CA4011 – Simulation Assignment
Patient Queue Simulator

Name: Glen Devlin

Student no. 12524303

Email: glen.devlin4@mail.dcu.ie

DCU School of Computing
Assignment Submission

Student Name(s): Glen Devlin

Student Number(s): 12524303

Programme: BSc in Computer Applications

Project Title: Simulation Assignment

Module code: CA4011

Lecturer: Liam Tuohey

Project Due Date: 15/3/19

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying is a grave and serious offence in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion, or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references.

I have not copied or paraphrased an extract of any length from any source without identifying the source and using quotation marks as appropriate. Any images, audio recordings, video or other materials have likewise been originated and produced by me or are fully acknowledged and identified.

This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. I have read and understood the referencing guidelines found at http://www.library.dcu.ie/citing&refguide08.pdf and/or recommended in the assignment guidelines.

I understand that I may be required to discuss with the module lecturer/s the contents of this submission.

I/me/my incorporates we/us/our in the case of group work, which is signed by all of us.

Signed: Glen Devlin

Element A1

The source code for this program was made in python and is available in the appendix.

Program Description

Single program for both multiplicative method and language random number generator. When using language random number generator, the parameters are the desired number of integers (N) and the upper range of integers (r). The lower range is always 0. When using the multiplicative congruential method, the parameters are the seed, a, b and mod.

The number of replications is also taken from user input. The program amasses a series of Chi-squared stats and out puts how many are in the good range 2√r.

Each test consisted of 100 repetitions.

Multiplicative Congruential Method

- seed = 1, a = 13, b = 0, mod = 31
  - 0% in good range, chi-squared stats were all in between 1 and 0
- seed = 0, a = 4, b = 1, mod = 9
  - 0% in good range
- seed = 11, a = 9, b = 5, mod = 12
  - 0% in good range

Uniform Random Number Generator

- N = 1000, r = 10.
  - 87% of chi-squared stats were in good range
- N = 1000, r 20
  - 84% of chi-squared stats were in good range
- N = 1000, r = 50
  - 86% of chi-squared stats were in good range

Conclusion

From these results it seems that the obvious conclusion is that the built-in language random generators perform far better to the multiplicative congruential method. The congruential method seems to repeat the same numbers the same amount of time while missing other numbers completely.

Element A2

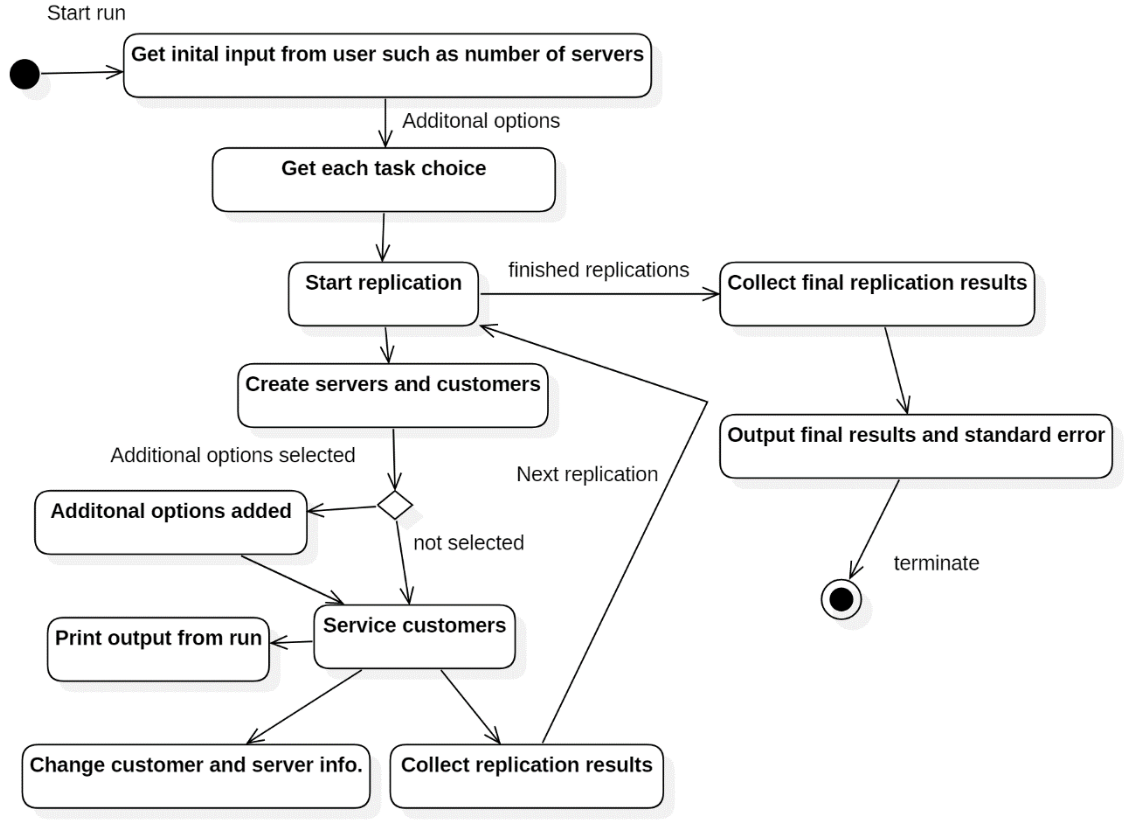
The source code for this program was made in python and is available in the appendix.

Program Description

The model for the simulation of the patient queueing system was made using python. The program takes various inputs from the user such as the number of servers, rate a server takes to see a patient and the arrival rate per hour. The program also takes a start time and end time. These are two point in which all initial appointments are scheduled. Appointments can still happen later than the end time, but it is because patients had to queue for a long time.

The program generates a list of customers based on the amount off arrival times created. Initially arrival times are distributed evenly with a bit of deviation. If the user chooses to use a Poisson distribution the times the patient times are changed to random times between the start and end point. There will still however be the same number.

