PROJECT 03

COMSC 340 Analysis of Algorithms, Spring 2019

Victoria Cigarroa, Kristina Mendela, Dhaval Patel

Abstract

This project analyzes and implements Two Lane and Three Lane Assembly-Line processes using Dynamic Programming.

Contents

Abbreviations and Glossary

Dynamic Programming: a bottom-up approach to solving problems; solving smaller problems first and storing the values along the way to get the final solution.

.txt: an unformatted text file.

Introduction

In Project 03, Dynamic Programming was used in order to analyze and implement Two Lane and Three Lane Assembly-Line process. This assembly line process can best be explained using a car assembly line example. A single car enters the assembly line and as it goes through, parts are added and a finished car is at the end of the assembly line. However, when a rush order is made, the manager will need to produce the cars in a faster way. Through Dynamic Programming, a process was completed in order to solve this problem.

Requirements

For Project 03 it is required to first implement a Two Lane and Three Lane Assembly-Line process. The code that is used for this project is Java. The second requirement is to create a time-complexity equation for the code that was written.

Design & Implementation

In order to track the results based on the various stations, we ensured that our handwritten results matched the ProcessTimesTest.txt file that was provided. This process is further explained in the "Testing" section of this report. When the transition from two lanes to three lanes, we started with the same algorithm so we could have a basis of how to move forward. The first step in creating the three lanes was to read in the files differently so that all the data will be read in correctly. Finally, a new row had to be added to the final answers of the files.

Development Environment

Machine 1: HP Notebook running Windows 10, 12 GB of RAM, and using Java as the programming language.

Machine 2: ASUS ROG running Windows 10, 16GB of RAM, and using Java as the programming language.

Machine 3: HP Spectre running Windows 10, 16GB of RAM, and Java as the programming language.

Target Environment

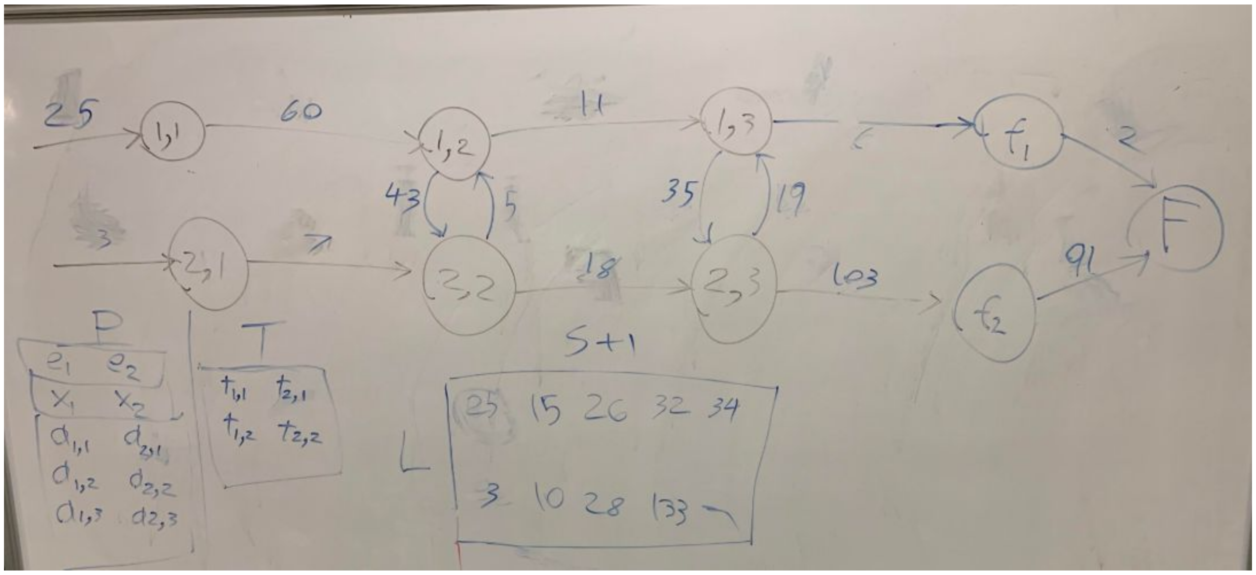
Machine 1: HP Notebook running Windows 10, 12GB of RAM, and using Java as the programming language.

Machine 2: ASUS ROG running Windows 10, 16GB of RAM, and using Java as the programming language.

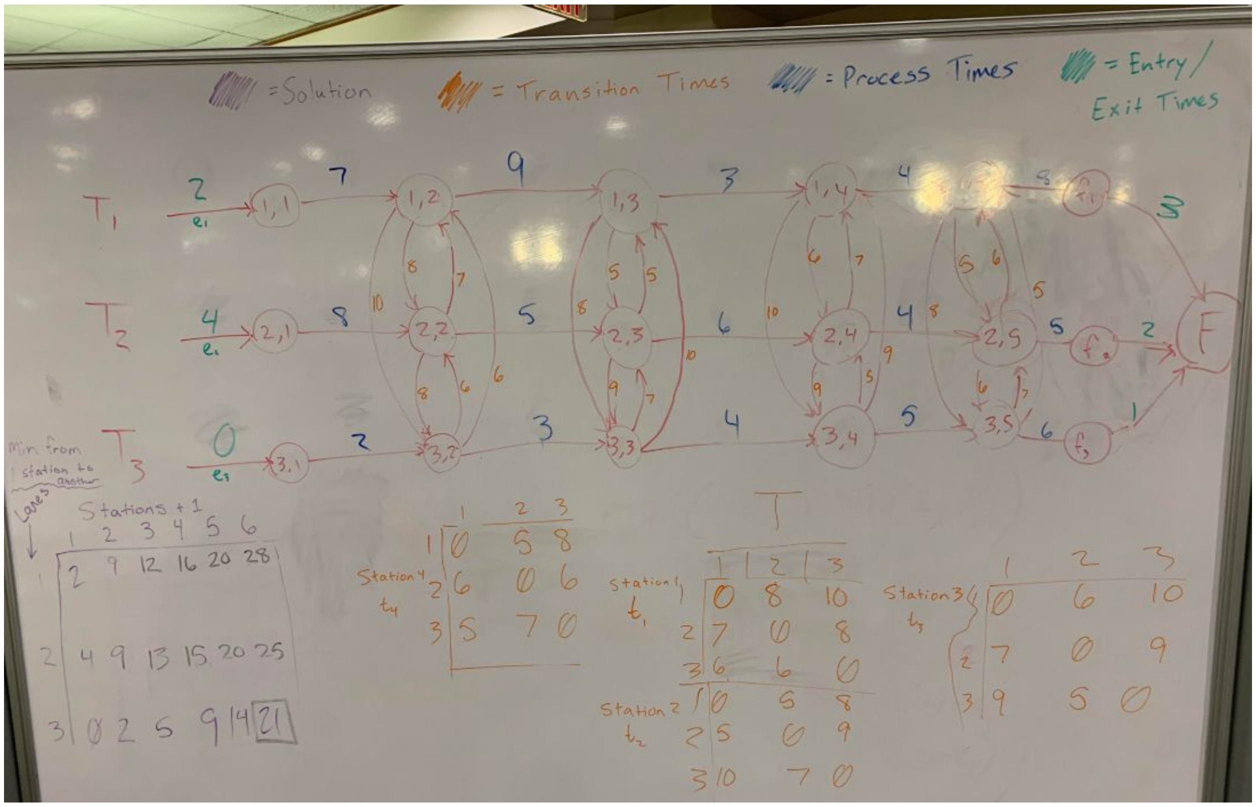
Machine 3: HP Spectre running Windows 10, 16GB of RAM, and Java as the programming language.

Testing

We filled out the transition and process times by hand first so we can follow the path by hand. The code was then ran so we can compare that output to the hand written output. The next step was to compare what is being outputted in the code to the files ProcessTimesTest.txt that were provided in the project. By comparing our results to the test file results, we drew the process out by hand and compared those results to what was given in the test files.



Above is the hand written testing for two lane.



Stations + 1

Lanes	1	2	3	4	5	6
1	2	9	12	16	20	28
2	4	9	13	15	20	25
3	0	2	5	9	14	$\boxed{21}$

Station 1 t_1

	1	2	3
1	0	8	10
2	7	0	8
3	6	6	0

Station 2 t_2

	1	2	3
1	0	5	8
2	5	0	9
3	10	7	0

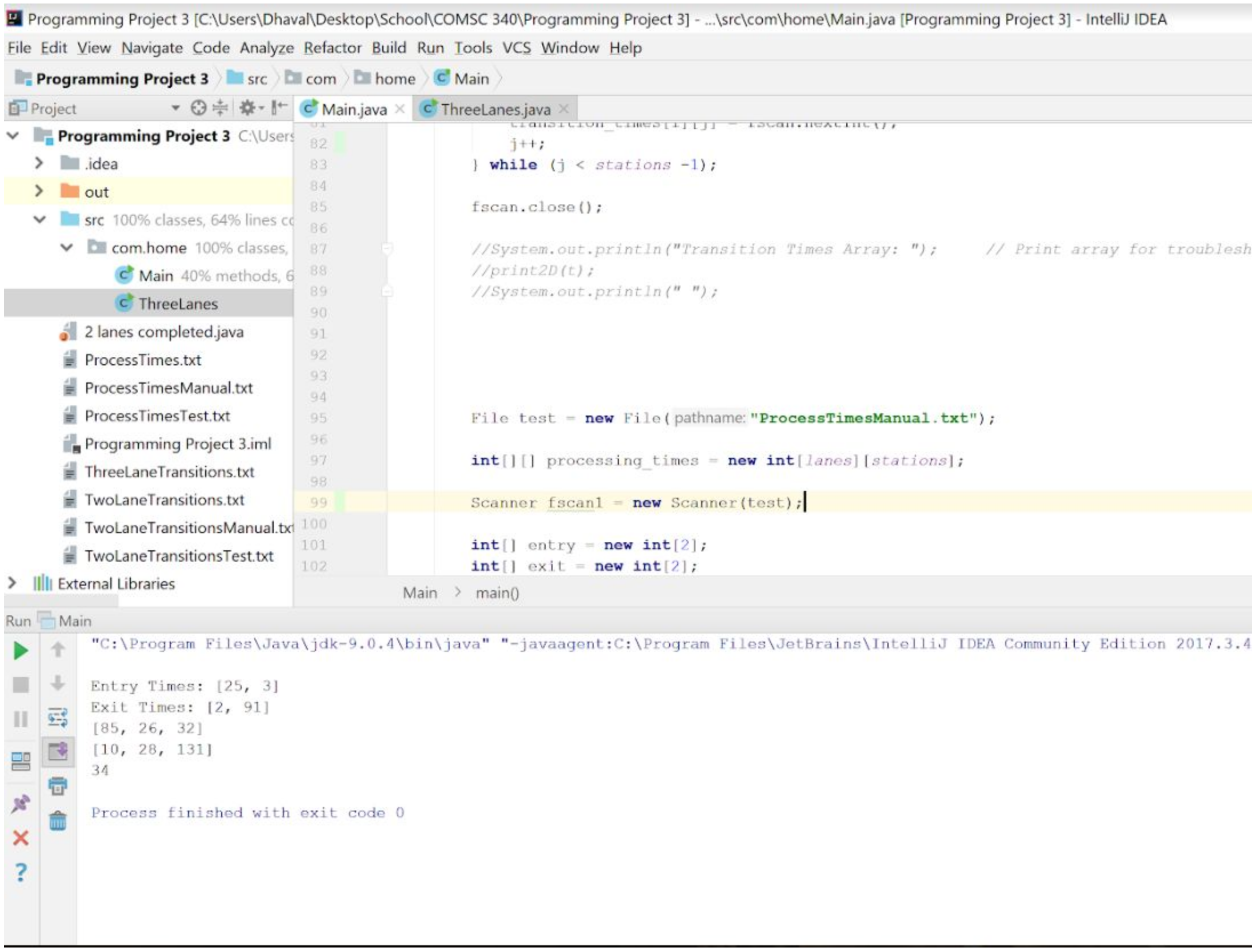
Station 3 t_3

	1	2	3
1	0	6	10
2	7	0	9
3	9	5	0

Station 4 t_4

	1	2	3
1	0	5	8
2	6	0	6
3	5	7	0

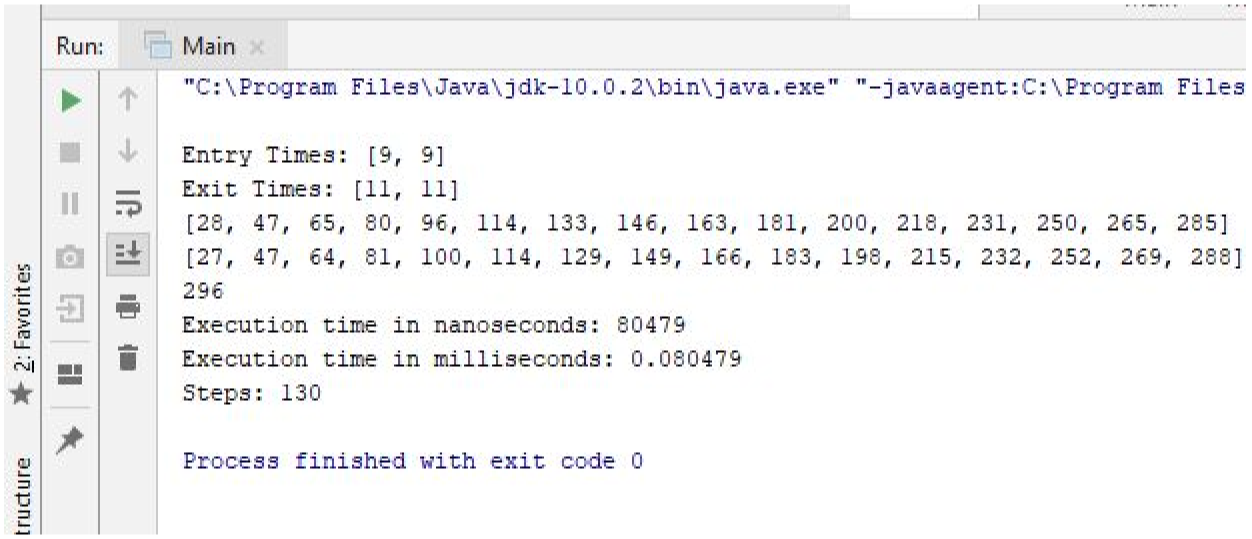
Above is the hand written testing for three lane.


