seatx=='A' and (arr[2][p[i].seaty] and arr[3][p[i].seaty])){
                        wait[i]=floor(0.5+5*t_seat); continue; }
                    else if(p[i].seatx=='A' and (arr[2][p[i].seaty] or arr[3][p[i].seaty])){
                        wait[i]=floor(0.5+3*t_seat); continue; }
                    else if(p[i].seatx=='B' and arr[3][p[i].seaty]){
                        wait[i]=floor(0.5+3*t_seat); continue; }
                    else if(p[i].seatx=='E' and arr[5][p[i].seaty]){
                        wait[i]=floor(0.5+3*t_seat); continue; }
                    else if(p[i].seatx=='F' and (arr[5][p[i].seaty] and arr[6][p[i].seaty])){
                        wait[i]=floor(0.5+5*t_seat); continue; }
                    else if(p[i].seatx=='F' and (arr[5][p[i].seaty] or arr[6][p[i].seaty])){
                        wait[i]=floor(0.5+3*t_seat); continue; }
                    state[i]="Seated";
                    arr[4][p[i].seaty]=false;
                    arr[c2i[p[i].seatx]][p[i].seaty]=true;
                    p[i].pos=-1;
                    s.erase(i);
                }
            }
            else if(state[i]=="Waiting"){
                wait[i]--;
                if(wait[i]==0){
                    state[i]="Seated";
                    arr[4][p[i].seaty]=false;
                    arr[c2i[p[i].seatx]][p[i].seaty]=true;
                    p[i].pos=-1;
                    s.erase(i);
                }
```

```
            }
            else if(state[i]=="Seated") continue;
        }
    }
}

vector<pair<int,int>> a[3];

void re_wilma(){
    for(int k=0;k<3;k++){
        sort(a[k].begin(),a[k].end());
        while(!a[k].empty()){
            t_disembarking++;
            for(int j=0;j<a[k].size();j++){
                int i=a[k][j].second;
                if(state[i]=="Seated"){
                    p[i].pos=p[i].seaty;
                    if(!arr[4][p[i].seaty]){
                        arr[c2i[p[i].seatx]][p[i].seaty]=false;
                        arr[4][p[i].seaty]=true;
                        if(p[i].seatx<='C') p[i].seatx='X';
                        else p[i].seatx='Y';
                        wait[i]=t_seat;
                        state[i]="Waiting";
                    }
                }
                else if(state[i]=="Waiting"){
                    wait[i]--;
                    if(wait[i]==0){
                        state[i]="Storage";
                        // time step for storage
                        wait[i]=ceil((bin[p[i].seaty][side[p[i].seatx]]+p[i].luggage)*p[i].
    luggage*p[i].speed/2);
                        if(wait[i]==0){
                            bin[p[i].seaty][side[p[i].seatx]]-=p[i].luggage;
                            state[i]="In Aisle";
                            wait[i]=p[i].speed;
                        }
                    }
                }
                else if(state[i]=="Storage"){
                    wait[i]--;
                    if(wait[i]==0){
                        bin[p[i].seaty][side[p[i].seatx]]-=p[i].luggage;
                        state[i]="In Aisle";
                        wait[i]=p[i].speed;
                    }
                }
                else if(state[i]=="In Aisle"){
                    wait[i]--;
                    if(wait[i]==0){
                        // there is space in front of them
                        if(!arr[4][p[i].pos-1]){
                            arr[4][p[i].pos]=false;
                            arr[4][--p[i].pos]=true;
                            a[k][j].first=p[i].pos;
                            if(p[i].pos==0){
                                state[i]="Not in Aisle";
                                p[i].pos=-1;
                                arr[4][0]=false;
                                a[k].erase(a[k].begin()+j);
                                continue;
                            }
                            wait[i]=p[i].speed;
                        }
                        else wait[i]=1;
                    }
                }
            }
        }
```