Additional choices such as the ones for each task, like giving the patients gender can also be selected. Either together or by themselves. The program makes extensive use of the python time and timedelta classes. Output is printed at the end of every cycle and the final data from all replications is presented at the end of the program.



Note: For all of the below results the results are given as minutes unless otherwise specified. The standard deviation on times was 5 minutes and the start time and end time that were consistently used were 9.05 and 17.30 respectively. Every test was run with 100 replications.

2.1 Initial Simulation

(a) Deviated Regular Distribution

	$\lambda = 6, \mu = 4$	Standard Error	$\lambda = 8, \mu = 4$	Standard Error	$\lambda = 6, \mu = 2$	Standard Error
Avg. time in system (W)	15.2541	0.0097	34.6279	0.1231	152.001	0.2372
Avg. time in Queue (Wq)	0.2541	0.0097	19.6279	0.1231	122.001	0.2372
Max time in system	18.87	0.0895	55.87	0.2073	279.81	0.04503
Max time in queue	3.87	0.0895	40.87	0.2073	249.81	0.4503
Avg. no. in system (L)	1.5074	00.014	4.5304	0.0163	9.9258	0.0155
Avg. no. in queue (Lq)	0.0251	0.001	2.568	0.162	7.9668	0.0155
Proportion of idle server time	4 hours, 25 minutes	0.8105	18 minutes	0.4891	30 minutes	0
Last customer leaves	17.42		18.22		22.04	

(b) Poisson Distributed Times

	$\lambda = 6, \mu = 4$	Standard Error	$\lambda = 8, \mu = 4$	Standard Error	$\lambda = 6, \mu = 2$	Standard Error
Avg. time in system (W)	22.115	0.3239	50.127	0.9755	164.4082	1.6549
Avg. time in Queue (Wq)	7.1155	0.3239	35.127	0.9755	134.4082	1.6549
Max time in system	44.88	1.1266	92.86	1.7621	303.52	1.7051

Max time in queue	29.88	1.1266	77.86	1.7621	273.52	1.7051
Avg. no. in system (L)	2.2061	0.327	6.4137	0.124	10.6758	0.109
Avg. no. in queue (Lq)	0.709	0.0323	4.4934	0.1239	8.7279	0.1087
Proportion of idle server time	4 hrs, 16 mins	3.5237	44 min, 14 sec	3.062	39 mins, 8 sec	1.7385
Last customer leaves	17.44		18.41		22.19	

Overall the Poisson distribution led to mostly higher times. The average time in the system was higher and the average time in the queue was far higher for the Poisson distributed times. The maximum time the patient spent in the system and queue was also higher. This is all understandable as patient arrival times could all clump together leading to a bottleneck situation.

The average amount of patients in the system and queue was only slightly higher for the Poisson times. This may be because it is the same number of patients over the same number of minutes in a day. The proportion of the time both servers were idle was stable as either way there was the same number of patients with the same amount of time spent on them. The time the last patient left was also quite consistent.

<u>Queue Theory checks</u>

$\lambda = 6$, $\mu = 4$

W = 1.5, average time in system was 22 minutes which was close to a quarter of an hour

Wq = 0.37,

L = -3

Lq = 0.0617, close to my estimate of .709

P0 = 4.5, if 1 is equal to one hour, then I did have servers being idle for close to 4 and a half hours.

2.2 Patients have Genders

(a) Deviated Regular Distribution

	λ = 6, μ = 4	Standard Error	λ = 8, μ = 4	Standard Error	λ = 6, μ = 3	Standard Error
Avg. time in system (W)	23.1994	0.362	56.9581	1.1612	53.8818	1.138
Avg. time in Queue (Wq)	8.1994	0.362	41.9581	1.1612	33.8818	115.75
Max time in system	47.06	1.072	130.62	3.446	115.75	3.152
Max time in queue	32.06	1.072	115.62	3.446	95.75	3.152
Avg. no. in system (L)	2.2513	0.0336	6.8558	0.1191	4.6727	0.0789
Avg. no. in queue (Lq)	0.7952	0.0344	5.04	0.1231	2.9288	0.0844
Proportion of idle server time	4 hrs, 41 min	2.113	7 hrs, 56 min	58.8635	2 hrs, 2 min	8.504
Last customer leaves	17.48		18.48		18.35	

Having servers see specific genders gave patients a slightly higher time in system and queueing time. The max time in system and queue however rose quite a bit. This may be because there are now two chances for two different queues to bottleneck. For this same reason there is a higher number of patients in the system and in the queue at any given time.

(b) Poisson Distributed Times

	λ = 6, μ = 4	Standard Error	λ = 8, μ = 4	Standard Error	λ = 6, μ = 3	Standard Error
Avg. time in system (W)	30.6782	0.6148	69.5677	1.3045	66.6949	1.2401

Avg. time in Queue (Wq)	15.6782	0.6148	54.5677	1.3045	46.6949	1.2401
Max time in system	74.11	2.1044	163.3	3.9101	151.85	3.3178
Max time in queue	59.11	2.1044	148.3	3.9101	131.85	3.3178
Avg. no. in system (L)	2.9632	0.0572	8.2816	0.1353	5.7699	0.1062
Avg. no. in queue (Lq)	1.5139	0.0576	6.4933	0.1353	4.045	0.1051
Proportion of idle server time	4 hrs, 42 min	4.7996	6 hrs, 24 mins	51.9805	4 hrs, 35 min	39.4263
Last customer leaves	17.58		19.08		18.50	

Average time in system was higher, as was average time in queue. The max time in the system and the max time in the queue were much higher. Again, this is from bottlenecks happening. The number of patients in the system and queue was slightly higher but still quite stable. However, the time the servers were idle was much higher here. The time the last patient left was a little later.