Programming Project 3 [C:\Users\Dhaval\Desktop\School\COMSC 340\Programming Project 3] - ...\src\com\home\Main.java [Programming Project 3] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Programming Project 3 › src › com › home › Main
Project
Main.java
ThreeLanes.java
Programming Project 3 C:\Users
.idea
out
src 100% classes, 64% lines co
com.home 100% classes,
Main 40% methods, 6
ThreeLanes
2 lanes completed.java
ProcessTimes.txt
ProcessTimesManual.txt
ProcessTimesTest.txt
Programming Project 3.iml
ThreeLaneTransitions.txt
TwoLaneTransitions.txt
TwoLaneTransitionsManual.txt
TwoLaneTransitionsTest.txt
External Libraries
Main › main()
Run Main

```
        j++;
    } while (j < stations -1);

    fscan.close();

    //System.out.println("Transition Times Array: ");      // Print array for troublesh
    //print2D(t);
    //System.out.println(" ");




    File test = new File( pathname: "ProcessTimesManual.txt");

    int[][] processing_times = new int[lanes][stations];

    Scanner fscan1 = new Scanner(test);

    int[] entry = new int[2];
    int[] exit = new int[2];
```

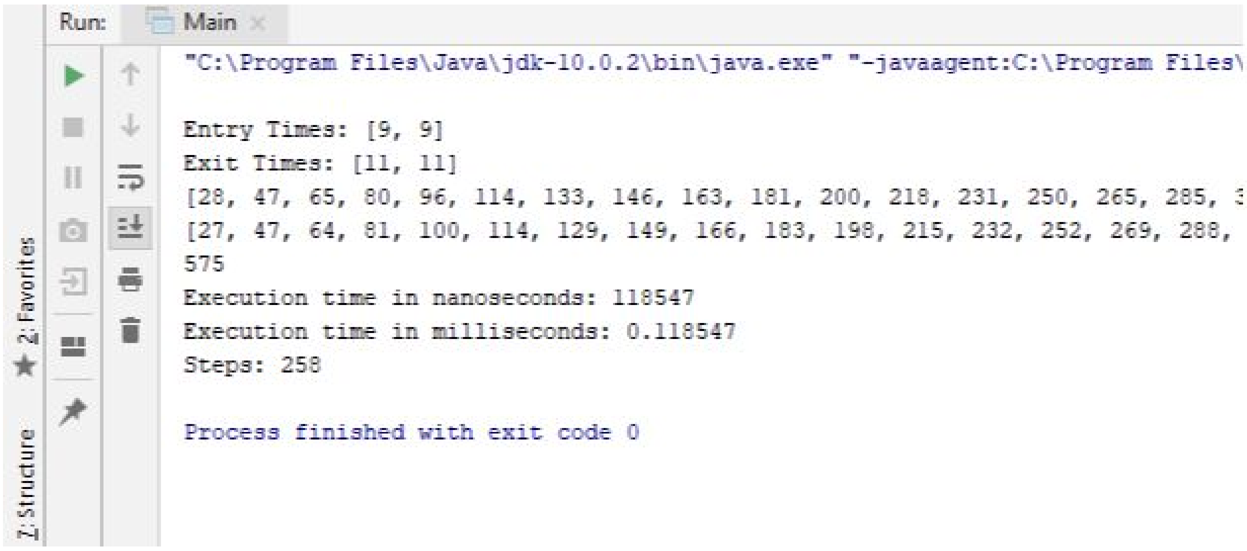
```
"C:\Program Files\Java\jdk-9.0.4\bin\java" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2017.3.4

Entry Times: [25, 3]
Exit Times: [2, 91]
[85, 26, 32]
[10, 28, 131]
34

Process finished with exit code 0
```

Above is the testing of Two Lane.



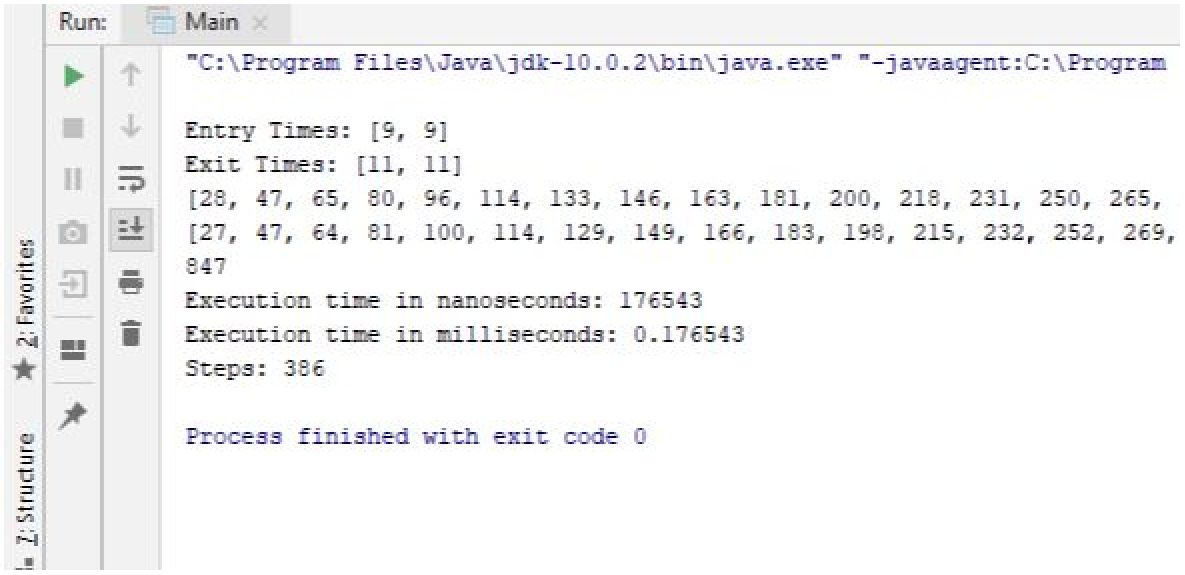
```
"C:\Program Files\Java\jdk-10.0.2\bin\java.exe" "-javaagent:C:\Program Files

Entry Times: [9, 9]
Exit Times: [11, 11]
[28, 47, 65, 80, 96, 114, 133, 146, 163, 181, 200, 218, 231, 250, 265, 285]
[27, 47, 64, 81, 100, 114, 129, 149, 166, 183, 198, 215, 232, 252, 269, 288]
296
Execution time in nanoseconds: 80479
Execution time in milliseconds: 0.080479
Steps: 130

Process finished with exit code 0
```

Above is the testing for 16 station of 2 lanes.



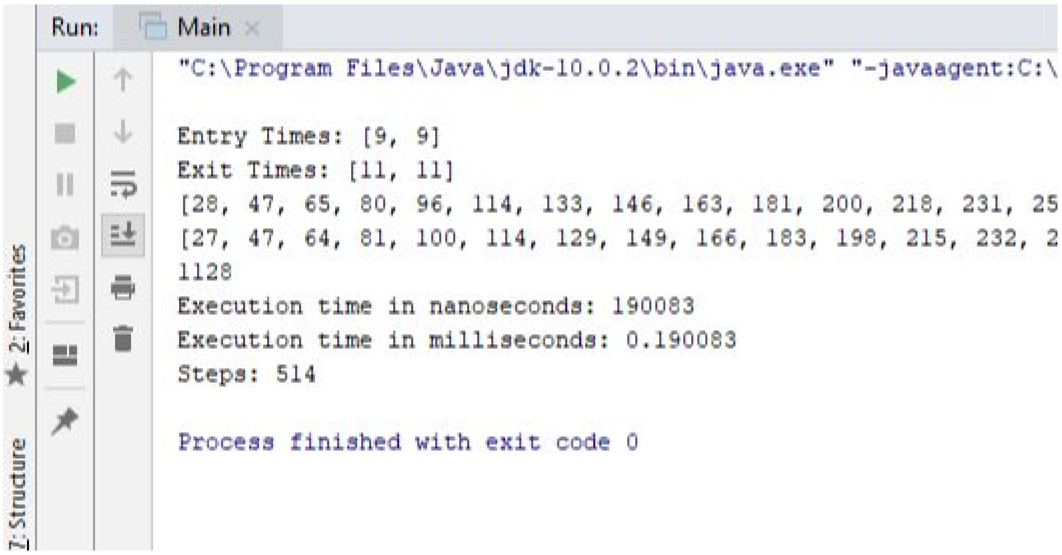
```
"C:\Program Files\Java\jdk-10.0.2\bin\java.exe" "-javaagent:C:\Program Files\

Entry Times: [9, 9]
Exit Times: [11, 11]
[28, 47, 65, 80, 96, 114, 133, 146, 163, 181, 200, 218, 231, 250, 265, 285, 3
[27, 47, 64, 81, 100, 114, 129, 149, 166, 183, 198, 215, 232, 252, 269, 288,
575
Execution time in nanoseconds: 118547
Execution time in milliseconds: 0.118547
Steps: 258

Process finished with exit code 0
```

Above is the testing for 32 stations of 2 lanes.



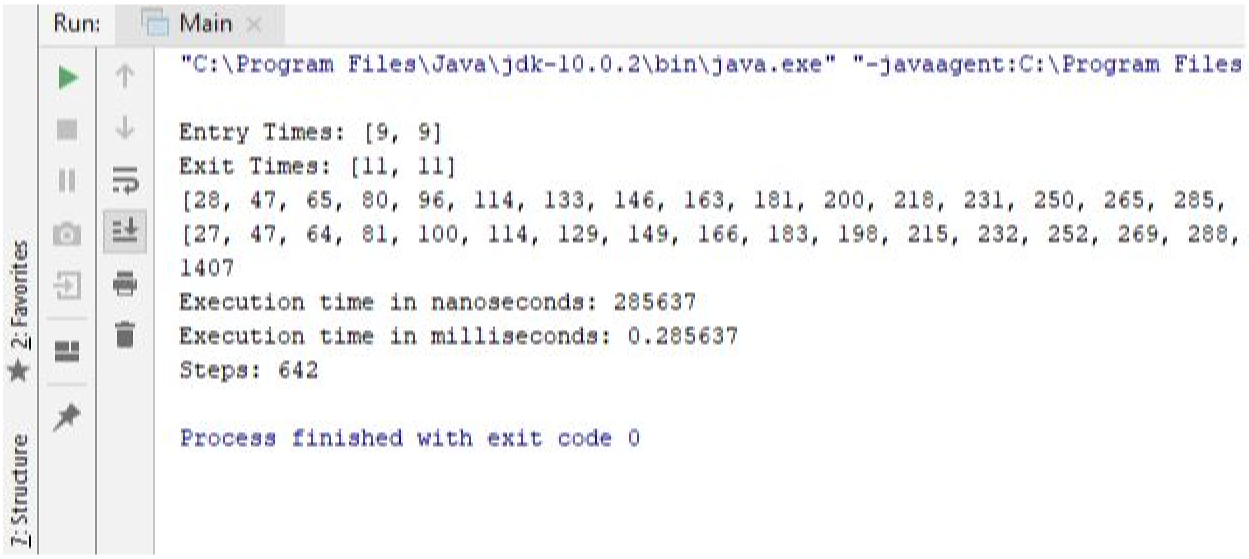
```
"C:\Program Files\Java\jdk-10.0.2\bin\java.exe" "-javaagent:C:\Program

Entry Times: [9, 9]
Exit Times: [11, 11]
[28, 47, 65, 80, 96, 114, 133, 146, 163, 181, 200, 218, 231, 250, 265,
[27, 47, 64, 81, 100, 114, 129, 149, 166, 183, 198, 215, 232, 252, 269,
847
Execution time in nanoseconds: 176543
Execution time in milliseconds: 0.176543
Steps: 386

Process finished with exit code 0
```

Above is the testing for 48 stations of 2 lanes.



