Project 2

Name: Richard Xie

ID: 915505564

Part 1

1.

Table compares the mean delay of simulated value with the theoretical value. ($\overline{D} = \frac{1/\mu}{1-\rho} = \frac{1/\mu}{1-\lambda/\mu}$)

λ(pkts/s)	0.2	0.4	0.6	0.8	0.9	0.99
Simulated Value (s)	1.251	1.658	2.529	4.996	9.558	86.938
Theoretical Value (s)	1.250	1.667	2.500	5.000	10.000	100.000
Percent Difference	0.080%	0.540%	1.160%	0.080%	4.420%	13.062%

2.

$p_n \lambda = p_{n+1} \mu$, n = 0, 1, 2, … , N – 1

$p_n = \rho p_{n-1} = \rho^n p_0$ for n = 0, 1, 2, … , N

$$\sum_{n=0}^{N} p_n = 1$$

$$p_0 = 1 - \sum_{n=1}^{N} p^n p_0 = \frac{1-\rho}{1-\rho^{N+1}}$$

$$p_n = \frac{(1-\rho)\rho^n}{1-\rho^{N+1}}$$ for n = 0, 1, 2, … , N

$$p_d = \frac{(1-\rho)\rho^B}{1-\rho^{B+1}}, \quad \rho = \frac{\lambda}{\mu}$$

3. code and output

```
# Richard Xie 915505564

# This is a simpy based   simulation of a M/M/1 queue system

import random
import simpy
import math

RANDOM_SEED = 29
SIM_TIME = 1000000
MU = 1
B = 10   # modify buffer size for different output


""" Queue system   """
class server_queue:
    def __init__(self, env, arrival_rate, Packet_Delay, Server_Idle_Periods):
        self.server = simpy.Resource(env, capacity = 1)
        self.env = env
        self.queue_len = 0
        self.flag_processing = 0
        self.packet_number = 0
        self.sum_time_length = 0
        self.start_idle_time = 0
        self.arrival_rate = arrival_rate
        self.Packet_Delay = Packet_Delay
        self.Server_Idle_Periods = Server_Idle_Periods

    def process_packet(self, env, packet):
        with self.server.request() as req:
            start = env.now
```

```
                yield req
                yield env.timeout(random.expovariate(MU))
                latency = env.now - packet.arrival_time
                self.Packet_Delay.addNumber(latency)
                #print("Packet number {0} with arrival time {1} latency {2}".format(packet.identifier, packet.arrival_time, latency))
                self.queue_len -= 1
                if self.queue_len == 0:
                    self.flag_processing = 0
                    self.start_idle_time = env.now

    def packets_arrival(self, env):
        # packet arrivals

        while True:
            # Infinite loop for generating packets
            yield env.timeout(random.expovariate(self.arrival_rate))
            # arrival time of one packet

            self.Packet_Delay.addC()
            self.packet_number += 1
            # packet id
            arrival_time = env.now
            #print(self.num_pkt_total, "packet arrival")
            new_packet = Packet(self.packet_number,arrival_time)
            if self.flag_processing == 0:
                self.flag_processing = 1
                idle_period = env.now - self.start_idle_time
                self.Server_Idle_Periods.addNumber(idle_period)
                #print("Idle period of length {0} ended".format(idle_period))
            if self.queue_len < B :
                self.queue_len += 1
            else :
                continue
            env.process(self.process_packet(env, new_packet))


""" Packet class """
class Packet:
    def __init__(self, identifier, arrival_time):
        self.identifier = identifier
        self.arrival_time = arrival_time
```

```
class StatObject:
    def __init__(self):
        self.dataset =[]
        self.total = 0

    def addNumber(self,x):
        self.dataset.append(x)

    def addC(self) :
        self.total += 1

    def totalC(self) :
        return self.total

    def sum(self):
        n = len(self.dataset)
        sum = 0
        for i in self.dataset:
            sum = sum + i
        return sum
    def mean(self):
        n = len(self.dataset)
        sum = 0
        for i in self.dataset:
            sum = sum + i
        return sum/n
    def maximum(self):
        return max(self.dataset)
    def minimum(self):
        return min(self.dataset)
    def count(self):
        return len(self.dataset)
    def median(self):
        self.dataset.sort()
        n = len(self.dataset)
        if n//2 != 0: # get the middle number
            return self.dataset[n//2]
        else: # find the average of the middle two numbers
            return ((self.dataset[n//2] + self.dataset[n//2 + 1])/2)
    def standarddeviation(self):
        temp = self.mean()
        sum = 0
        for i in self.dataset:
            sum = sum + (i - temp)**2
```

```
        sum = sum/(len(self.dataset) - 1)
        return math.sqrt(sum)


def main():
    print("Simple queue system model:mu = {0}".format(MU))
    print ("{0:<9} {1:<9} {2:<9} {3:<9} {4:<9} {5:<9} {6:<9} {7:<9} {8:<9}".format(
        "Lambda", "Count", "Min", "Max", "Mean", "Median", "Sd", "Utilization", "Pd"))
    random.seed(RANDOM_SEED)
    for arrival_rate in [0.2, 0.4, 0.6,  0.8, 0.9, 0.99]:
        env = simpy.Environment()
        Packet_Delay = StatObject()
        Server_Idle_Periods = StatObject()
        router = server_queue(env, arrival_rate, Packet_Delay, Server_Idle_Periods)
        env.process(router.packets_arrival(env))
        env.run(until=SIM_TIME)
        print ("{0:<9.3f} {1:<9} {2:<9.3f} {3:<9.3f} {4:<9.3f} {5:<9.3f} {6:<9.3f} {7:<9.3f} {8:<9.9f}".format(
            round(arrival_rate, 3),
            int(Packet_Delay.count()),
            round(Packet_Delay.minimum(), 3),
            round(Packet_Delay.maximum(), 3),
            round(Packet_Delay.mean(), 3),
            round(Packet_Delay.median(), 3),
            round(Packet_Delay.standarddeviation(), 3),
            round(1-Server_Idle_Periods.sum()/SIM_TIME, 3),
            (Packet_Delay.totalC() - Packet_Delay.count()) / float(Packet_Delay.totalC())))

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