2.3 Servers Take Breaks

	λ = 6, μ = 4	Standard Error	λ = 8, μ = 4	Standard Error	λ = 6, μ = 3	Standard Error
Avg. time in system (W)	18.2253	0.1648	69.073	0.4416	56.5876	0.3762
Avg. time in Queue (Wq)	3.2253	0.1648	53.073	0.4416	36.5876	0.3762
Max time in system	34.79	0.6669	113.24	0.5523	87.97	0.3775
Max time in queue	19.79	0.6669	98.24	0.5523	67.97	0.3775
Avg. no. in system (L)	1.8028	0.0164	8.0716	0.0462	4.945	0.0303
Avg. no. in queue (Lq)	0.3191	0.0163	6.292	0.477	3.1964	0.315
Proportion of idle server time	4 hrs, 24 min	0.8463	2 hrs, 13 min	1.2249	2 hrs, 24 min	1.0704
Last customer leaves	17.40		19.19		18.47	

The time a patient spends in the system and in the queue increases by bit when there is only 6 patients and hour but by a lot when there are 8 patients an hour. The max time a patient spends in the system and the max time a patient spends in a queue is over doubled. The time that the servers spent idle was quite a bit higher across the results. This may be from the breaks or the time waiting around after breaks for a new patient to arrive. When there are 8 patients arriving every hour, the time the last patient left increased by nearly an hour in comparison to task 1.

2.4 Patients are new and regular

(a) New and regular patients

	λ = 6, μ = 4	Standard Error	λ = 8, μ = 4	Standard Error	λ = 6, μ = 3	Standard Error
Avg. time in system (W)	17.3924	0.026	51.019	0.1386	45.0902	0.0963
Avg. time in Queue (Wq)	0.9218	0.026	34.9916	0.1386	23.1294	0.0963
Max time in system	31.38	0.1523	96.6	0.3143	84.08	0.3589
Max time in queue	9.5	0.2533	77.11	0.3107	48.69	0.1376
Avg. no. in system (L)	1.7115	0.0028	6.2606	0.018	4.0561	0.0093
Avg. no. in queue (Lq)	0.0907	0.0912	4.2939	0.0176	2.0885	0.009
Proportion of idle server time	4 hrs, 24 min	0.9288	1 hrs, 32 min	0.6595	1 hr, 47 min	0.5747
Last customer leaves	17.39		18.59		18.29	

For this task the proportion of patients that were new was 20%.

The results are very similar to part 3. This is surprising as some patients would been in service for a longer time.

(b) All new patient at front

	λ = 6, μ = 4	Standard Error	λ = 8, μ = 4	Standard Error	λ = 6, μ = 3	Standard Error
Avg. time in system (W)	33.001	0.1267	129.34	0.208	112.1018	0.2349
Avg. time in Queue (Wq)	15.0598	0.1267	111.4633	0.208	88.1802	0.2349
Max time in system	73.23	0.3287	159.55	0.2556	126.66	0.2727

Max time in queue	52.58	0.3599	144.55	0.2556	106.65	0.2724
Avg. no. in system (L)	3.2635	0.0144	14.2837	0.023	9.2064	0.0193
Avg. no. in queue (Lq)	1.4896	0.0133	12.3095	0.023	7.2419	0.0193
Proportion of idle server time	4 hrs, 24 min	0.9397	3 hrs, 45 min	0.0281	3 hrs, 40 min	0
Last customer leaves	17.40		20.04		19.24	

Compared to the first part of this task the average time in the system is much higher here. This may be from the first few patients causing a queue which led to a knock-on effect through the day. Both the average time in the system and average time in the queue are higher, but the max time in system and queue takes huge leaps. This is from a huge queue forming early in the day. Server idle time sees an increase as does the last time a customer leaves.

2.5 Experienced servers

(a) 1 experienced server

	λ = 6, μ = 4	Standard Error	λ = 8, μ = 4	Standard Error	λ = 6, μ = 3	Standard Error
Avg. time in system (W)	7.7624	0.0195	11.6389	0.0391	14.5733	0.0456
Avg. time in Queue (Wq)	0.7624	0.0195	4.6389	0.0391	4.5733	0.0456
Max time in system	12.95	0.1019	16.98	0.0141	19.96	0.0243
Max time in queue	5.95	0.1019	9.98	0.0141	9.96	0.0243
Avg. no. in system (L)	0.7773	0.0022	1.6441	0.0059	1.4419	0.0047
Avg. no. in queue (Lq)	0.0764	0.0022	0.6554	0.0057	0.4525	0.0046
Proportion of idle server time	2 hrs, 31 min	0.4054	4 min, 49 sec	0.3229	4 min, 30 sec	0.3003
Last customer leaves	17.33		17.41		17.39	

(b) 3 servers, 1 is experienced

	λ = 6, μ = 4	Standard Error	λ = 8, μ = 4	Standard Error	λ = 6, μ = 3	Standard Error
Avg. time in system (W)	8.7396	0.0298	10.0082	0.0217	13.4549	0.0287
Avg. time in Queue (Wq)	0	0	0.0022	0.0008	0	0
Max time in system	15	0	15.1	0.0414	20	0
Max time in queue	0	0	0.13	0.0442	0	0
Avg. no. in system (L)	0.874	0.003	1.4208	0.0033	14 hrs, 13 min	2.4086

Avg. no. in queue (Lq)	0	0	0.0003	0.0001	1.3344	0.034
Proportion of idle server time	18 hrs, 1 min		13 hrs, 28 min	2.0903	0	0
Last customer leaves	17.34		17 hrs, 38 min		17.37	

For this task server 1's servicing time was set to half of that of the other servers. For example, if it took server 2 and 3 20 minutes to see a customer, then it took server 1 only 10 minutes.

Compared to each part of this task to each other, the queueing time was non-existent when there were 3 servers on. This is because server 1 could see almost all patients by themselves. Server 3 was in fact rarely, if ever used. This had the knock-on effect of drastically increasing server idle times as server 3 spent most of the day doing nothing.

This may be the only test that resulted in better results that task 1. Customer time in the system extremely reduced as server 1 could do the vast amount of the work in half the time.