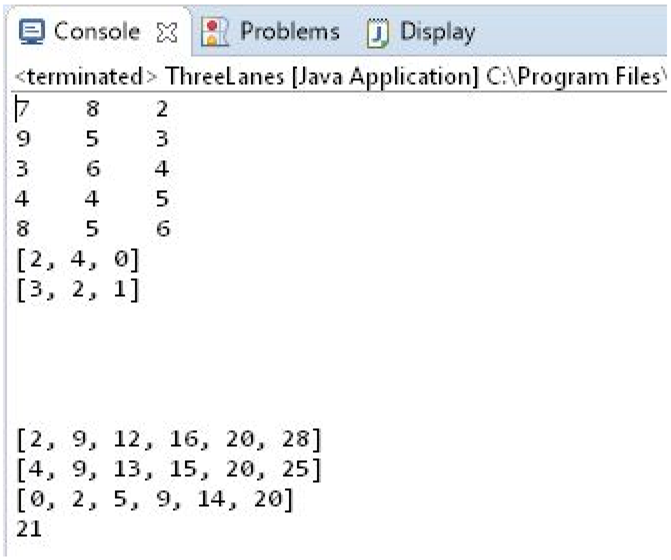
```
"C:\Program Files\Java\jdk-10.0.2\bin\java.exe" "-javaagent:C:\

Entry Times: [9, 9]
Exit Times: [11, 11]
[28, 47, 65, 80, 96, 114, 133, 146, 163, 181, 200, 218, 231, 25
[27, 47, 64, 81, 100, 114, 129, 149, 166, 183, 198, 215, 232, 2
1128
Execution time in nanoseconds: 190083
Execution time in milliseconds: 0.190083
Steps: 514

Process finished with exit code 0
```

Above is the testing for 64 stations of 2 lanes.



```
"C:\Program Files\Java\jdk-10.0.2\bin\java.exe" "-javaagent:C:\Program Files

Entry Times: [9, 9]
Exit Times: [11, 11]
[28, 47, 65, 80, 96, 114, 133, 146, 163, 181, 200, 218, 231, 250, 265, 285,
[27, 47, 64, 81, 100, 114, 129, 149, 166, 183, 198, 215, 232, 252, 269, 288,
1407
Execution time in nanoseconds: 285637
Execution time in milliseconds: 0.285637
Steps: 642

Process finished with exit code 0
```

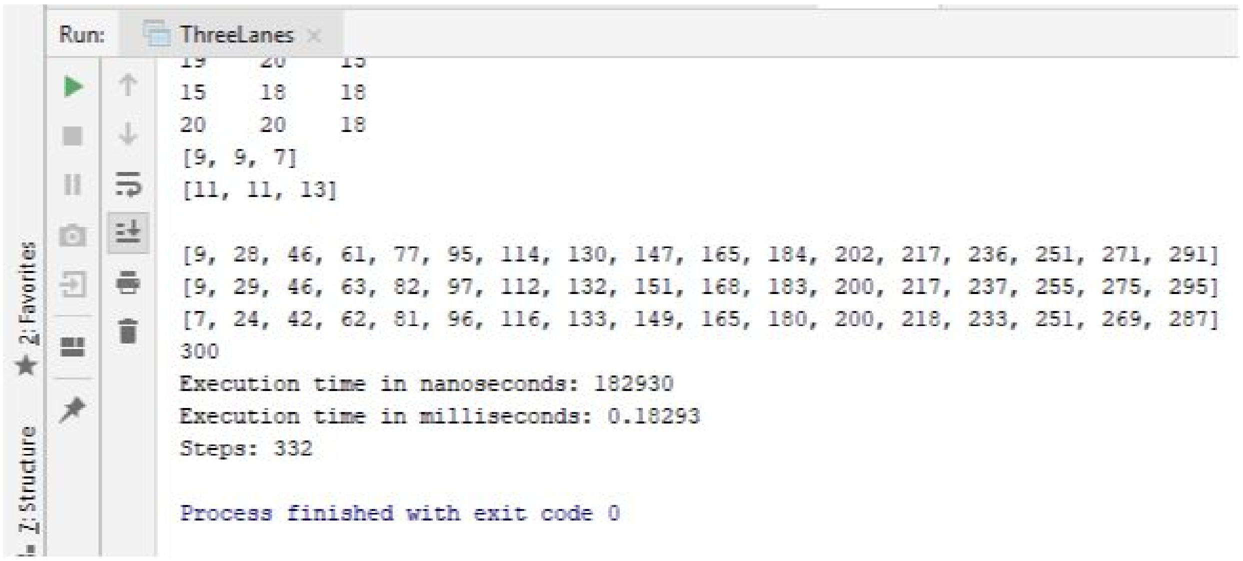
Above is the testing for 80 stations of 2 lanes.


Console Problems Display

```
<terminated> ThreeLanes [Java Application] C:\Program Files\
7    8    2
9    5    3
3    6    4
4    4    5
8    5    6
[2, 4, 0]
[3, 2, 1]




[2, 9, 12, 16, 20, 28]
[4, 9, 13, 15, 20, 25]
[0, 2, 5, 9, 14, 20]
21
```

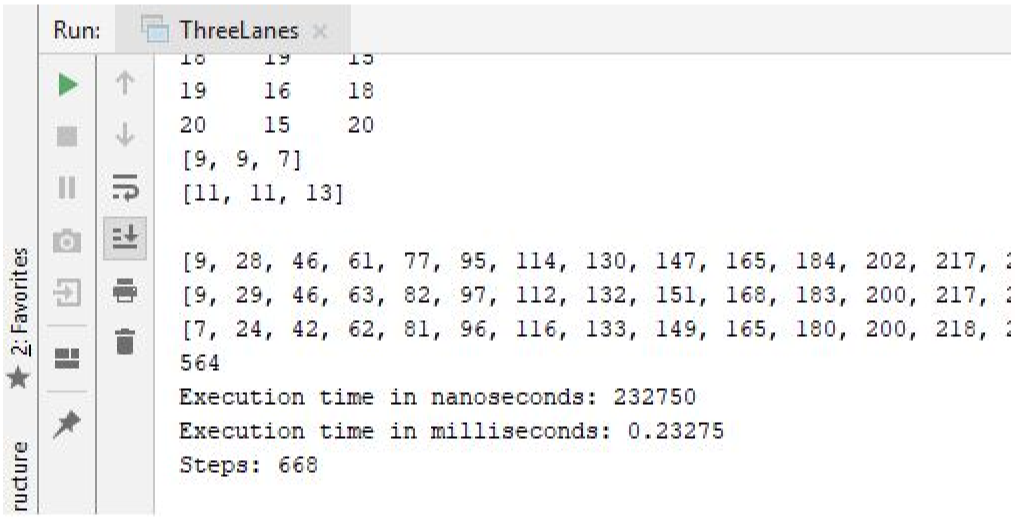
Above is the code testing for three lane.

Below is testing three lane for time complexity.



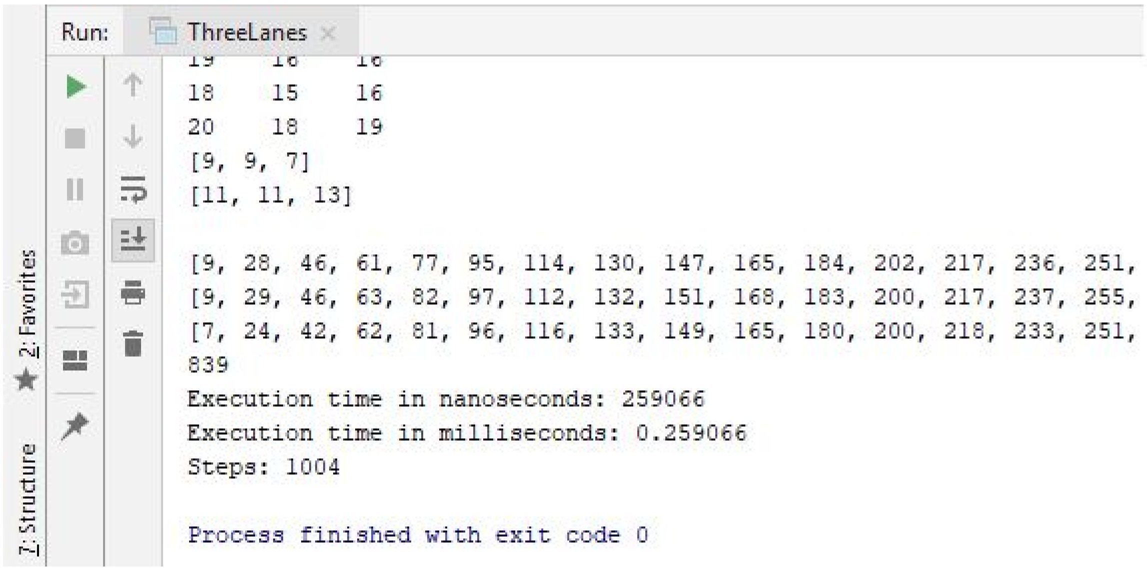
```
19    20    15
15    18    18
20    20    18
[9, 9, 7]
[11, 11, 13]

[9, 28, 46, 61, 77, 95, 114, 130, 147, 165, 184, 202, 217, 236, 251, 271, 291]
[9, 29, 46, 63, 82, 97, 112, 132, 151, 168, 183, 200, 217, 237, 255, 275, 295]
[7, 24, 42, 62, 81, 96, 116, 133, 149, 165, 180, 200, 218, 233, 251, 269, 287]
300
Execution time in nanoseconds: 182930
Execution time in milliseconds: 0.18293
Steps: 332

Process finished with exit code 0
```

Above is the testing for 16 stations of 3 lanes



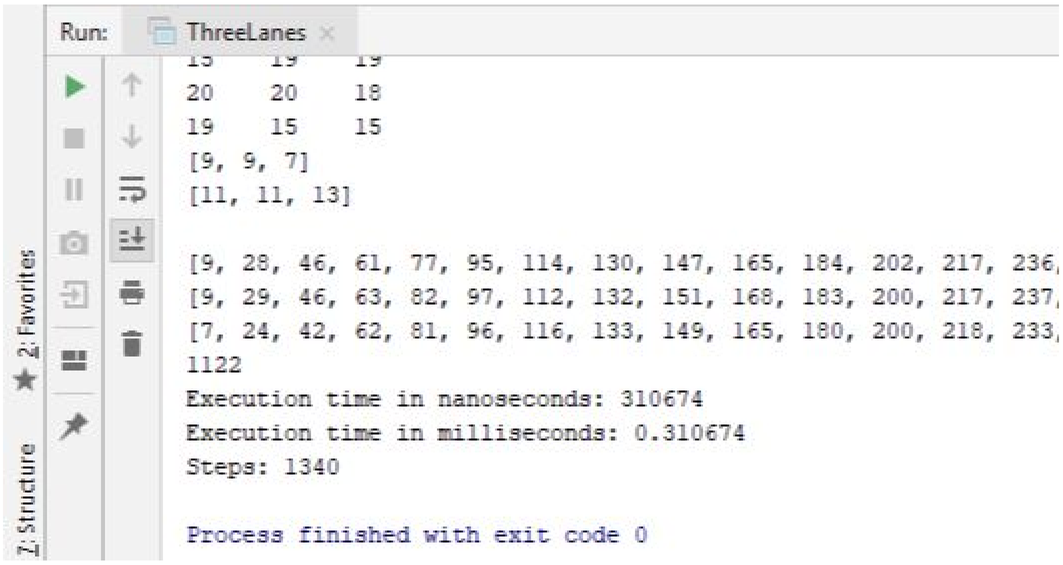
```
18    19    15
19    16    18
20    15    20
[9, 9, 7]
[11, 11, 13]

[9, 28, 46, 61, 77, 95, 114, 130, 147, 165, 184, 202, 217, 
[9, 29, 46, 63, 82, 97, 112, 132, 151, 168, 183, 200, 217, 
[7, 24, 42, 62, 81, 96, 116, 133, 149, 165, 180, 200, 218, 
564
Execution time in nanoseconds: 232750
Execution time in milliseconds: 0.23275
Steps: 668
```

Above is the testing for 32 stations of 3 lanes



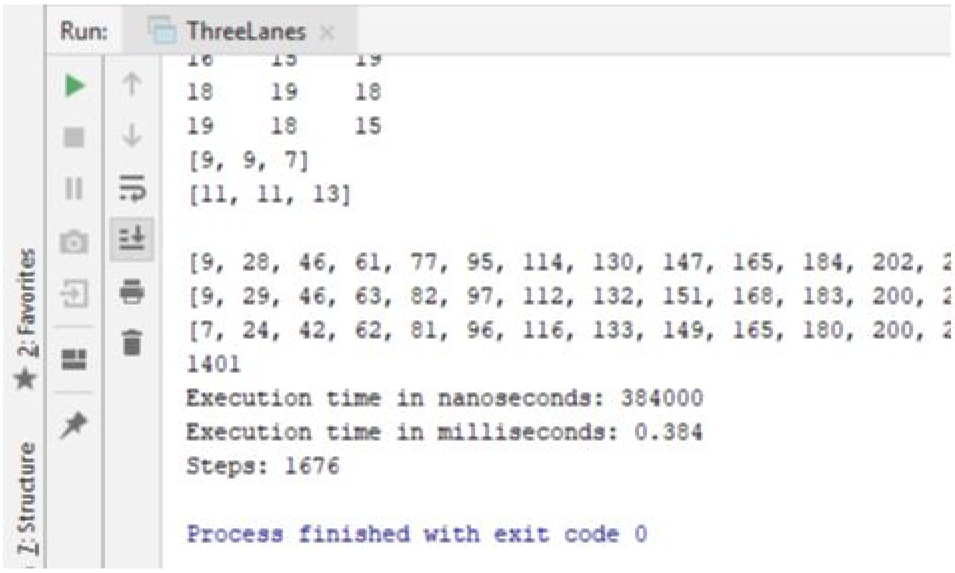
```
19    16    16
18    15    16
20    18    19
[9, 9, 7]
[11, 11, 13]

[9, 28, 46, 61, 77, 95, 114, 130, 147, 165, 184, 202, 217, 236, 251,
[9, 29, 46, 63, 82, 97, 112, 132, 151, 168, 183, 200, 217, 237, 255,
[7, 24, 42, 62, 81, 96, 116, 133, 149, 165, 180, 200, 218, 233, 251,
839
Execution time in nanoseconds: 259066
Execution time in milliseconds: 0.259066
Steps: 1004

Process finished with exit code 0
```