```
    }

}

int dis[NXM],tt[NXM];
struct G{
    int d,t,i,p;
    bool operator<(const G&o)const{
        if(d==o.d) return t<o.t;
        return d<o.d;
    }
};
bool cmpp(G a,G b){
    return a.p<b.p;
}
void greedy(){
    set<int> out;
    vector<G> q;
    for(int i=1;i<=nPeople;i++){
        dis[i]=abs(4-c2i[p[i].seatx])+p[i].seaty;
        tt[i]=t_seat+ceil(p[i].luggage*p[i].luggage*p[i].speed/2);
        p[i].pos=p[i].seaty;
        q.push_back({dis[i],tt[i],i,p[i].pos});
    }
    while(out.size()<nPeople){
        t_disembarking++;
        sort(q.begin(),q.end());
        for(int j=0;j<q.size();j++){
            int i=q[j].i;
            if(state[i]=="Seated"){
                if(arr[4][p[i].seaty]) continue;
                arr[c2i[p[i].seatx]][p[i].seaty]=false;
                arr[4][p[i].seaty]=true;
                if(p[i].seatx<='C') p[i].seatx='X';
                else p[i].seatx='Y';
                wait[i]=t_seat;
                state[i]="Waiting";
            }
        }
        sort(q.begin(),q.end(),cmpp);
        for(int j=0;j<q.size();j++){
            int i=q[j].i;
            if(state[i]=="Waiting"){
                wait[i]--;
                if(wait[i]==0){
                    state[i]="Storage";
                    // time step for storage
                    wait[i]=ceil((bin[p[i].seaty][side[p[i].seatx]]+p[i].luggage)*p[i].luggage
    *p[i].speed/2);
                    if(wait[i]==0){
                        bin[p[i].seaty][side[p[i].seatx]]-=p[i].luggage;
                        state[i]="In Aisle";
                        wait[i]=p[i].speed;
                    }
                }
            }
            else if(state[i]=="Storage"){
                wait[i]--;
                if(wait[i]==0){
                    bin[p[i].seaty][side[p[i].seatx]]-=p[i].luggage;
                    state[i]="In Aisle";
                    wait[i]=p[i].speed;
                }
            }
            else if(state[i]=="In Aisle"){
                wait[i]--;
                if(wait[i]==0){
                    // there is space in front of them
                    if(!arr[4][p[i].pos-1]){
                        arr[4][p[i].pos]=false;
```

```
                        arr[4][--p[i].pos]=true;
                        q[j].p=p[i].pos;
                        if(p[i].pos==0){
                            state[i]="Not in Aisle";
                            p[i].pos=-1;
                            arr[4][0]=false;
                            q.erase(q.begin()+j);
                            out.insert(i);
                            continue;
                        }
                        wait[i]=p[i].speed;
                    }
                    else wait[i]=1;
                }
            }
        }
    }
}

int main(){
    // file for collecting the data
    ofstream myfile;
    myfile.open("data.csv");
    side['A']=side['B']=side['C']=side['X']=0;
    side['D']=side['E']=side['F']=side['Y']=1;
    c2i['A']=1;
    c2i['B']=2;
    c2i['C']=3;
    c2i['D']=5;
    c2i['E']=6;
    c2i['F']=7;
    p[0].pos=1e9;
    for(int j=1;j<=NS;j++){
        fill_n(arr[0],N*M,false);
        fill_n(wait,NXM,0);
        fill_n(bin[0],M*2,0);
        // choose boarding method from sequence generator
        /*
        random(p,nPeople,6,j);
        seat(p,nPeople,6,0,j);
        section(p,nPeople,6,0,j,1);
        repyramid(p,60,j);
        seatsection(p,nPeople,6,20,j);
        */
        // initialize the passengers
        for(int i=1;i<=nPeople;i++){
            s.insert(i);
            state[i]="Not in Aisle";
            p[i].pos=0;
            if(p[i].seatx=='C' || p[i].seatx=='D') a[0].push_back({p[i].pos,i});
            else if(p[i].seatx=='B' || p[i].seatx=='E') a[1].push_back({p[i].pos,i});
            else a[2].push_back({p[i].pos,i});
        }
        boarding();
        // choose disembarking method
        /*
        re_wilma();
        greedy();
        */
        t_boarding_avg+=t_boarding;
        t_disembarking_avg+=t_disembarking;
        t_boarding_acc.push_back(t_boarding);
        t_disembarking_acc.push_back(t_disembarking);
        t_boarding=t_disembarking=0;
    }
    t_boarding_avg/=NS;
    t_disembarking_avg/=NS;
    sort(t_boarding_acc.begin(),t_boarding_acc.end());
    sort(t_disembarking_acc.begin(),t_disembarking_acc.end());
    myfile << "Time AVG. = " << t_boarding_avg << " " << t_disembarking_avg << "\n";
```