Conclusion

Overall, the results seem to be as expected. Standard error seems fairly stable across all results and only takes a select few jumps up to higher number. Results such as average time in system increase in areas where you would expect them too. In areas where queues build up. The task that showed the most promising results was task 5. Having an experienced server made a huge impact on seeing patients.

Tasks 2, 3, and 4 understandably increased the time patients spent in the system due to the extra constraints that servers had to work around.

Poisson distributed times also had the effect of making the servicing more efficient as they led to a lot of patients arriving at once and this led to the patients having to queue much longer and servers having to do most of their work in more concentrated pockets of the day.

Appendix

Screen shots of A2 Program

Input

```
D:\College\4th Year\Operations Research\CA A - Simulation>python Patient_Simulator.py
Number of servers: 2
Server rate: 4
Arrival Rate: 6
Enter start time: 9.05
Enter end time: 17.30
Standard Deviation in minutes: 5
Random uniform distribution (r) or Possoin Distribution (p): r
Would you like specific servers for each gender (y/n): n
Would you like the servers to have breaks (y/n): n
Would you like new and regualr customers(y/n): y
What percentage of the customers should be new: 20
Would you like all new customers moved to the front (y/n): n
Is the lead server more experienced (y/n): n
How many replications: 100
```

Final output

```
After all replications
Total average time in system:  17.3788 minutes
                 standarad error:  0.0242
Total average time in queue:  0.9082 minutes
                 standarad error:  0.0242
Average of maximum time spent in system:  31.04 minutes
                 standarad error:  0.1456
Average of maximum time spent in queue:  9.42 minutes
                 standarad error:  0.2371
Average amount of time all servers were idle:  4:25:54 minutes
                 standarad error:  0.7534
Average number in system at any given minute:  1.7088
                 standarad error:  0.0026
Average nummber in queue at any given time:  0.0897
                 standarad error:  0.0024
Avg. time last customer left:  17:40:30
```

One cycle output

```
Output from this run
Customer no.            Actual Arrival time     Admitted to server      departed server         queue time
1                       09:05:00                09:05:00 (Server 1)     09:20:00                0:00:00         , regular
2                       09:14:00                09:14:00 (Server 2)     09:29:00                0:00:00         , regular
3                       09:24:00                09:24:00 (Server 1)     09:39:00                0:00:00         , regular
4                       09:31:00                09:31:00 (Server 2)     09:46:00                0:00:00         , regular
5                       09:44:00                09:44:00 (Server 1)     09:59:00                0:00:00         , regular
6                       09:53:00                09:53:00 (Server 2)     10:08:00                0:00:00         , regular
7                       10:04:00                10:04:00 (Server 1)     10:19:00                0:00:00         , regular
8                       10:16:00                10:16:00 (Server 2)     10:31:00                0:00:00         , regular
9                       10:29:00                10:29:00 (Server 1)     10:44:00                0:00:00         , regular
10                      10:33:00                10:33:00 (Server 2)     11:03:00                0:00:00         , new
11                      10:44:00                10:44:00 (Server 1)     10:59:00                0:00:00         , regular
12                      10:55:00                10:59:00 (Server 1)     11:14:00                0:04:00         , regular
13                      11:08:00                11:08:00 (Server 2)     11:23:00                0:00:00         , regular
14                      11:18:00                11:18:00 (Server 1)     11:33:00                0:00:00         , regular
15                      11:28:00                11:28:00 (Server 2)     11:43:00                0:00:00         , regular
16                      11:34:00                11:34:00 (Server 1)     11:49:00                0:00:00         , regular
17                      11:42:00                11:43:00 (Server 2)     11:58:00                0:01:00         , regular
18                      11:57:00                11:57:00 (Server 1)     12:12:00                0:00:00         , regular
19                      12:04:00                12:04:00 (Server 2)     12:19:00                0:00:00         , regular
20                      12:19:00                12:19:00 (Server 1)     12:49:00                0:00:00         , new
21                      12:22:00                12:22:00 (Server 2)     12:37:00                0:00:00         , regular
22                      12:39:00                12:39:00 (Server 2)     12:54:00                0:00:00         , regular
23                      12:47:00                12:49:00 (Server 1)     13:04:00                0:02:00         , regular
24                      12:57:00                12:57:00 (Server 2)     13:12:00                0:00:00         , regular
25                      13:02:00                13:04:00 (Server 1)     13:19:00                0:02:00         , regular
26                      13:13:00                13:13:00 (Server 2)     13:28:00                0:00:00         , regular
27                      13:20:00                13:20:00 (Server 1)     13:35:00                0:00:00         , regular
28                      13:34:00                13:34:00 (Server 2)     13:49:00                0:00:00         , regular
29                      13:48:00                13:48:00 (Server 1)     14:03:00                0:00:00         , regular
30                      13:52:00                13:52:00 (Server 2)     14:22:00                0:00:00         , new
31                      14:02:00                14:03:00 (Server 1)     14:18:00                0:01:00         , regular
```