Above is the testing for 48 stations of 3 lanes.



```
15     19     19
20     20     18
19     15     15
[9, 9, 7]
[11, 11, 13]

[9, 28, 46, 61, 77, 95, 114, 130, 147, 165, 184, 202, 217, 236,
[9, 29, 46, 63, 82, 97, 112, 132, 151, 168, 183, 200, 217, 237,
[7, 24, 42, 62, 81, 96, 116, 133, 149, 165, 180, 200, 218, 233,
1122
Execution time in nanoseconds: 310674
Execution time in milliseconds: 0.310674
Steps: 1340

Process finished with exit code 0
```

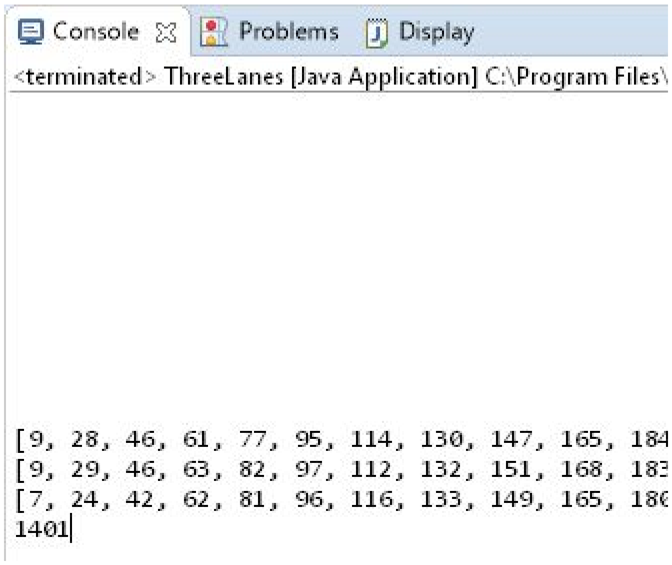
Above is the testing for 64 stations of 3 lanes



```
16     15     19
18     19     18
19     18     15
[9, 9, 7]
[11, 11, 13]

[9, 28, 46, 61, 77, 95, 114, 130, 147, 165, 184, 202, 2
[9, 29, 46, 63, 82, 97, 112, 132, 151, 168, 183, 200, 2
[7, 24, 42, 62, 81, 96, 116, 133, 149, 165, 180, 200, 2
1401
Execution time in nanoseconds: 384000
Execution time in milliseconds: 0.384
Steps: 1676

Process finished with exit code 0
```

Above is the testing for 80 stations of 3 lanes.

Summation and Conclusions



```
<terminated> ThreeLanes [Java Application] C:\Program Files\
[9, 28, 46, 61, 77, 95, 114, 130, 147, 165, 184
[9, 29, 46, 63, 82, 97, 112, 132, 151, 168, 18
[7, 24, 42, 62, 81, 96, 116, 133, 149, 165, 18
1401
```

Three Lane Min Time (80 stations)

The following is the time complexity for the two lane assembly line.

T(n) = 8n + 1

The following is the proof for the time complexity.

(2n - 2) + 4 + 2n +2+(4n - 4) + 1

p = processing array

t = (2n-2)

ex = 4

p = 2n

lane = 2

fill arrays = (4n -4)

return min = 1

f(n) ≤ c * g(n) for n ≥ k

f(n)/g(n) ≤ c for n ≥ k

T(n) = 8n + 1 = O(n)

$(8n+1)/n < (8n + n)/n$ for $n > 1$

$(8n + 1)/n < 9$ for $n > c = 9$

$(8n + 1)/n \leq 9$ for $n \geq 1$ $n_0 = 1$

Thus $8n + 1 = O(n)$

The following is the time complexity for the three lane assembly line.

$T(n) = 21n - 4$

$(9n - 9) + 6 + (3n) + 3 + (9n - 9) + 3 + 2$

t = $(9n - 9)$

ex: 6

p = $(3n)$

lane = 3

fill arrays = $(9n -9)$

add last p = 3

return min = 3

$T(n) = 21n - 4 = O(n)$

$(21n - 4)/n < (21n + n)/n$ for $n > 0$

$(21n-4)/n) < 22$ $n >0$

$c = 22$ $n_0 = 1$

Thus $21n - 4 = O(n)$

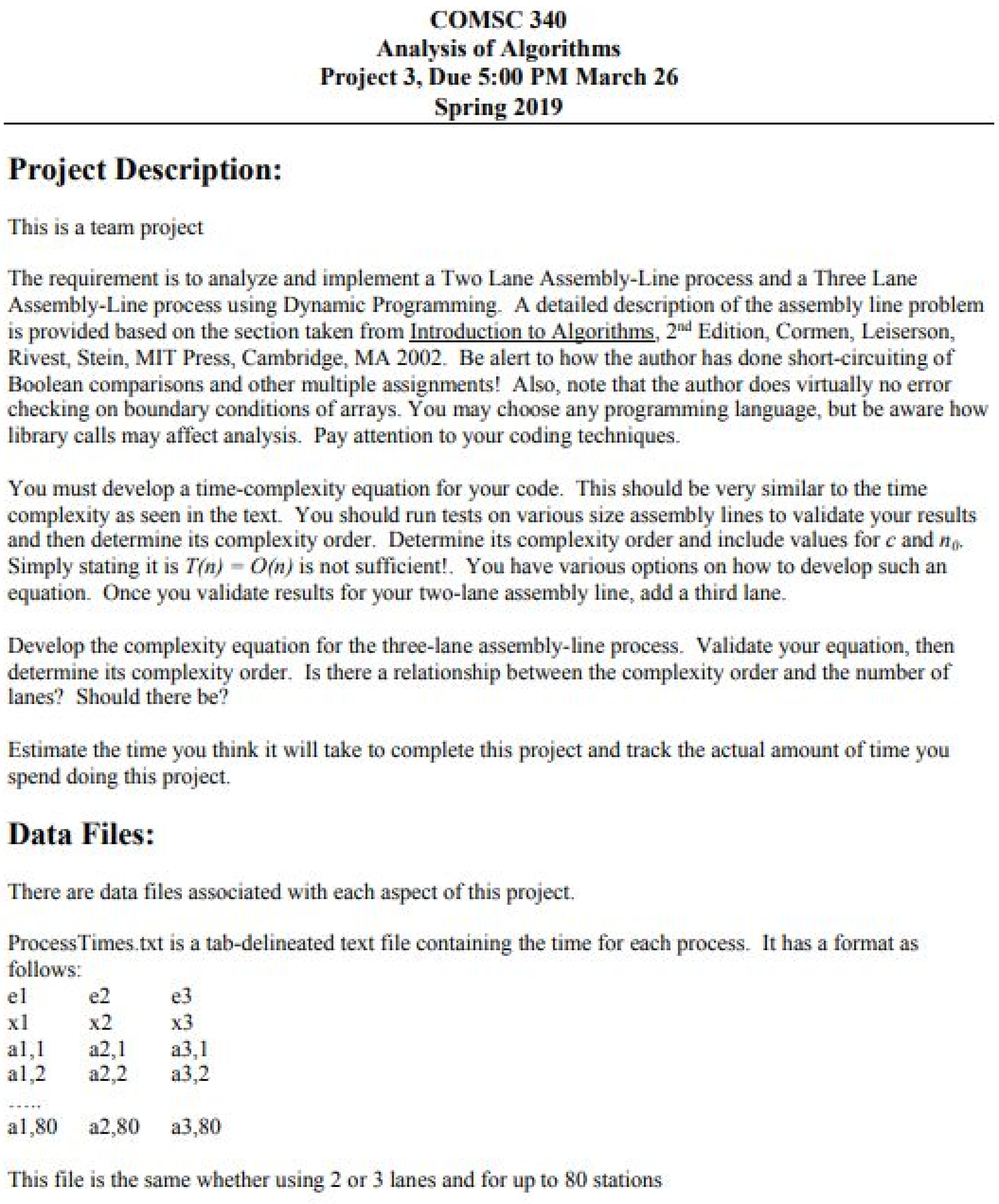
There is a relationship with complexity order in regards to lanes because the more lanes there are in an assembly line, the more time it would take to process the numbers. The time complexity is clearly larger in regards to the three lane assembly line because there needs to be more rows processed and read, while the two lane assembly line has less. This is clearly shown in the lane numbers in the proof, where each assembly line as their respective number.

References

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. (2002) *Introduction to Algorithms, 2nd Edition*, Boston, MA: McGraw-Hill Book Company

Appendices

Requirements



COMSC 340
Analysis of Algorithms
Project 3, Due 5:00 PM March 26
Spring 2019

Project Description:

This is a team project

The requirement is to analyze and implement a Two Lane Assembly-Line process and a Three Lane Assembly-Line process using Dynamic Programming. A detailed description of the assembly line problem is provided based on the section taken from Introduction to Algorithms, 2nd Edition, Cormen, Leiserson, Rivest, Stein, MIT Press, Cambridge, MA 2002. Be alert to how the author has done short-circuiting of Boolean comparisons and other multiple assignments! Also, note that the author does virtually no error checking on boundary conditions of arrays. You may choose any programming language, but be aware how library calls may affect analysis. Pay attention to your coding techniques.

You must develop a time-complexity equation for your code. This should be very similar to the time complexity as seen in the text. You should run tests on various size assembly lines to validate your results and then determine its complexity order. Determine its complexity order and include values for c and n_0. Simply stating it is $T(n) = O(n)$ is not sufficient!. You have various options on how to develop such an equation. Once you validate results for your two-lane assembly line, add a third lane.

Develop the complexity equation for the three-lane assembly-line process. Validate your equation, then determine its complexity order. Is there a relationship between the complexity order and the number of lanes? Should there be?

Estimate the time you think it will take to complete this project and track the actual amount of time you spend doing this project.

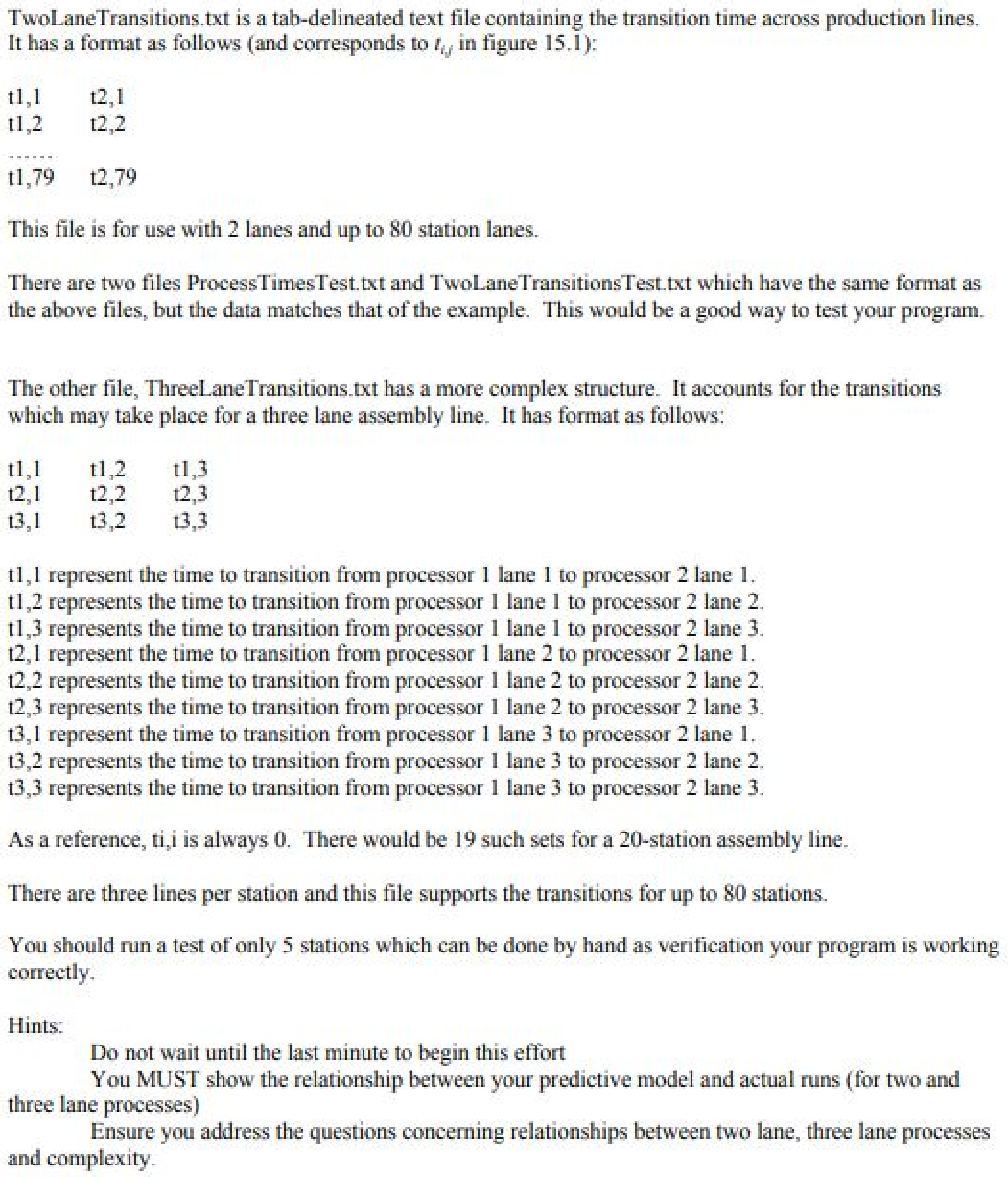
Data Files:

There are data files associated with each aspect of this project.