```
    myfile << "Time 5% = " << t_boarding_acc[5*NS/100-1] << " " << t_disembarking_acc[5*NS/100-1] << "\n";
    myfile << "Time 95% = " << t_boarding_acc[95*NS/100-1] << " " << t_disembarking_acc[95*NS/100-1] << "\n";
    myfile << "Time boarding:\n";
    for(int i=0;i<t_boarding_acc.size();i++) myfile << t_boarding_acc[i] << "\n";
    myfile << "Time disembarking\n";
    for(int i=0;i<t_disembarking_acc.size();i++) myfile << t_disembarking_acc[i] << "\n";
    myfile.close();
}
```

11.1.3 Flying Wing Passenger Aircraft

```
#include<bits/stdc++.h>
#include "Sequence_Generator.h"
using namespace std;
const int N=50,M=50,NXM=350,NS=1000;
info p[NXM];
bool arr[N][M];
double t_boarding,t_disembarking,t_seat=4,t_walk=2,wait[NXM];
int nPeople=318,nRows=15,bin[M][10];
double t_boarding_avg,t_disembarking_avg;
vector<double> t_boarding_acc,t_disembarking_acc;
map<int,string> state;
map<char,int> side,c2i;
set<int> s;

void boarding(){
    while(!s.empty()){
        t_boarding++;
        for(int i=1;i<=nPeople;i++){
            if(state[i]=="Not in Aisle"){
                if(arr[4][1]) break;
                state[i]="In Aisle";
                arr[4][1]=true;
                wait[i]=p[i].speed;
                p[i].pos=1;
                p[i].posx=4;
                break;
            }
            else if(state[i]=="In Aisle"){
                // at their seat
                if(p[i].pos==p[i].seaty){
                    state[i]="Storage";
                    // time step for storage
                    wait[i]=ceil((bin[p[i].seaty][side[p[i].seatx]]+p[i].luggage)*p[i].luggage*p[i].speed/2);
                    // no luggage, just wait for seating
                    if(wait[i]==0){
                        state[i]="Waiting";
                        bin[p[i].seaty][side[p[i].seatx]]+=p[i].luggage;
                        // if there are people sit before
                        if(p[i].seatx=='A' and arr[2][p[i].seaty] and arr[3][p[i].seaty]){
                            wait[i]=floor(0.5+5*t_seat); continue; }
                        else if(p[i].seatx=='A' and (arr[2][p[i].seaty] or arr[3][p[i].seaty])){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='B' and arr[3][p[i].seaty]){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='E' and arr[5][p[i].seaty]){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='F' and arr[6][p[i].seaty] and arr[5][p[i].seaty]){
                            wait[i]=floor(0.5+5*t_seat); continue; }
                        else if(p[i].seaty=='F' and (arr[6][p[i].seaty] or arr[5][p[i].seaty])){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='H' and arr[10][p[i].seaty]){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='G' and arr[9][p[i].seaty] and arr[10][p[i].seaty]){
```