Element A1 program

```
# Get 1000 random numbers and their frequencies
from scipy.stats import chisquare
import random
import math

def populate_dictionary(num_range):
    num_count = {}
    i = 0
    while i < num_range:
        num_count[str(i)] = 0
        i += 1
    return num_count

def chi_sqaure_test(num_count):
    num_frequencies = list(num_count.values())
    chi_sqaure_stat = chisquare(num_frequencies)
    return chi_sqaure_stat

def language_random(num_integers, num_range, num_count):
    num_count = populate_dictionary(num_range)

    for i in range(num_integers):
        rand_num = random.randint(0, num_range - 1)
        if str(rand_num) in num_count:
            num_count[str(rand_num)] += 1
        else:
            num_count[str(rand_num)] = 1
    return num_count

def mult_congru(num_count, num_integers, seed, a, b, m):
    while num_integers >  0:
        rand_num = ((a * seed) + b) % m
```

```
			if str(rand_num) in num_count:
				num_count[str(rand_num)] += 1
			else:
				num_count[str(rand_num)] = 1
			seed = rand_num
			num_integers -= 1
		return num_count



#Returns percentage of chi-squared stats that are in good range
def results_check(list_chisq_stats, num_range):
		in_good_range = 2*(math.sqrt(num_range))#2SqrootR
		count = 0
		for chisq in  list_chisq_stats:
			if not ((num_range - in_good_range) <= chisq and chisq <=  (num_range + in_good_range)):
				count += 1

		amount_in_range = len(list_chisq_stats) - count
		percentage_good_range = (100 * amount_in_range)/ len(list_chisq_stats)
		print("percentage in good range", percentage_good_range)

		good_range = "Acceptable number range: " + str(num_range - in_good_range) + " - " + str(num_range + in_good_range)
		return good_range


def main():
		print("Which random number generator would you like to use?")
		generator_decision = input("Language(l) or Muliplicative Congruential Method(m): ")

		repeitions = int(input("How many repeitions: "))
		list_chisq_stats = []

		num_count = {}#Dict of random numbers and their occurences
		num_integers = int(input("Enter number of integers: "))
```

```
        if generator_decision == 'l':
                num_range = int(input("Enter Upper Range, (lower range = 0): "))
                num_count = language_random(num_integers, num_range, num_count)
        elif generator_decision == 'm':
                seed = int(input("Seed(u0): "))#u0
                a = int(input("a: "))
                b = int(input("b: "))
                m = int(input("m: "))
                num_count = mult_congru(num_count, num_integers, seed, a, b, m)

        while repeitions > 0:
                num_count.clear()
                if generator_decision == 'l':
                        num_count = language_random(num_integers, num_range, num_count)
                elif generator_decision == 'm':
                        num_count = mult_congru(num_count, num_integers, seed, a, b, m)

                print("Frequencies of numbers: ", num_count)
                chisq_stat = chi_sqaure_test(num_count)
                print("Chi-Sqaure stat: ", chisq_stat[0])
                list_chisq_stats.append(chisq_stat[0])

                repeitions -= 1
                print()

        if generator_decision == 'l':
                print(list_chisq_stats)
                print(results_check(list_chisq_stats, num_range))
        elif generator_decision == 'm':
                print("List of chi squared stats: ", list_chisq_stats)
                print(results_check(list_chisq_stats, m-1))
if __name__ == '__main__':
        main()
```