ProcessTimes.txt is a tab-delineated text file containing the time for each process. It has a format as follows:

```
e1      e2      e3
x1      x2      x3
a1,1    a2,1    a3,1
a1,2    a2,2    a3,2
.....
a1,80   a2,80   a3,80
```

This file is the same whether using 2 or 3 lanes and for up to 80 stations



TwoLaneTransitions.txt is a tab-delineated text file containing the transition time across production lines. It has a format as follows (and corresponds to $t_{i,j}$ in figure 15.1):

t1,1	t2,1
t1,2	t2,2
------	
t1,79	t2,79

This file is for use with 2 lanes and up to 80 station lanes.

There are two files ProcessTimesTest.txt and TwoLaneTransitionsTest.txt which have the same format as the above files, but the data matches that of the example. This would be a good way to test your program.

The other file, ThreeLaneTransitions.txt has a more complex structure. It accounts for the transitions which may take place for a three lane assembly line. It has format as follows:

t1,1	t1,2	t1,3
t2,1	t2,2	t2,3
t3,1	t3,2	t3,3

t1,1 represent the time to transition from processor 1 lane 1 to processor 2 lane 1.
t1,2 represents the time to transition from processor 1 lane 1 to processor 2 lane 2.
t1,3 represents the time to transition from processor 1 lane 1 to processor 2 lane 3.
t2,1 represent the time to transition from processor 1 lane 2 to processor 2 lane 1.
t2,2 represents the time to transition from processor 1 lane 2 to processor 2 lane 2.
t2,3 represents the time to transition from processor 1 lane 2 to processor 2 lane 3.
t3,1 represent the time to transition from processor 1 lane 3 to processor 2 lane 1.
t3,2 represents the time to transition from processor 1 lane 3 to processor 2 lane 2.
t3,3 represents the time to transition from processor 1 lane 3 to processor 2 lane 3.

As a reference, ti,i is always 0. There would be 19 such sets for a 20-station assembly line.

There are three lines per station and this file supports the transitions for up to 80 stations.

You should run a test of only 5 stations which can be done by hand as verification your program is working correctly.

Hints:

Do not wait until the last minute to begin this effort

You MUST show the relationship between your predictive model and actual runs (for two and three lane processes)

Ensure you address the questions concerning relationships between two lane, three lane processes and complexity.

Planned and Actual Schedule

Projected time spent writing code: 3 hours

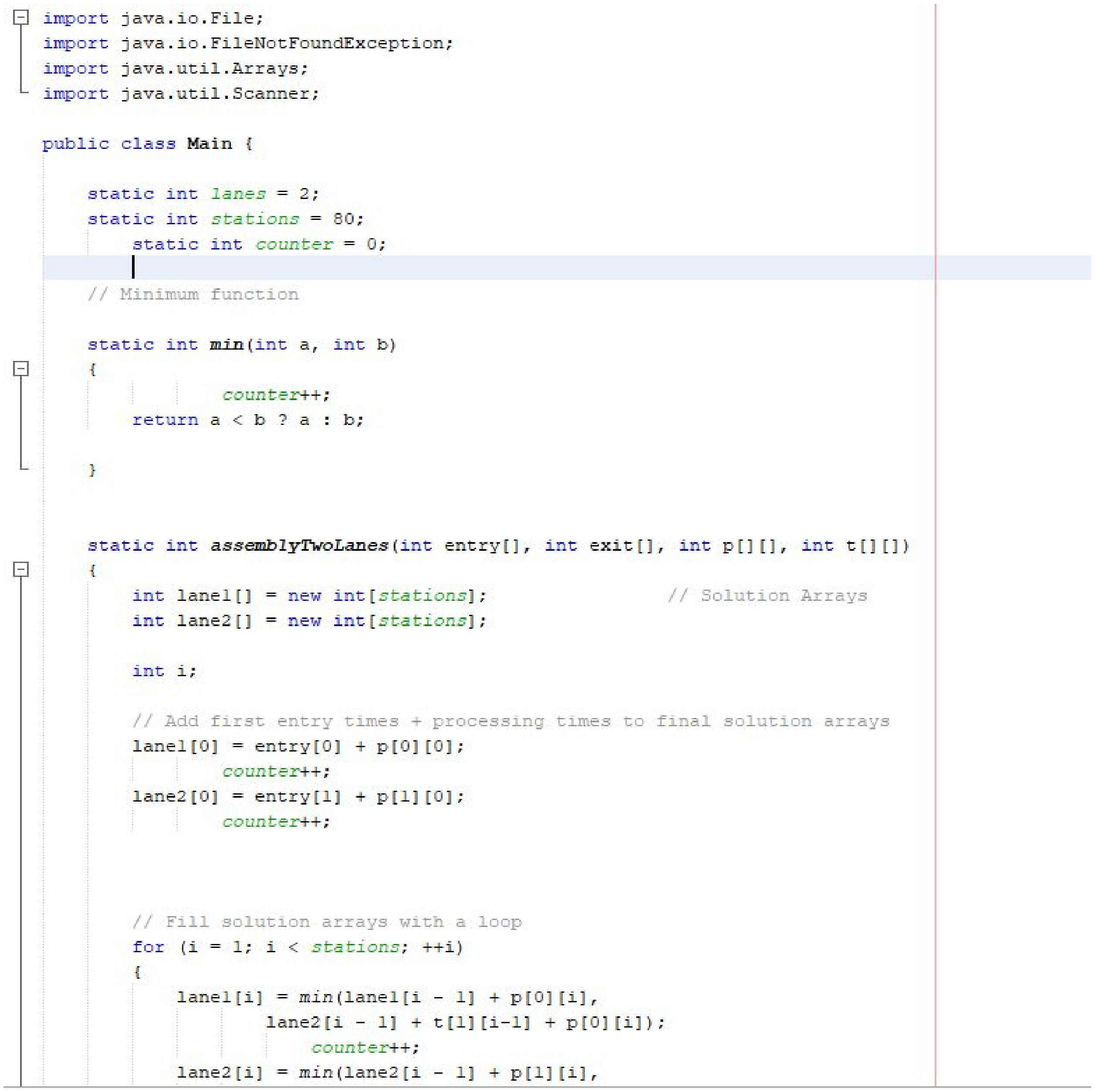
Actual time spent writing code: 10.5 hours

Projected time spent writing the report: 1 hour

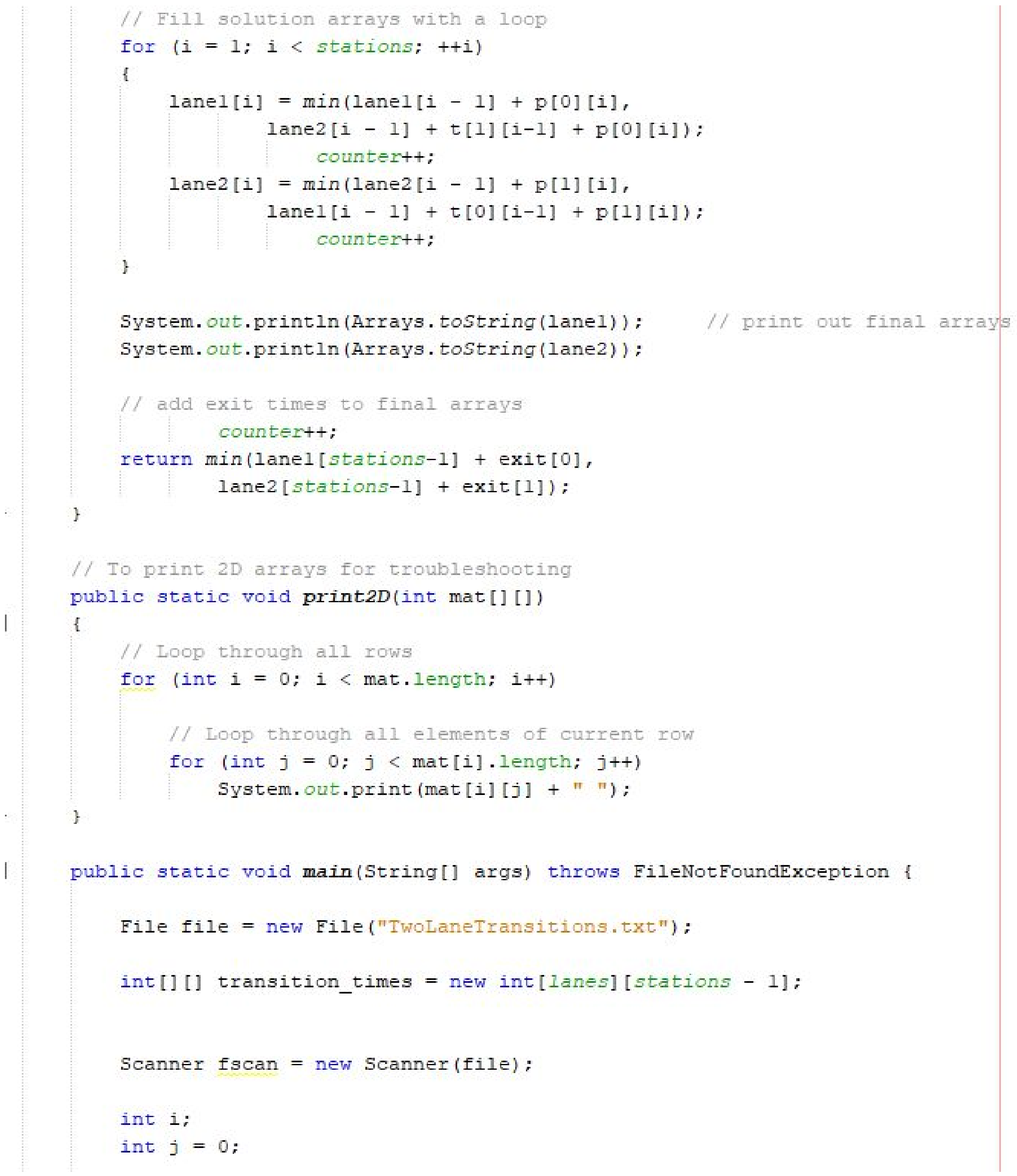
Actual time spent writing the report: 2.5 hours

Manuals

The following is the code for two lanes.