```
                            wait[i]=floor(0.5+5*t_seat); continue; }
                        else if(p[i].seaty=='G' and (arr[9][p[i].seaty] or arr[10][p[i].seaty])){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='K' and arr[12][p[i].seaty]){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='L' and arr[12][p[i].seaty] and arr[13][p[i].seaty]){
                            wait[i]=floor(0.5+5*t_seat); continue; }
                        else if(p[i].seaty=='L' and (arr[12][p[i].seaty] or arr[13][p[i].seaty])){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='N' and arr[17][p[i].seaty]){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='M' and arr[16][p[i].seaty] and arr[17][p[i].seaty]){
                            wait[i]=floor(0.5+5*t_seat); continue; }
                        else if(p[i].seaty=='M' and (arr[16][p[i].seaty] or arr[17][p[i].seaty])){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='Q' and arr[19][p[i].seaty]){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='R' and arr[19][p[i].seaty] and arr[20][p[i].seaty]){
                            wait[i]=floor(0.5+5*t_seat); continue; }
                        else if(p[i].seaty=='R' and (arr[19][p[i].seaty] or arr[20][p[i].seaty])){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='T' and arr[24][p[i].seaty]){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='S' and arr[23][p[i].seaty] and arr[24][p[i].seaty]){
                            wait[i]=floor(0.5+5*t_seat); continue; }
                        else if(p[i].seaty=='S' and (arr[23][p[i].seaty] or arr[24][p[i].seaty])){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='W' and arr[26][p[i].seaty]){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='X' and arr[26][p[i].seaty] and arr[27][p[i].seaty]){
                            wait[i]=floor(0.5+5*t_seat); continue; }
                        else if(p[i].seaty=='X' and (arr[26][p[i].seaty] or arr[27][p[i].seaty])){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        state[i]="Seated";
                        arr[p[i].posx][p[i].seaty]=false;
                        arr[c2i[p[i].seatx]][p[i].seaty]=true;
                        p[i].pos=p[i].posx=-1;
                        s.erase(i);
                    }
                    continue;
                }
                wait[i]--;
                if(wait[i]==0){
                    // check if it is the correct column or not
                    if(p[i].posx==25 && p[i].seatx>='S'){
                        // there is space in front of them
                        if(!arr[p[i].posx][p[i].pos+1]){
                            arr[p[i].posx][p[i].pos]=false;
                            arr[p[i].posx][++p[i].pos]=true;
                            wait[i]=p[i].speed;
                        }
                        else wait[i]=1;
                        continue;
                    }
                    else if(p[i].posx==18 && p[i].seatx>='M'){
                        if(!arr[p[i].posx][p[i].pos+1]){
                            arr[p[i].posx][p[i].pos]=false;
                            arr[p[i].posx][++p[i].pos]=true;
                            wait[i]=p[i].speed;
```