Element A2 Program

```
#Simulation of hospital queue
from datetime import time
from datetime import timedelta
import datetime
import random
import math

#Server Class
class Server:
	def __init__(self, number, rate):
		self.number = number
		self.rate = rate
		self.service_time = int(60/rate)
		self.is_free = True
		self.next_free = 0
		self.time_idle = 0

		self.server_gender = "unknown"
		self.server_break_times = []
		self.is_server_on_break = False
		self.experience = "normal"

	def get_server_number(self):
		return self.number

	def get_is_free(self):
		return self.is_free

	def get_service_time(self):
		return self.service_time
```

```
def get_next_free(self):
    return self.next_free

def get_time_idle(self):
    return self.time_idle

def get_gender(self):
    return self.server_gender

def get_server_break_times(self):
    return self.server_break_times

def get_server_on_break(self):
    return self.is_server_on_break

def get_server_experience(self):
    return self.experience

def set_next_free(self, next_free):
    self.next_free = next_free

def set_is_free(self, free_status):
    self.is_free = free_status

def set_time_idle(self, time_idle):
    self.time_idle = time_idle

def set_gender(self, server_gender):
    self.server_gender = server_gender

def subtract_from_time_idle(self, time_to_subtract):
```

```
        time_to_subtract_seconds = timedelta(seconds = time_to_subtract * 60)
        self.time_idle = self.time_idle - time_to_subtract_seconds

    def set_server_break_times(self, server_break_times):
        self.server_break_times = server_break_times

    def set_server_on_break(self, break_status):
        self.is_server_on_break = break_status

    def set_experience(self, experience):
        self.experience = experience

#Customer Class
class Customer:
    def __init__(self, number, arrival_time):
        self.number = number
        self.arrival_time = arrival_time
        self.has_been_serviced = False

        self.time_admitted = 0
        self.time_in_queue = 0
        self.departure_time = 0
        self.which_sever = 0

        self.c_gender = "unknown"
        self.visiting_status = "regular"

    def __str__(self):
        return "Customer %s Arrival Time %s" % (self.number, self.arrival_time)

    def get_customer_number(self):
        return self.number
```

```
def get_arrival_time(self):
        return self.arrival_time

def get_customer_number(self):
        return self.number

def get_has_been_serviced(self):
        return self.has_been_serviced

def get_which_server(self):
        return self.which_sever

def get_departure_time(self):
        return self.departure_time

def get_time_admitted(self):
        return self.time_admitted

def get_time_in_queue(self):
        return self.time_in_queue

def get_customer_gender(self):
        return self.c_gender

def get_visiting_status(self):
        return self.visiting_status

def set_arrival_time(self, arrival_time):
        self.arrival_time = arrival_time

def set_has_been_serviced(self, service_status):
```

```
        self.has_been_serviced = service_status

    def set_departure_time(self, departure_time):
        self.departure_time = departure_time

    def set_which_server(self, which_sever):
        self.which_sever = which_sever

    def set_time_admitted(self, time_admitted):
        self.time_admitted = time_admitted

    def set_queue_time(self, time_in_queue):
        self.time_in_queue = time_in_queue

    def set_customer_gender(self, c_gender):
        self.c_gender = c_gender

    def set_visiting_status(self, visiting_status):
        self.visiting_status = visiting_status


#Parse time to create a time object
def parse_time(time_string):
    time_input = time_string.split(".")
    input_hour = int(time_input[0])
    input_minute = int(time_input[1])
    time_object = time(input_hour, input_minute)
    return time_object

#Add/take away minutes from time
def add_minutes_to_time(time_object, minutes):
    time_hour = time_object.hour
```

```
    time_minute = time_object.minute + minutes
    if time_minute >= 60:
        time_minute = time_minute - 60
        time_hour += 1
    elif time_minute < 0:
        time_minute = 60 + (time_minute)
        time_hour -= 1

    if time_hour == 24:
        time_hour = 0

    new_time = time(time_hour, time_minute)
    return new_time

#Generate a uniformally distributed list of times
#Starting with start times
def generate_arrival_times(start_time, last_time, arrival_rate):
    list_of_times = []
    arrival_interval = int(60/arrival_rate)

    list_of_times.append(start_time)
    next_time = start_time
    while next_time <= last_time:
        next_time = add_minutes_to_time(next_time, arrival_interval)
        if next_time <= last_time:
            list_of_times.append(next_time)

    return list_of_times

#Add standard deviation to times
def deviate_times(list_of_times, stand_dev):
    deviated_list_of_times = []
```

```
    for arr_time in list_of_times:
        minute_variation = random.randint(stand_dev * -1, stand_dev)
        deviated_time = add_minutes_to_time(arr_time, minute_variation)
        deviated_list_of_times.append(deviated_time)

    return deviated_list_of_times

#Later time should be second
def difference_between_times(first_time, second_time):
    first_delta = timedelta(hours = first_time.hour, minutes = first_time.minute, seconds = first_time.second)
    second_delta = timedelta(hours = second_time.hour, minutes = second_time.minute, seconds = second_time.second)
    time_diff = second_delta - first_delta
    return time_diff

def possoin_dristributed_times(customers, start_time, end_time):
    print("possoin_times")
    poisson_times = []
    num_of_customers = len(customers)
    p_count = 0
    while p_count < num_of_customers:
        rand_hour = random.randint(start_time.hour, end_time.hour)
        rand_minute = random.randint(0, 59)
        rand_time = time(rand_hour, rand_minute)

        if start_time <= rand_time <= end_time:
            poisson_times.append(rand_time)
            p_count += 1

    poisson_times.sort()
    i = 0
```

```
	for p_time in poisson_times:
		customers[i].set_arrival_time(p_time)
		i += 1

	return customers

def service_customers(servers, customers, start_time, last_time):
	current_time = customers[0].get_arrival_time()
	longest_server_time = 0
	for serv in servers:
		if serv.get_service_time() > longest_server_time:
			longest_server_time = serv.get_service_time()

	latest_time = add_minutes_to_time(last_time, longest_server_time)
	serviced_customers = []

	print()
	print("Customer arriving and being served.")

	all_customers_served = False

	while all_customers_served == False:
	#while current_time <= latest_time:
		for cust in customers:
			#Customer Arrived
			if cust.get_arrival_time() == current_time:
				print(current_time, "Customer", cust.get_customer_number(), 
"arrived")
			#Customer Departed
			elif cust.get_departure_time() == current_time:
				#Set server free
				for serv in servers:
```

```
                        if cust.get_which_server() ==
serv.get_server_number():
                            if serv.get_server_on_break() == False:
                                serv.set_is_free(True)

        #Is it a servers break time
        for serv in servers:
            if current_time in serv.get_server_break_times():
                if serv.get_is_free():
                    end_of_break = add_minutes_to_time(current_time,
30)
                    serv.set_next_free(end_of_break)
                    serv.set_server_on_break(True)
                else:
                    end_of_break =
add_minutes_to_time(serv.get_next_free(), 30)
                    serv.set_next_free(end_of_break)
                    serv.set_server_on_break(True)

            #Is a break over
            if current_time == serv.get_next_free() and