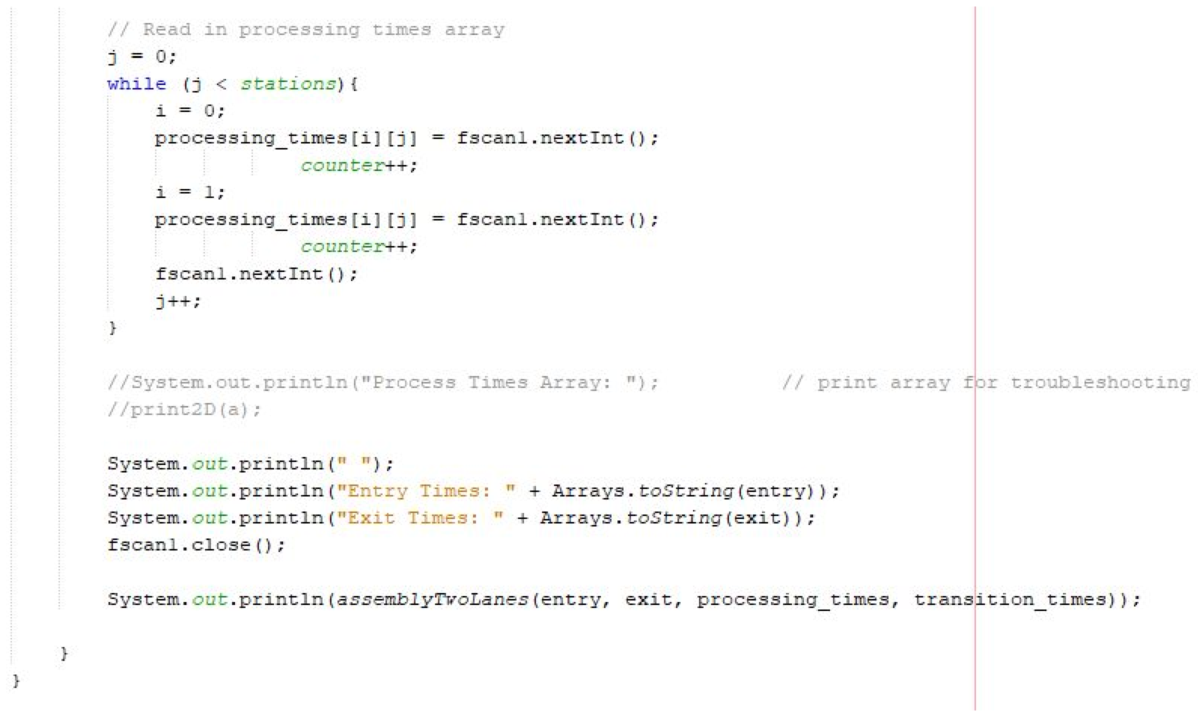


```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Arrays;
import java.util.Scanner;

public class Main {

    static int lanes = 2;
    static int stations = 80;
        static int counter = 0;

    // Minimum function

    static int min(int a, int b)
    {
                counter++;
        return a < b ? a : b;

    }


    static int assemblyTwoLanes(int entry[], int exit[], int p[][], int t[][])
    {
        int lane1[] = new int[stations];                // Solution Arrays
        int lane2[] = new int[stations];

        int i;

        // Add first entry times + processing times to final solution arrays
        lane1[0] = entry[0] + p[0][0];
                counter++;
        lane2[0] = entry[1] + p[1][0];
                counter++;



        // Fill solution arrays with a loop
        for (i = 1; i < stations; ++i)
        {
            lane1[i] = min(lane1[i - 1] + p[0][i],
                    lane2[i - 1] + t[1][i-1] + p[0][i]);
                        counter++;
            lane2[i] = min(lane2[i - 1] + p[1][i],
```

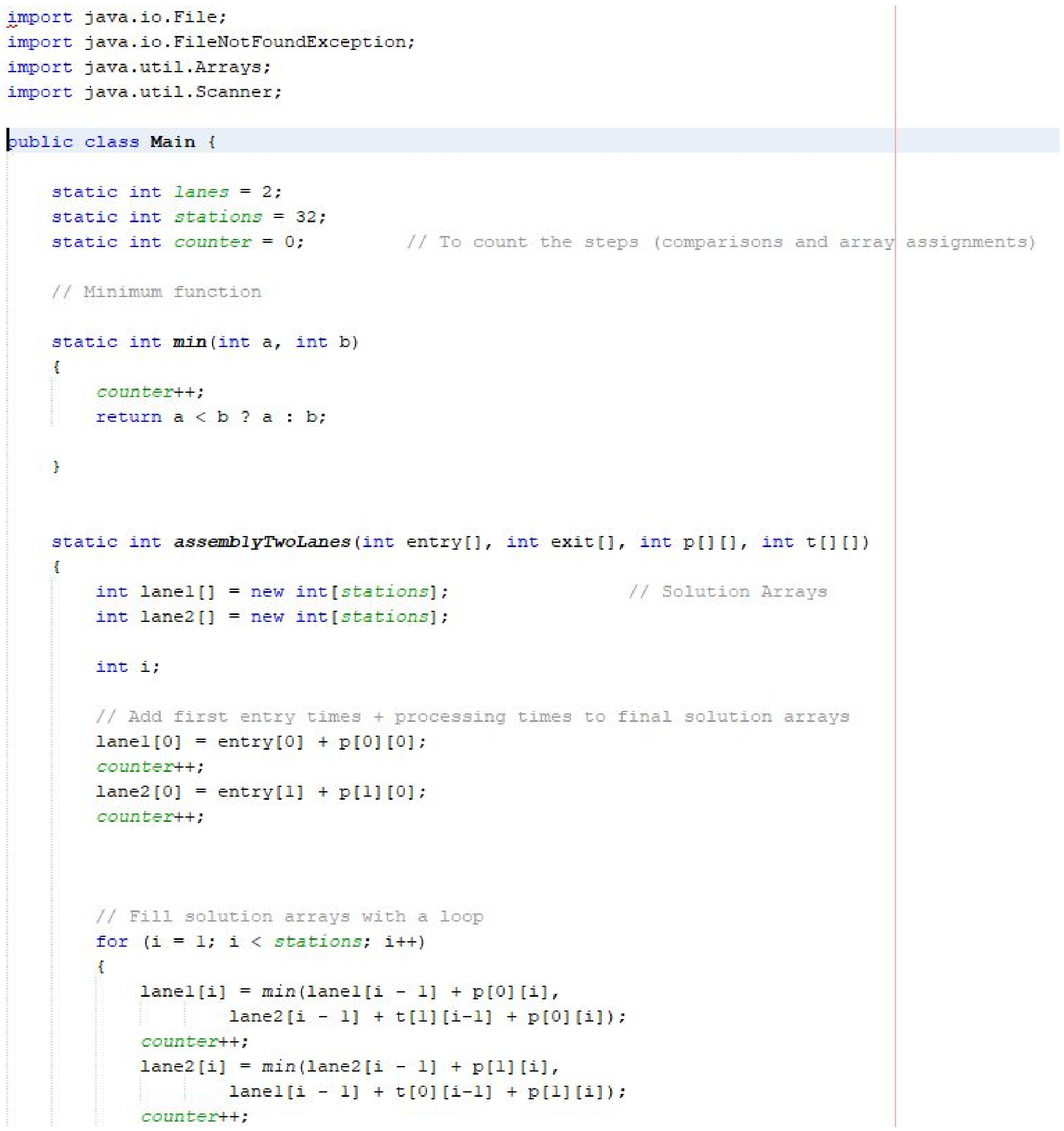


```java
        // Fill solution arrays with a loop
        for (i = 1; i < stations; ++i)
        {
            lane1[i] = min(lane1[i - 1] + p[0][i],
                    lane2[i - 1] + t[1][i-1] + p[0][i]);
                        counter++;
            lane2[i] = min(lane2[i - 1] + p[1][i],
                    lane1[i - 1] + t[0][i-1] + p[1][i]);
                        counter++;
        }

        System.out.println(Arrays.toString(lane1));     // print out final arrays
        System.out.println(Arrays.toString(lane2));

        // add exit times to final arrays
                counter++;
        return min(lane1[stations-1] + exit[0],
                lane2[stations-1] + exit[1]);
    }

    // To print 2D arrays for troubleshooting
    public static void print2D(int mat[][])
    {
        // Loop through all rows
        for (int i = 0; i < mat.length; i++)

            // Loop through all elements of current row
            for (int j = 0; j < mat[i].length; j++)
                System.out.print(mat[i][j] + " ");
    }

    public static void main(String[] args) throws FileNotFoundException {

        File file = new File("TwoLaneTransitions.txt");

        int[][] transition_times = new int[lanes][stations - 1];


        Scanner fscan = new Scanner(file);

        int i;
        int j = 0;
```

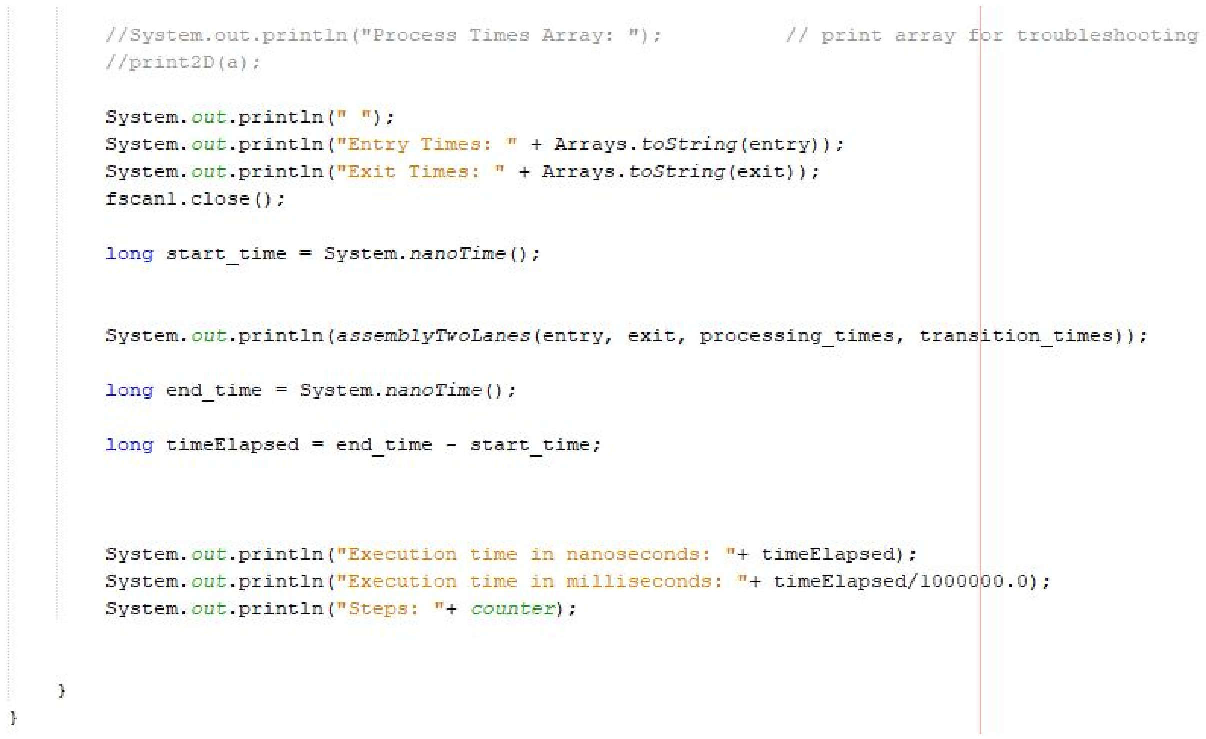
```
// Read in transition times array
do {
    i = 0;
    transition_times[i][j] = fscan.nextInt();
                counter++;
    i = 1;
    transition_times[i][j] = fscan.nextInt();
                counter++;
    j++;
} while (j < stations -1);

fscan.close();

//System.out.println("Transition Times Array: ");     // Print array for troubleshooting
//print2D(t);
//System.out.println(" ");




File test = new File("ProcessTimes.txt");

int[][] processing_times = new int[lanes][stations];

Scanner fscan1 = new Scanner(test);

int[] entry = new int[2];
int[] exit = new int[2];

// Read in entry and exit arrays
entry[0] = fscan1.nextInt();
        counter++;
entry[1] = fscan1.nextInt();
        counter++;
fscan1.nextInt();
exit[0] = fscan1.nextInt();
        counter++;
exit[1] = fscan1.nextInt();
        counter++;
fscan1.nextInt();

// Read in processing times array
j = 0;
```


```
        // Read in processing times array
        j = 0;
        while (j < stations){
            i = 0;
            processing_times[i][j] = fscan1.nextInt();
                        counter++;
            i = 1;
            processing_times[i][j] = fscan1.nextInt();
                        counter++;
            fscan1.nextInt();
            j++;
        }

        //System.out.println("Process Times Array: ");          // print array for troubleshooting
        //print2D(a);

        System.out.println(" ");
        System.out.println("Entry Times: " + Arrays.toString(entry));
        System.out.println("Exit Times: " + Arrays.toString(exit));
        fscan1.close();

        System.out.println(assemblyTwoLanes(entry, exit, processing_times, transition_times));

    }
}
```

The following is the code for the two lane with counter.