```
                        }
                        else wait[i]=1;
                        continue;
                    }
                    else if(p[i].posx==11 && p[i].seatx>='G'){
                        if(!arr[p[i].posx][p[i].pos+1]){
                            arr[p[i].posx][p[i].pos]=false;
                            arr[p[i].posx][++p[i].pos]=true;
                            wait[i]=p[i].speed;
                        }
                        else wait[i]=1;
                        continue;
                    }
                    else if(p[i].posx==4){
                        if(!arr[p[i].posx][p[i].pos+1]){
                            arr[p[i].posx][p[i].pos]=false;
                            arr[p[i].posx][++p[i].pos]=true;
                            wait[i]=p[i].speed;
                        }
                        else wait[i]=1;
                        continue;
                    }
                    if(!arr[p[i].posx+1][p[i].pos]){
                        arr[p[i].posx][p[i].pos]=false;
                        arr[++p[i].posx][p[i].pos]=true;
                        wait[i]=p[i].speed;
                    }
                    else wait[i]=1;
                }
            }
            else if(state[i]=="Storage"){
                wait[i]--;
                if(wait[i]==0){
                    state[i]="Waiting";
                    bin[p[i].seaty][side[p[i].seatx]]+=p[i].luggage;
                    // if there are people sit before
                    if(p[i].seatx=='A' and arr[2][p[i].seaty] and arr[3][p[i].seaty]){
                        wait[i]=floor(0.5+5*t_seat); continue; }
                    else if(p[i].seatx=='A' and (arr[2][p[i].seaty] or arr[3][p[i].seaty])){
                        wait[i]=floor(0.5+3*t_seat); continue; }
                    else if(p[i].seatx=='B' and arr[3][p[i].seaty]){
                        wait[i]=floor(0.5+3*t_seat); continue; }
                    else if(p[i].seatx=='E' and arr[5][p[i].seaty]){
                        wait[i]=floor(0.5+3*t_seat); continue; }
                    else if(p[i].seatx=='F' and arr[6][p[i].seaty] and arr[5][p[i].seaty]){
                        wait[i]=floor(0.5+5*t_seat); continue; }
                    else if(p[i].seaty=='F' and (arr[6][p[i].seaty] or arr[5][p[i].seaty])){
                        wait[i]=floor(0.5+3*t_seat); continue; }
                    else if(p[i].seatx=='H' and arr[10][p[i].seaty]){
                        wait[i]=floor(0.5+3*t_seat); continue; }
                    else if(p[i].seatx=='G' and arr[9][p[i].seaty] and arr[10][p[i].seaty]){
                        wait[i]=floor(0.5+5*t_seat); continue; }
                    else if(p[i].seaty=='G' and (arr[9][p[i].seaty] or arr[10][p[i].seaty])){
                        wait[i]=floor(0.5+3*t_seat); continue; }
                    else if(p[i].seatx=='K' and arr[12][p[i].seaty]){
                        wait[i]=floor(0.5+3*t_seat); continue; }
                    else if(p[i].seatx=='L' and arr[12][p[i].seaty] and arr[13][p[i].seaty]){
                        wait[i]=floor(0.5+5*t_seat); continue; }
                    else if(p[i].seaty=='L' and (arr[12][p[i].seaty] or arr[13][p[i].seaty])){
                        wait[i]=floor(0.5+3*t_seat); continue; }
                    else if(p[i].seatx=='N' and arr[17][p[i].seaty]){
                        wait[i]=floor(0.5+3*t_seat); continue; }
                    else if(p[i].seatx=='M' and arr[16][p[i].seaty] and arr[17][p[i].seaty]){
                        wait[i]=floor(0.5+5*t_seat); continue; }
                    else if(p[i].seaty=='M' and (arr[16][p[i].seaty] or arr[17][p[i].seaty])){
                        wait[i]=floor(0.5+3*t_seat); continue; }
                    else if(p[i].seatx=='Q' and arr[19][p[i].seaty]){
                        wait[i]=floor(0.5+3*t_seat); continue; }
                    else if(p[i].seatx=='R' and arr[19][p[i].seaty] and arr[20][p[i].seaty]){
                        wait[i]=floor(0.5+5*t_seat); continue; }
```

```
                    else if(p[i].seaty=='R' and (arr[19][p[i].seaty] or arr[20][p[i].seaty])){
                        wait[i]=floor(0.5+3*t_seat); continue; }
                    else if(p[i].seatx=='T' and arr[24][p[i].seaty]){
                        wait[i]=floor(0.5+3*t_seat); continue; }
                    else if(p[i].seatx=='S' and arr[23][p[i].seaty] and arr[24][p[i].seaty]){
                        wait[i]=floor(0.5+5*t_seat); continue; }
                    else if(p[i].seaty=='S' and (arr[23][p[i].seaty] or arr[24][p[i].seaty])){
                        wait[i]=floor(0.5+3*t_seat); continue; }
                    else if(p[i].seatx=='W' and arr[26][p[i].seaty]){
                        wait[i]=floor(0.5+3*t_seat); continue; }
                    else if(p[i].seatx=='X' and arr[26][p[i].seaty] and arr[27][p[i].seaty]){
                        wait[i]=floor(0.5+5*t_seat); continue; }
                    else if(p[i].seaty=='X' and (arr[26][p[i].seaty] or arr[27][p[i].seaty])){
                        wait[i]=floor(0.5+3*t_seat); continue; }
                    state[i]="Seated";
                    arr[p[i].posx][p[i].seaty]=false;
                    arr[c2i[p[i].seatx]][p[i].seaty]=true;
                    p[i].pos=p[i].posx=-1;
                    s.erase(i);
                }
            }
            else if(state[i]=="Waiting"){
                wait[i]--;
                if(wait[i]==0){
                    state[i]="Seated";
                    arr[p[i].posx][p[i].seaty]=false;
                    arr[c2i[p[i].seatx]][p[i].seaty]=true;
                    p[i].pos=p[i].posx=-1;
                    s.erase(i);
                }
            }
            else if(state[i]=="Seated") continue;
        }
    }
}