serv.get_server_on_break() == True:
                serv.set_is_free(True)

        #Is server free for unseen customers
        for serv in servers:
            if serv.get_is_free():
                #If a server is free, see a customer that has arrived
                for cust in customers:
                    if (cust.get_arrival_time() <= current_time and
                        cust.get_has_been_serviced() == False and
cust.get_customer_gender() == serv.get_gender()):

                        #Announce Action
```

```
#Set server info
serv.set_is_free(False)
server_next_free = add_minutes_to_time(current_time, serv.get_service_time())
if cust.get_visiting_status() == "new":
    server_next_free = add_minutes_to_time(current_time, serv.get_service_time()*2)
if serv.get_server_experience() == "experienced":
    server_next_free = add_minutes_to_time(current_time, int(serv.get_service_time()/2))

serv.set_next_free(server_next_free)

#Set customer info
cust.set_has_been_serviced(True)

customer_departure_time = add_minutes_to_time(current_time, serv.get_service_time())
if cust.get_visiting_status() == "new":
    customer_departure_time = add_minutes_to_time(current_time, serv.get_service_time()*2)
if serv.get_server_experience() == "experienced":
    customer_departure_time = add_minutes_to_time(current_time, int(serv.get_service_time()/2))

cust.set_departure_time(customer_departure_time)

cust.set_which_server(serv.get_server_number())
cust.set_time_admitted(current_time)
customer_queueing_time = difference_between_times(cust.get_arrival_time(), cust.get_time_admitted())
cust.set_queue_time(customer_queueing_time)
break
```

```
        cust_count = 0
        for cust in customers:
            if cust.get_has_been_serviced() == True:
                cust_count += 1

        if cust_count == len(customers):
            all_customers_served = True



        current_time = add_minutes_to_time(current_time, 1)

    return customers

def print_output(customers, gender_choice, new_regular_choice):

    print()
    print("Output from this run")
    print("Customer no. \t\t Actual Arrival time \t Admitted to server \t departed server \t
queue time")

    for cust in customers:
        server_string = "(" + str(cust.get_which_server()) + ")"

        additonal_string = "\t"
        if gender_choice == 'y':
            additonal_string += ", " + str(cust.get_customer_gender())
        if new_regular_choice == 'y':
            additonal_string += ", " + str(cust.get_visiting_status())

        print(cust.get_customer_number(), '\t\t\t', cust.get_arrival_time(), '\t\t',
cust.get_time_admitted(), server_string, '\t', cust.get_departure_time(), '\t\t',
cust.get_time_in_queue(), additonal_string)
```

```
def performence_metrics(servers, customers):

        print()
        print("Final Performence Metrics (from this cycle)")
        # Avg time a customer is in system
        total_time_in_system = 0
        total_time_in_queue = 0

        # Maximum time a customer spends in service and in queue
        max_time_in_system = 0
        max_time_in_queue = 0

        for cust in customers:

                time_in_system = difference_between_times(cust.get_arrival_time(),
cust.get_departure_time())
                #Convert time to minutes, timedeltas store time in seconds
                time_in_system = time_in_system.seconds / 60
                total_time_in_system += time_in_system

                time_in_queue = cust.get_time_in_queue().seconds / 60
                total_time_in_queue += time_in_queue

                if time_in_system > max_time_in_system:
                        max_time_in_system = time_in_system

                if time_in_queue > max_time_in_queue:
                        max_time_in_queue= time_in_queue


        avg_time_in_system = total_time_in_system/len(customers)
```

```
    avg_time_in_queue = total_time_in_queue/len(customers)

    #Time server spent idle
    first_time = customers[0].get_arrival_time()
    second_time = customers[len(customers)-1].get_departure_time()
    total_time_in_day = difference_between_times(first_time, second_time)

    time_all_servers_idle = timedelta()
    for serv in servers:
        serv.set_time_idle(total_time_in_day)
        for cust in customers:
            if cust.get_which_server() == serv.get_server_number():
                if serv.get_server_experience() == "experienced":

    serv.subtract_from_time_idle(int(serv.get_service_time()/2))
                else:
                    serv.subtract_from_time_idle(serv.get_service_time())

        #time_server_spent_idle = difference_between_times(total_time_servicing,
total_time_in_day)
        #serv.set_time_idle(time_server_spent_idle)
        print("Amount of time server ", serv.get_server_number(), "was idle:",
serv.get_time_idle())
        time_all_servers_idle += serv.get_time_idle()



    #Avg number in queue, system
    current_time = first_time
    minute_count = 0
    in_system_times = []
    in_queue_times = []
    currently_in_system = 0
    currently_in_queue = 0
```

```
while current_time <= second_time:
    for cust in customers:
        if current_time == cust.get_arrival_time():
            currently_in_system += 1
            currently_in_queue += 1

        if current_time == cust.get_time_admitted():
            currently_in_queue -= 1

        if current_time == cust.get_departure_time():
            currently_in_system -= 1

    in_system_times.append(currently_in_system)
    in_queue_times.append(currently_in_queue)

    current_time = add_minutes_to_time(current_time, 1)
    minute_count += 1

print(minute_count)
avg_num_in_system = sum(in_system_times)/minute_count
avg_num_in_queue = sum(in_queue_times)/minute_count



#Print Metrics
print("Avg. time of a customer in system (W):", avg_time_in_system)
print("Avg. time of a customer in queue (Wq):", avg_time_in_queue)
print("Max time a customer spends in system: ", max_time_in_system)
print("Max time a customer spends in queue: ", max_time_in_queue)
print("Total time all servers were idle: ", time_all_servers_idle)
print("Avg. in system at any given minute: ", avg_num_in_system)
print("Avg. in queue at any given minute: ", avg_num_in_queue)
```

```
	rep_performence_metrics = []
	rep_performence_metrics.append(avg_time_in_system)
	rep_performence_metrics.append(avg_time_in_queue)
	rep_performence_metrics.append(max_time_in_system)
	rep_performence_metrics.append(max_time_in_queue)
	rep_performence_metrics.append(time_all_servers_idle)
	rep_performence_metrics.append(avg_num_in_system)
	rep_performence_metrics.append(avg_num_in_queue)

	#time last customer leaves
	rep_performence_metrics.append(customers[len(customers)-
1].get_departure_time())

	return rep_performence_metrics

def calculate_standard_error(list_of_num):
	mean_of_list = sum(list_of_num)/float(len(list_of_num))

	deviations_from_mean = []
	for num in list_of_num:
		deviation = mean_of_list - num
		deviation = deviation * deviation#Square number to get rid of negatives
		deviations_from_mean.append(deviation)

	total_squared_deviations = sum(deviations_from_mean)
	total_squared_deviations = total_squared_deviations/(len(list_of_num)-1)
	sd = math.sqrt(total_squared_deviations)

	se = sd/(math.sqrt(len(list_of_num)))
	return se

def calculate_standard_error_timedelta(list_of_timedelta):
	td_list = []
```

```
    for t in list_of_timedelta:
        td_list.append(t.seconds)

    mean_of_list = sum(td_list)/float(len(td_list))

    deviations_from_mean = []
    for num in td_list:
        deviation = mean_of_list - num
        deviation = deviation * deviation#Square number to get rid of negatives
        deviations_from_mean.append(deviation)

    total_squared_deviations = sum(deviations_from_mean)
    total_squared_deviations = total_squared_deviations/(len(td_list)-1)
    sd = math.sqrt(total_squared_deviations)