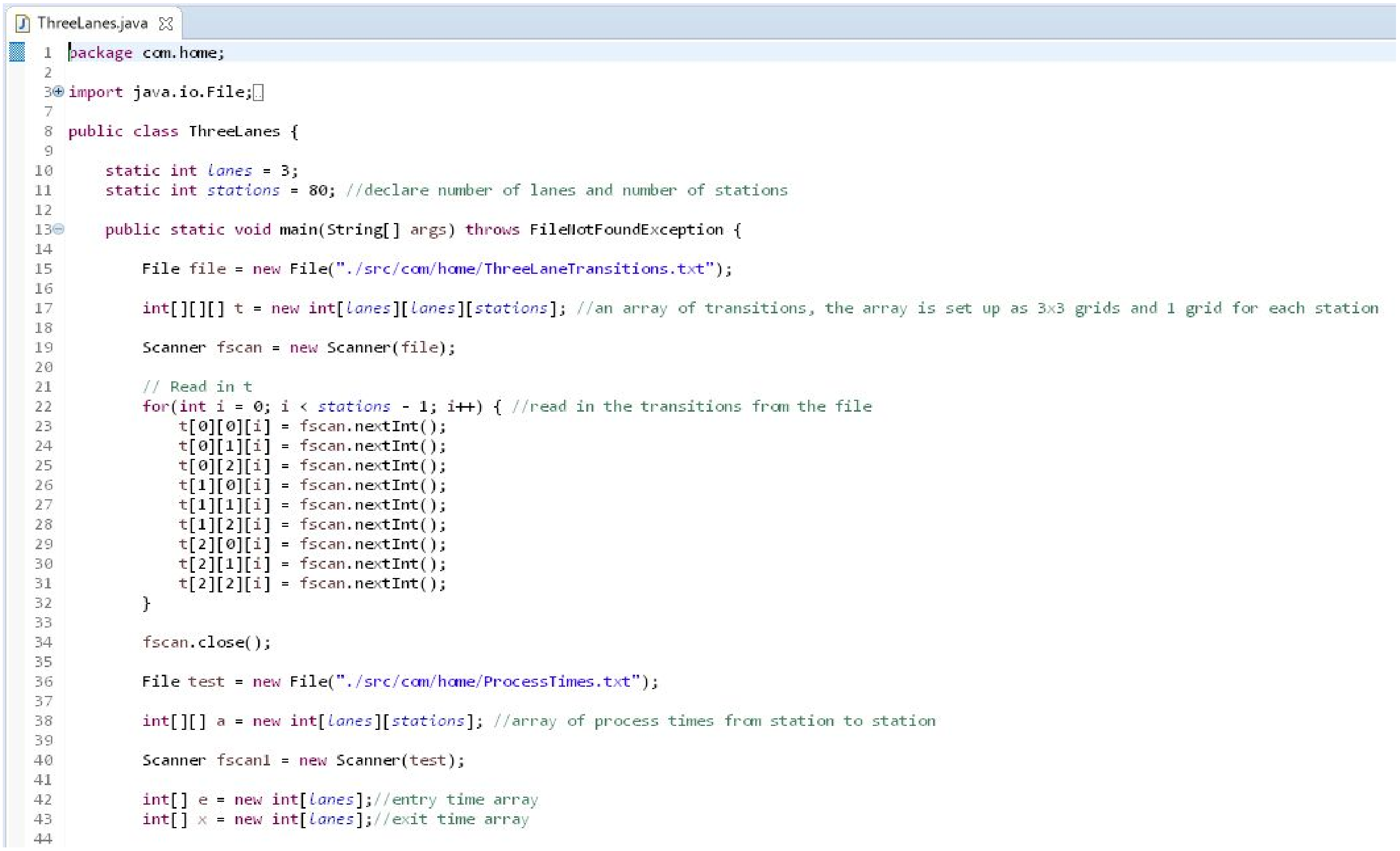


```java
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Arrays;
import java.util.Scanner;

public class Main {

    static int lanes = 2;
    static int stations = 32;
    static int counter = 0;         // To count the steps (comparisons and array assignments)

    // Minimum function

    static int min(int a, int b)
    {
        counter++;
        return a < b ? a : b;

    }


    static int assemblyTwoLanes(int entry[], int exit[], int p[][], int t[][])
    {
        int lane1[] = new int[stations];                // Solution Arrays
        int lane2[] = new int[stations];

        int i;

        // Add first entry times + processing times to final solution arrays
        lane1[0] = entry[0] + p[0][0];
        counter++;
        lane2[0] = entry[1] + p[1][0];
        counter++;



        // Fill solution arrays with a loop
        for (i = 1; i < stations; i++)
        {
            lane1[i] = min(lane1[i - 1] + p[0][i],
                    lane2[i - 1] + t[1][i-1] + p[0][i]);
            counter++;
            lane2[i] = min(lane2[i - 1] + p[1][i],
                    lane1[i - 1] + t[0][i-1] + p[1][i]);
            counter++;
        counter++;
    }

    System.out.println(Arrays.toString(lane1));     // print out final arrays
    System.out.println(Arrays.toString(lane2));

    // add exit times to final arrays

    return min(lane1[stations-1] + exit[0],
            lane2[stations-1] + exit[1]);
}

// To print 2D arrays for troubleshooting
public static void print2D(int mat[][])
{
    // Loop through all rows
    for (int i = 0; i < mat.length; i++)

        // Loop through all elements of current row
        for (int j = 0; j < mat[i].length; j++)
            System.out.print(mat[i][j] + " ");
}

public static void main(String[] args) throws FileNotFoundException {

    File file = new File("TwoLaneTransitions.txt");

    int[][] transition_times = new int[lanes][stations - 1];


    Scanner fscan = new Scanner(file);

    int i;
    int j = 0;

    // Read in transition times array
    do {
        i = 0;
        transition_times[i][j] = fscan.nextInt();
        counter++;
        i = 1;
        transition_times[i][j] = fscan.nextInt();
        counter++;
        j++;
    } while (j < stations -1);
} while (j < stations -1);

fscan.close();

//System.out.println("Transition Times Array: ");     // Print array for troubleshooting
//print2D(t);
//System.out.println(" ");





File test = new File("ProcessTimes.txt");

int[][] processing_times = new int[lanes][stations];

Scanner fscan1 = new Scanner(test);

int[] entry = new int[2];
int[] exit = new int[2];

// Read in entry and exit arrays
entry[0] = fscan1.nextInt();
counter++;
entry[1] = fscan1.nextInt();
counter++;
fscan1.nextInt();
exit[0] = fscan1.nextInt();
counter++;
exit[1] = fscan1.nextInt();
counter++;
fscan1.nextInt();

// Read in processing times array
j = 0;
while (j < stations){
    i = 0;
    processing_times[i][j] = fscan1.nextInt();
    counter++;
    i = 1;
    processing_times[i][j] = fscan1.nextInt();
    counter++;
    fscan1.nextInt();
    j++;
}
```



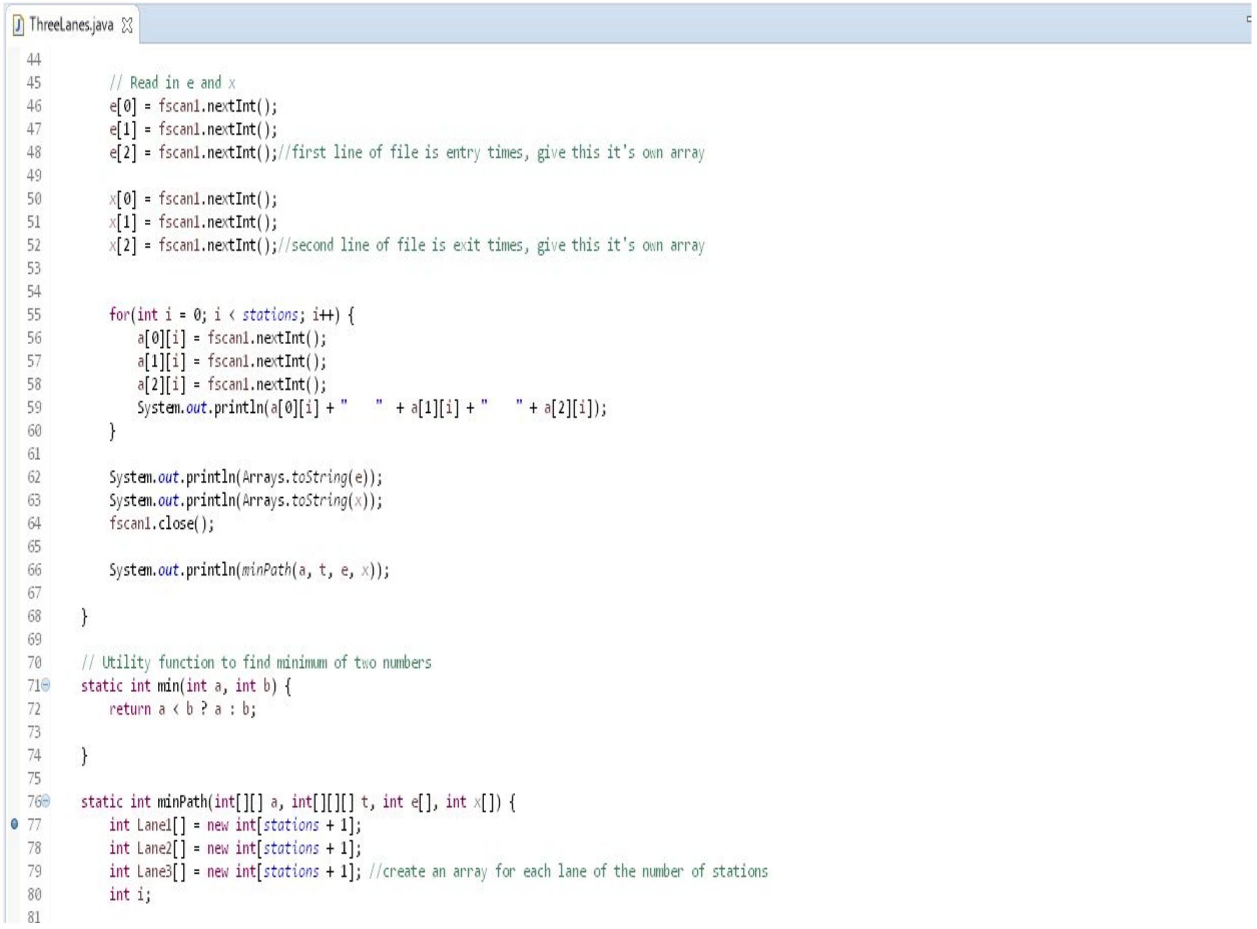
```
        //System.out.println("Process Times Array: ");          // print array for troubleshooting
        //print2D(a);

        System.out.println(" ");
        System.out.println("Entry Times: " + Arrays.toString(entry));
        System.out.println("Exit Times: " + Arrays.toString(exit));
        fscan1.close();

        long start_time = System.nanoTime();


        System.out.println(assemblyTwoLanes(entry, exit, processing_times, transition_times));

        long end_time = System.nanoTime();

        long timeElapsed = end_time - start_time;



        System.out.println("Execution time in nanoseconds: "+ timeElapsed);
        System.out.println("Execution time in milliseconds: "+ timeElapsed/1000000.0);
        System.out.println("Steps: "+ counter);


    }
}
```

The following is the code for three lanes.



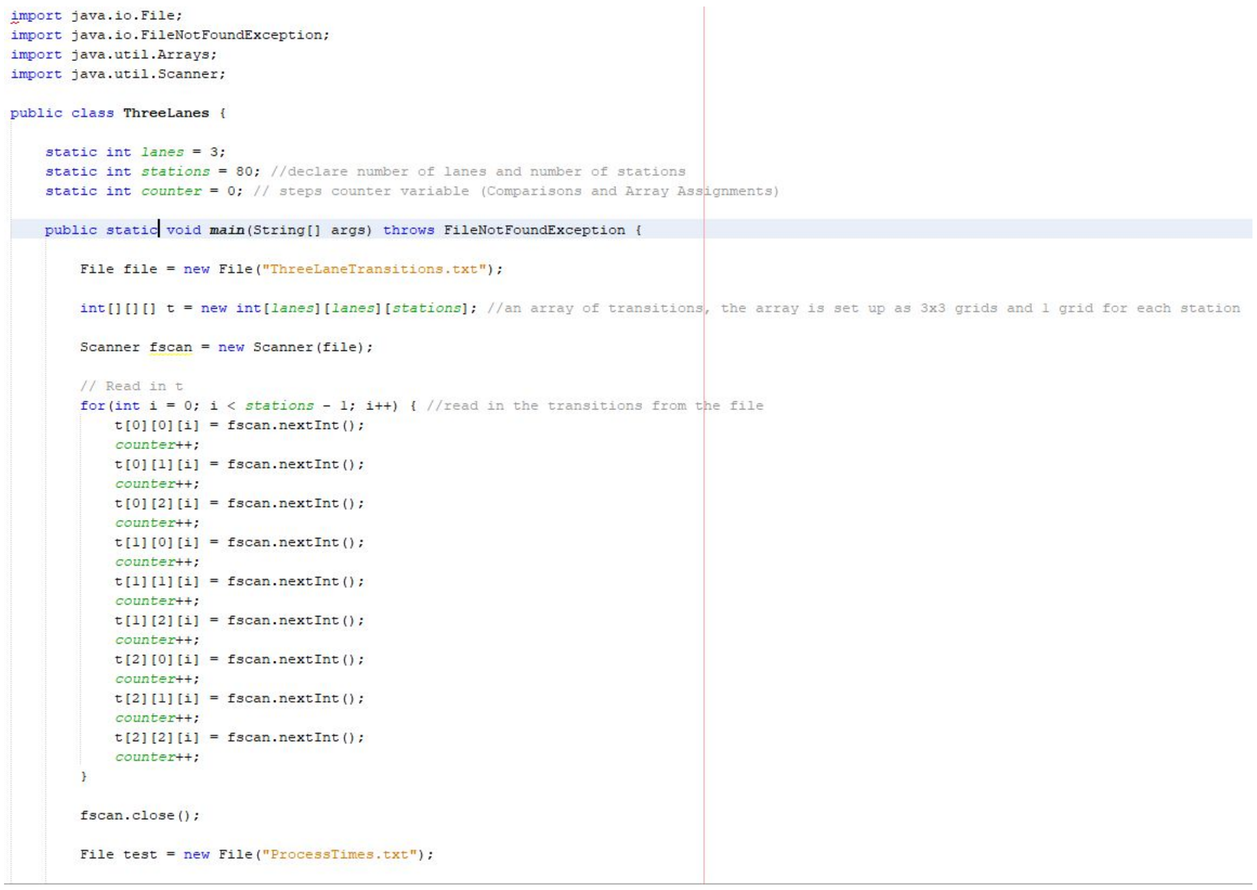
ThreeLanes.java

```java
package com.home;

import java.io.File;

public class ThreeLanes {

    static int lanes = 3;
    static int stations = 80; //declare number of lanes and number of stations

    public static void main(String[] args) throws FileNotFoundException {

        File file = new File("./src/com/home/ThreeLaneTransitions.txt");

        int[][][] t = new int[lanes][lanes][stations]; //an array of transitions, the array is set up as 3x3 grids and 1 grid for each station

        Scanner fscan = new Scanner(file);

        // Read in t
        for(int i = 0; i < stations - 1; i++) { //read in the transitions from the file
            t[0][0][i] = fscan.nextInt();
            t[0][1][i] = fscan.nextInt();
            t[0][2][i] = fscan.nextInt();
            t[1][0][i] = fscan.nextInt();
            t[1][1][i] = fscan.nextInt();
            t[1][2][i] = fscan.nextInt();
            t[2][0][i] = fscan.nextInt();
            t[2][1][i] = fscan.nextInt();
            t[2][2][i] = fscan.nextInt();
        }

        fscan.close();

        File test = new File("./src/com/home/ProcessTimes.txt");

        int[][] a = new int[lanes][stations]; //array of process times from station to station

        Scanner fscan1 = new Scanner(test);

        int[] e = new int[lanes];//entry time array
        int[] x = new int[lanes];//exit time array