```
In [1]: runfile('C:/Users/P-Ming/Desktop/ECS 152A/mm1-queue-infinte-queue-simulation.py', wdir='C:/Users/P-Ming/Desktop/ECS 152A')
Simple queue system model:mu = 1
Lambda    Count     Min       Max       Mean      Median    Sd        Utilization Pd
0.200     200377    0.000     15.023    1.251     0.867     1.254     0.200     0.000000000
0.400     401172    0.000     23.096    1.664     1.154     1.660     0.402     0.000057329
0.600     599482    0.000     24.461    2.455     1.730     2.375     0.601     0.002527450
0.800     781331    0.000     30.045    3.790     2.954     3.200     0.781     0.022823232
0.900     854259    0.000     32.460    4.640     3.931     3.528     0.854     0.050927731
0.990     905912    0.000     27.556    5.389     4.887     3.687     0.904     0.085379902

In [2]: runfile('C:/Users/P-Ming/Desktop/ECS 152A/mm1-queue-infinte-queue-simulation.py', wdir='C:/Users/P-Ming/Desktop/ECS 152A')
Simple queue system model:mu = 1
Lambda    Count     Min       Max       Mean      Median    Sd        Utilization Pd
0.200     200377    0.000     15.023    1.251     0.867     1.254     0.200     0.000000000
0.400     400070    0.000     18.180    1.658     1.146     1.660     0.399     0.000000000
0.600     601173    0.000     30.204    2.529     1.749     2.539     0.603     0.000004990
0.800     799712    0.000     54.270    4.995     3.452     4.984     0.800     0.000003751
0.900     898827    0.000     67.014    9.434     6.714     8.954     0.897     0.000331433
0.990     975865    0.000     79.286    23.329    21.843    15.195    0.974     0.015288351
```

4.

Table compares the loss probability Pd of simulated value with the theoretical value.

B = 10

λ(pkts/s)	Simulated Value	Theoretical Value	Percent Difference
0.2	0.000000000	8.192 x 10^-8	0
0.4	0.000057329	0.000062317	8.882%
0.6	0.002527450	0.002427454	4.119%
0.8	0.022823232	0.023492858	2.850%
0.9	0.050927731	0.050813731	0.224%
0.99	0.085379902	0.086409993	1.192%

B = 50

λ(pkts/s)	Simulated Value	Theoretical Value	Percent Difference
0.2	0.000000000	9.0072 x 10^-36	0
0.4	0.000000000	7.6059 x 10^-21	0
0.6	0.000004990	3.2331 x 10^-12	1.5434 x 10^8 %
0.8	0.000003751	0.000002855	31.405%
0.9	0.000331433	0.000517779	35.990%
0.99	0.015288351	0.015085778	1.343%

Part 2

1.1 See the other pdf file.

1.2

Table of throughput with binary exponential backoff algorithm.

λ(pkts/s)	Throughput
0.01	0.100074
0.02	0.200657
0.03	0.300477
0.04	0.399229
0.05	0.499506
0.06	0.598079
0.07	0.698049
0.08	0.798214
0.09	0.893855

Code output

```
In [1]: runfile('C:/Users/P-Ming/Desktop/ECS 152A/backoff-algorithm-analysis.py', wdir='C:/Users/P-Ming/Desktop/ECS 152A')
Arrival rate Transmitted pkts  Throughput Collision
0.01         100074            0.100074   19328
0.02         200657            0.200657   104020
0.03         300477            0.300477   311402
0.04         399229            0.399229   436152
0.05         499506            0.499506   374682
0.06         598079            0.598079   282785
0.07         698049            0.698049   196134
0.08         798214            0.798214   124884
0.09         893855            0.893855   73371
```

It’s almost like linear grow, which satisfies what we discussed in class.

Table of throughput with linear backoff algorithm.

λ(pkts/s)	Throughput
0.01	0.099801
0.02	0.19991
0.03	0.290137
0.04	0.290115
0.05	0.288485
0.06	0.290463
0.07	0.290635
0.08	0.290796
0.09	0.289979

Code output

```
In [1]: runfile('C:/Users/P-Ming/Desktop/ECS 152A/backoff-algorithm-analysis.py', wdir='C:/Users/P-Ming/Desktop/ECS 152A')
Arrival rate Transmitted pkts  Throughput Collision
0.01         99801             0.099801   32222
0.02         199910            0.19991    198364
0.03         290137            0.290137   1657152
0.04         290115            0.290115   1664041
0.05         288485            0.288485   1673546
0.06         290463            0.290463   1664077
0.07         290635            0.290635   1662383
0.08         290796            0.290796   1662505
0.09         289979            0.289979   1666322
```

It grows for a while, then stops growing, which satisfies what we discussed in class.