    se = sd/(math.sqrt(len(td_list)))
    return se/60

def main():

    #Default values
    # num_servers = 2
    # server_rate = 4
    # arrival_rate = 6
    # stand_dev = 5
    # scheduling_choice = 'r'

    # time_string = "9.05"
    # last_time_string = "17.30"

    # replications = 100
```

```
#User Input
num_servers = int(input("Number of servers: "))
server_rate = int(input("Server rate: "))# Rate servers take per hour

arrival_rate = int(input("Arrival Rate: "))


#first customer enters service
time_string = input("Enter start time: ")
start_time = parse_time(time_string)

#last time a customer can enter service
last_time_string = input("Enter end time: ")
last_time = parse_time(last_time_string)

stand_dev = int(input("Standard Deviation in minutes: "))#Standard Deviation in minutes
scheduling_choice = input("Random uniform distribution (r) or Possoin Distribution (p): ")

gender_choice = input("Would you like specific servers for each gender (y/n): ")

break_choice = input("Would you like the servers to have breaks (y/n): ")

new_regular_choice = input("Would you like new and regualr customers(y/n): ")
if new_regular_choice == 'y':
    percent_new  = int(input("What percentage of the customers should be new: "))
    new_at_front = input("Would you like all new customers moved to the front (y/n): ")

experienced_server_choice = input("Is the lead server more experienced (y/n): ")

#Replicate from here:
```

```
replications = int(input("How many replications: "))

total_avg_time_in_system = []
total_avg_time_in_queue = []
total_max_time_in_system = []
total_max_time_in_queue = []
total_time_servers_idle = []
total_avg_num_in_system = []
total_avg_num_in_queue = []

times_last_customer_leaves = []

rep_count = 0
while rep_count < replications:
    #Generated list of arrival times
    list_of_times = generate_arrival_times(start_time, last_time, arrival_rate)
    list_of_times = deviate_times(list_of_times, stand_dev)
    list_of_times.sort()

    # Create a list of servers
    servers = []
    server_count = 1
    while server_count <= num_servers:
        server_name = "Server " + str(server_count)
        server = Server(server_name, server_rate)
        servers.append(server)
        server_count += 1

    #Create a list of customers
    customers = []
    customer_count = 1
    for arr_time in list_of_times:
```

```
            new_customer = Customer(customer_count, arr_time)
            customers.append(new_customer)
            customer_count += 1

        #Poisson Distributed times
        if scheduling_choice == 'p':
            customers = possoin_dristributed_times(customers, start_time,
last_time)

        #Give customers genders
        if gender_choice == 'y':
            genders = ["male", "female"]

            for cust in customers:
                cust.set_customer_gender(random.choice(genders))

            next_gender = 0
            for serv in servers:
                if next_gender == 0:
                    serv.set_gender(genders[0])
                    next_gender = 1
                elif  next_gender == 1:
                    serv.set_gender(genders[1])
                    next_gnder = 0

        #Give customers new or regular status
        if new_regular_choice == 'y':
            amount_to_change = int((len(customers)*percent_new)/100)
            if new_at_front == 'y':
                i = 0
                while i < amount_to_change:
                    customers[i].set_visiting_status("new")
                    i += 1
```

```
        else:
            print("change change_index", int(amount_to_change))
            i = 1
            while i <= len(customers):
                if i % int(amount_to_change) == 0:
                    customers[i-1].set_visiting_status("new")
                i += 1


    [print(cust.get_visiting_status()) for cust in customers]

    #Make lead server more experienced
    if experienced_server_choice == 'y':
        servers[0].set_experience("experienced")

    [print(serv.get_server_experience()) for serv in servers]

    #Give serverrs break times
    set_of_break_times =[[parse_time("10.45"), parse_time("14.45")],
[parse_time("11.15"), parse_time("15.15")]]
    next_break_set = 0
    if break_choice == 'y':
        for serv in servers:
            if next_break_set == 0:
                serv.set_server_break_times(set_of_break_times[0])
                next_break_set = 1
            elif next_break_set == 1:
                serv.set_server_break_times(set_of_break_times[1])
                next_break_set == 0

    print("Service customers throughout day")
    customers = service_customers(servers, customers, start_time, last_time)
```

```
            #output
            print_output(customers, gender_choice, new_regular_choice)

            rep_performence_metrics = performence_metrics(servers, customers)

            #Collect results from this cyclle
            total_avg_time_in_system.append(rep_performence_metrics[0])
            total_avg_time_in_queue.append(rep_performence_metrics[1])
            total_max_time_in_system.append(rep_performence_metrics[2])
            total_max_time_in_queue.append(rep_performence_metrics[3])
            total_time_servers_idle.append(rep_performence_metrics[4])
            total_avg_num_in_system.append(rep_performence_metrics[5])
            total_avg_num_in_queue.append(rep_performence_metrics[6])

            times_last_customer_leaves.append(rep_performence_metrics[7])

            rep_count += 1

        # Final Replication Results
        print()
        print("After all replications")
        print("Total average time in system: ",
round(sum(total_avg_time_in_system)/replications, 4), "minutes")
        print("\t\t standarad error: ",
round(calculate_standard_error(total_avg_time_in_system), 4))

        print("Total average time in queue: ",
round(sum(total_avg_time_in_queue)/replications, 4), "minutes")
        print("\t\t standarad error: ",
round(calculate_standard_error(total_avg_time_in_queue), 4))

        print("Average of maximum time spent in system: ",
round(sum(total_max_time_in_system)/replications, 4), "minutes")
```

```
        print("\t\t standarad error: ",
round(calculate_standard_error(total_max_time_in_system), 4))


        print("Average of maximum time spent in queue: ",
sum(total_max_time_in_queue)/replications, "minutes")

        print("\t\t standarad error: ",
round(calculate_standard_error(total_max_time_in_queue), 4))


        total_idle_time = 0

        for idle_time in total_time_servers_idle:

                total_idle_time += idle_time.seconds

        total_idle_time = total_idle_time/replications

        total_idle_time_delta = timedelta(seconds = int(total_idle_time))

        print("Average amount of time all servers were idle: ", total_idle_time_delta,
"minutes")

        print("\t\t standarad error: ",
round(calculate_standard_error_timedelta(total_time_servers_idle), 4))


        print("Average number in system at any given minute: ",
        round(sum(total_avg_num_in_system)/replications, 4))

        print("\t\t standarad error: ",
round(calculate_standard_error(total_avg_num_in_system), 4))


        print("Average nummber in queue at any given time: ",
        round(sum(total_avg_num_in_queue)/replications, 4))

        print("\t\t standarad error: ",
round(calculate_standard_error(total_avg_num_in_queue), 4))


        total_seconds = 0

        for last_time in times_last_customer_leaves:

                t_delta = timedelta(hours = last_time.hour, minutes = last_time.minute
,seconds = last_time.second)

                total_seconds += t_delta.seconds


        avg_last_time = timedelta(seconds =
total_seconds/len(times_last_customer_leaves))
```

```
    print("Avg. time last customer left: ", avg_last_time)

if __name__ == '__main__':
    main()
```