int main(){
    // file for collecting the data
    ofstream myfile;
    myfile.open("data.csv");
    int temp=1;
    for(int i='A';i<='X';i++){
        side[i]=temp;
        if((i-'A'+1)%3==0) temp++;
    }
    for(int i='A';i<='X';i++){
        c2i[char(i)]=i-'A'+1;
        if(i=='C' || i=='I' || i=='O' || i=='U') i++;
    }
    p[0].pos=1e9;
    for(int j=1;j<=NS;j++){
        fill_n(arr[0],N*M,false);
        fill_n(wait,NXM,0);
        fill_n(bin[0],M*10,0);
        // from sequene generator
        flying_wing(p,0,j);
        // initialize the passengers
        for(int i=1;i<=nPeople;i++){
            s.insert(i);
            state[i]="Not in Aisle";
            p[i].pos=0;
        }
        boarding();
        t_boarding_avg+=t_boarding;
        t_boarding_acc.push_back(t_boarding);
        t_boarding=0;
    }
    t_boarding_avg/=NS;
    sort(t_boarding_acc.begin(),t_boarding_acc.end());
    myfile << "Time AVG. = " << t_boarding_avg << "\n";
```

```
    myfile << "Time 5% = " << t_boarding_acc[5*NS/100-1] << "\n";
    myfile << "Time 95% = " << t_boarding_acc[95*NS/100-1] << "\n";
    myfile << "Time boarding:\n";
    for(int i=0;i<t_boarding_acc.size();i++) myfile << t_boarding_acc[i] << "\n";
    myfile.close();
}
```

11.1.4 Two-Entrance, Two-Aisle Passenger Aircraft

```
#include<bits/stdc++.h>
#include "Sequence_Generator.h"
using namespace std;
const int N=50,M=50,NXM=350,NS=1000;
info p[NXM];
bool arr[N][M];
double t_boarding,t_disembarking,t_seat=4,t_walk=2,wait[NXM];
int nPeople=252,nRows=36,bin[M][10];
double t_boarding_avg,t_disembarking_avg;
vector<double> t_boarding_acc,t_disembarking_acc;
map<int,string> state;
map<char,int> side,c2i;
set<int> s;

void boarding(){
    while(!s.empty()){
        t_boarding++;
        for(int i=1;i<=nPeople;i++){
            if(state[i]=="Not in Aisle"){
                int pos;
                if(p[i].ent=='F') pos=1;
                else pos=nRows+2;
                if(arr[3][pos]) continue;;
                state[i]="In Aisle";
                arr[3][pos]=true;
                wait[i]=p[i].speed;
                p[i].pos=pos;
                p[i].posx=3;
                break;
            }
            else if(state[i]=="In Aisle"){
                // at their seat
                if(p[i].pos==p[i].seaty){
                    state[i]="Storage";
                    wait[i]=ceil((bin[p[i].seaty][side[p[i].seatx]]+p[i].luggage)*p[i].luggage
    *p[i].speed/2); // time step for storage
                    // no luggage, just wait for seating
                    if(wait[i]==0){
                        state[i]="Waiting";
                        bin[p[i].seaty][side[p[i].seatx]]+=p[i].luggage;
                        // if there are people sit before
                        if(p[i].seatx=='A' and arr[2][p[i].seaty]){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='G' and arr[8][p[i].seaty]){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='D' and arr[6][p[i].seaty]){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        state[i]="Seated";
                        arr[p[i].posx][p[i].seaty]=false;
                        arr[c2i[p[i].seatx]][p[i].seaty]=true;
                        p[i].pos=p[i].posx=-1;
                        s.erase(i);
                    }
                    continue;
                }
                wait[i]--;
                if(wait[i]==0){
                    // check if it is the correct column or not
                    if(p[i].posx==3 && p[i].seatx<='C'){
                        // there is space in front of them
                        if(p[i].ent=='F' && !arr[p[i].posx][p[i].pos+1]){
                            arr[p[i].posx][p[i].pos]=false;
```