```

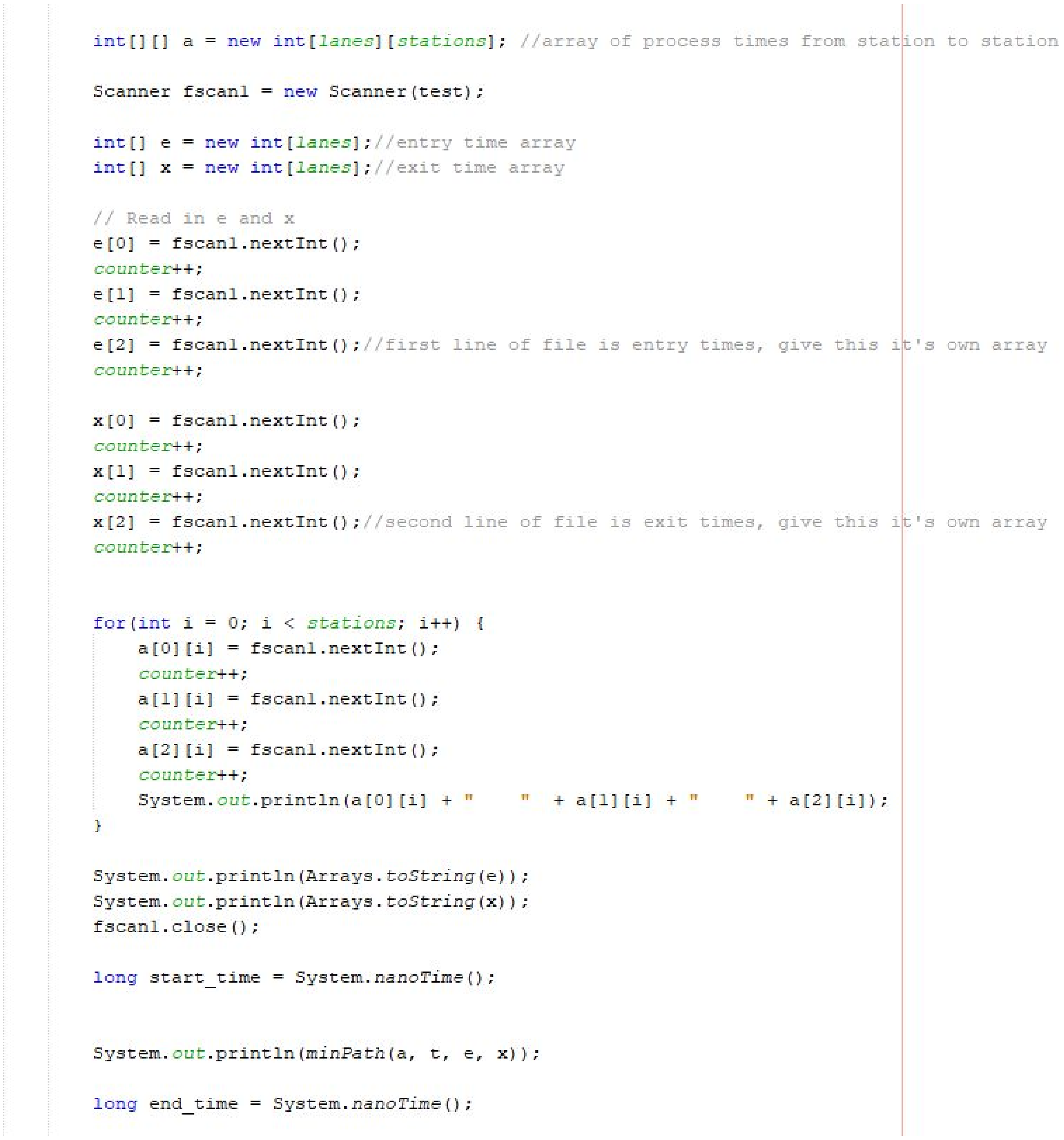


```java
        // Read in e and x
        e[0] = fscan1.nextInt();
        e[1] = fscan1.nextInt();
        e[2] = fscan1.nextInt();//first line of file is entry times, give this it's own array

        x[0] = fscan1.nextInt();
        x[1] = fscan1.nextInt();
        x[2] = fscan1.nextInt();//second line of file is exit times, give this it's own array


        for(int i = 0; i < stations; i++) {
            a[0][i] = fscan1.nextInt();
            a[1][i] = fscan1.nextInt();
            a[2][i] = fscan1.nextInt();
            System.out.println(a[0][i] + "    " + a[1][i] + "    " + a[2][i]);
        }

        System.out.println(Arrays.toString(e));
        System.out.println(Arrays.toString(x));
        fscan1.close();

        System.out.println(minPath(a, t, e, x));

    }

    // Utility function to find minimum of two numbers
    static int min(int a, int b) {
        return a < b ? a : b;

    }

    static int minPath(int[][] a, int[][][] t, int e[], int x[]) {
        int Lane1[] = new int[stations + 1];
        int Lane2[] = new int[stations + 1];
        int Lane3[] = new int[stations + 1]; //create an array for each lane of the number of stations
        int i;

```

The following is the code for three lanes with the counter.

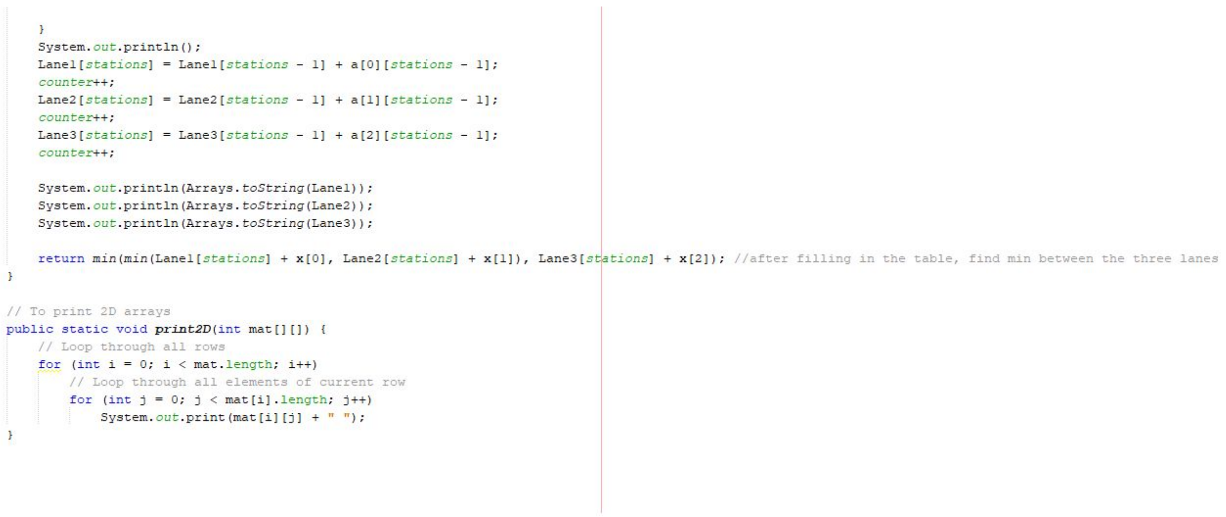


```java
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Arrays;
import java.util.Scanner;

public class ThreeLanes {

    static int lanes = 3;
    static int stations = 80; //declare number of lanes and number of stations
    static int counter = 0; // steps counter variable (Comparisons and Array Assignments)

    public static void main(String[] args) throws FileNotFoundException {

        File file = new File("ThreeLaneTransitions.txt");

        int[][][] t = new int[lanes][lanes][stations]; //an array of transitions, the array is set up as 3x3 grids and 1 grid for each station

        Scanner fscan = new Scanner(file);

        // Read in t
        for(int i = 0; i < stations - 1; i++) { //read in the transitions from the file
            t[0][0][i] = fscan.nextInt();
            counter++;
            t[0][1][i] = fscan.nextInt();
            counter++;
            t[0][2][i] = fscan.nextInt();
            counter++;
            t[1][0][i] = fscan.nextInt();
            counter++;
            t[1][1][i] = fscan.nextInt();
            counter++;
            t[1][2][i] = fscan.nextInt();
            counter++;
            t[2][0][i] = fscan.nextInt();
            counter++;
            t[2][1][i] = fscan.nextInt();
            counter++;
            t[2][2][i] = fscan.nextInt();
            counter++;
        }

        fscan.close();

        File test = new File("ProcessTimes.txt");
```



```java
int[][] a = new int[lanes][stations]; //array of process times from station to station

Scanner fscanl = new Scanner(test);

int[] e = new int[lanes];//entry time array
int[] x = new int[lanes];//exit time array

// Read in e and x
e[0] = fscanl.nextInt();
counter++;
e[1] = fscanl.nextInt();
counter++;
e[2] = fscanl.nextInt();//first line of file is entry times, give this it's own array
counter++;

x[0] = fscanl.nextInt();
counter++;
x[1] = fscanl.nextInt();
counter++;
x[2] = fscanl.nextInt();//second line of file is exit times, give this it's own array
counter++;


for(int i = 0; i < stations; i++) {
    a[0][i] = fscanl.nextInt();
    counter++;
    a[1][i] = fscanl.nextInt();
    counter++;
    a[2][i] = fscanl.nextInt();
    counter++;
    System.out.println(a[0][i] + "     " + a[1][i] + "     " + a[2][i]);
}

System.out.println(Arrays.toString(e));
System.out.println(Arrays.toString(x));
fscanl.close();

long start_time = System.nanoTime();


System.out.println(minPath(a, t, e, x));

long end_time = System.nanoTime();
```

```java
    long end_time = System.nanoTime();

    long timeElapsed = end_time - start_time;

    System.out.println("Execution time in nanoseconds: "+ timeElapsed);
    System.out.println("Execution time in milliseconds: "+ timeElapsed/1000000.0);
    System.out.println("Steps: "+ counter);


}

// Minimum Function
static int min(int a, int b) {
    counter++;
    return a < b ? a : b;

}

static int minPath(int[][] a, int[][][] t, int e[], int x[]) {
    int Lane1[] = new int[stations + 1];
    int Lane2[] = new int[stations + 1];
    int Lane3[] = new int[stations + 1]; //create an array for each lane of the number of stations
    int i;

    // Add entry times to final solution arrays
    Lane1[0] = e[0];
    counter++;

    Lane2[0] = e[1];
    counter++;

    Lane3[0] = e[2];
    counter++;

    // Fill arrays Lane1[] and Lane2[] and Lane3[]
    for (i = 1; i < stations; i++) {
        Lane1[i] = min(Lane1[i - 1] + a[0][i],
                min(Lane2[i - 1] + a[1][i] + t[1][0][i - 1],
                        Lane3[i - 1] + a[2][i] + t[2][0][i - 1]));
        counter++;

        Lane2[i] = min(Lane2[i - 1] + a[1][i],
                min(Lane1[i - 1] + a[0][i] + t[0][1][i - 1],
                        Lane3[i - 1] + a[2][i] + t[2][1][i - 1]));
        counter++;
```



```java
    }
    System.out.println();
    Lane1[stations] = Lane1[stations - 1] + a[0][stations - 1];
    counter++;
    Lane2[stations] = Lane2[stations - 1] + a[1][stations - 1];
    counter++;
    Lane3[stations] = Lane3[stations - 1] + a[2][stations - 1];
    counter++;

    System.out.println(Arrays.toString(Lane1));
    System.out.println(Arrays.toString(Lane2));
    System.out.println(Arrays.toString(Lane3));

    return min(min(Lane1[stations] + x[0], Lane2[stations] + x[1]), Lane3[stations] + x[2]); //after filling in the table, find min between the three lanes
}

// To print 2D arrays
public static void print2D(int mat[][]) {
    // Loop through all rows
    for (int i = 0; i < mat.length; i++)
        // Loop through all elements of current row
        for (int j = 0; j < mat[i].length; j++)
            System.out.print(mat[i][j] + " ");
}
```