```
                                arr[p[i].posx][++p[i].pos]=true;
                                wait[i]=p[i].speed;
                            }
                            else if(p[i].ent=='B' && !arr[p[i].posx][p[i].pos-1]){
                                arr[p[i].posx][p[i].pos]=false;
                                arr[p[i].posx][--p[i].pos]=true;
                                wait[i]=p[i].speed;
                            }
                            else wait[i]=1;
                            continue;
                        }
                        else if(p[i].posx==7 && p[i].seatx>'C'){
                            if(p[i].ent=='F' && !arr[p[i].posx][p[i].pos+1]){
                                arr[p[i].posx][p[i].pos]=false;
                                arr[p[i].posx][++p[i].pos]=true;
                                wait[i]=p[i].speed;
                            }
                            else if(p[i].ent=='B' && !arr[p[i].posx][p[i].pos-1]){
                                arr[p[i].posx][p[i].pos]=false;
                                arr[p[i].posx][--p[i].pos]=true;
                                wait[i]=p[i].speed;
                            }
                            else wait[i]=1;
                            continue;
                        }
                        if(!arr[p[i].posx+1][p[i].pos]){
                            arr[p[i].posx][p[i].pos]=false;
                            arr[++p[i].posx][p[i].pos]=true;
                            wait[i]=p[i].speed;
                        }
                        else wait[i]=1;
                    }
                }
                else if(state[i]=="Storage"){
                    wait[i]--;
                    if(wait[i]==0){
                        state[i]="Waiting";
                        bin[p[i].seaty][side[p[i].seatx]]+=p[i].luggage;
                        // if there are people sit before
                        if(p[i].seatx=='A' and arr[2][p[i].seaty]){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='G' and arr[8][p[i].seaty]){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        else if(p[i].seatx=='D' and arr[6][p[i].seaty]){
                            wait[i]=floor(0.5+3*t_seat); continue; }
                        state[i]="Seated";
                        arr[p[i].posx][p[i].seaty]=false;
                        arr[c2i[p[i].seatx]][p[i].seaty]=true;
                        p[i].pos=p[i].posx=-1;
                        s.erase(i);
                    }
                }
                else if(state[i]=="Waiting"){
                    wait[i]--;
                    if(wait[i]==0){
                        state[i]="Seated";
                        arr[p[i].posx][p[i].seaty]=false;
                        arr[c2i[p[i].seatx]][p[i].seaty]=true;
                        p[i].pos=p[i].posx=-1;
                        s.erase(i);
                    }
                }
                else if(state[i]=="Seated") continue;
            }
        }
}

int main(){
    // file for collecting the data
    ofstream myfile;
```

```
    myfile.open ("data.csv");
    side['A']=side['B']=side['X']=1;
    side['C']=side['Y']=2;
    side['D']=side['E']=side['Z']=3;
    side['F']=side['G']=side['W']=4;
    c2i['A']=1;
    c2i['B']=2;
    c2i['C']=4;
    c2i['D']=5;
    c2i['E']=6;
    c2i['F']=8;
    c2i['G']=9;
    p[0].pos=1e9;
    for(int j=1;j<=NS;j++){
        if(j==NS) cout << "END!";
        else if(j%100==0) cout << "Simulation number: " << j << "\nKeep going. . .\n";
        fill_n(arr[0],N*M,false);
        fill_n(wait,NXM,0);
        fill_n(bin[0],M*10,0);
        // from sequence generator
        doubleaisle(p,nPeople,7,36,60,j);
        // initialize the passengers
        for(int i=1;i<=nPeople;i++){
            s.insert(i);
            state[i]="Not in Aisle";
        }
        boarding();
        t_boarding_avg+=t_boarding;
        t_boarding_acc.push_back(t_boarding);
        t_boarding=t_disembarking=0;
    }
    t_boarding_avg/=NS;
    sort(t_boarding_acc.begin(),t_boarding_acc.end());
    myfile << "Time AVG. = " << t_boarding_avg << "\n";
    myfile << "Time 5% = " << t_boarding_acc[5*NS/100-1] << "\n";
    myfile << "Time 95% = " << t_boarding_acc[95*NS/100-1] << "\n";
    myfile << "Time boarding:\n";
    for(int i=0;i<t_boarding_acc.size();i++) myfile << t_boarding_acc[i] << "\n";
    myfile.close();
}
```