MGTA613 Assignment 4 - Owen Petter

April 13, 2024

[ ]:
```
import simpy
import random
```

Question 1

[ ]:
```
system_times = []
```

[ ]:
```
#By initaiting car id vairable I can then track cars that are not done the␣
 ↪system when the 12 hour day is over and every car clears the system
car_id = 0
#By storing car arrivals I can track the end time of a car even if it is not␣
 ↪done the system, whihc helps me more accuratly track loss of goodwhile for␣
 ↪cars in the system that did not complete the system before the car wash␣
 ↪closes after 12 horus
car_arrivals = {}
```

[ ]:
```
class Carwash(object):
    def __init__(self, env, num_bays):
        self.env = env
        self.bays = simpy.Resource(env, num_bays)

    def getWash(self, car_id):
        yield self.env.timeout(random.randint(4, 8))
        #This deletes the arrival time for cars that finish the system as its␣
 ↪not needed for them, only needed for cars that don't finish the system
        del car_arrivals[car_id]
```

[ ]:
```
def goToCarwash(env, car_id, carwash):
    global car_arrivals
    arrival_time = env.now
    #This records car arrivals into the system
    car_arrivals[car_id] = arrival_time

    with carwash.bays.request() as request:
        yield request
        yield env.process(carwash.getWash(car_id))

    #This calcs the total system time of cars that finish the system
```

```
    system_time = env.now - arrival_time
    system_times.append(system_time)
```

[ ]:
```
def run_Carwash(env, num_bays):
    carwash = Carwash(env, num_bays)
    global car_id, car_arrivals

    while True:
        #cars arrive at 80 customers per hour, which is turned into minutes
        yield env.timeout(random.expovariate(1 / (60 / 80)))
        car_id += 1
        env.process(goToCarwash(env, car_id, carwash))
```

[ ]:
```
def finalize_times(env):
    #This calcs the system times for cars that didnt finish the car wash␣
 ↪because it closes before they can complete the full system (at the end of␣
 ↪the 12 hours), it then adds these unfished time to system_times
    end_time = env.now
    for cid, arrival in car_arrivals.items():
        unfinished_system_time = end_time - arrival
        system_times.append(unfinished_system_time)
    car_arrivals.clear()
```

To ensure the most accurate simulation, I wanted to make sure the simluation stopped after 12 hours to mimic the real 12 hour open period each day, thats why env.run(until) is set to 720 minutes. This is run 1000 times which simulates just over 2.7 years of operations where the car wash closes and empties after each 12 hour day, this long simulation period helps make sure that when I analyze the results, enough events have occured to smooth out outlier results and ensure trust worthy analysis. 1000 operational day simluations are done for each set up of bays (1-15).

[ ]:
```
def run_simulations(range_of_bays, num_runs=1000):
    random.seed(12565)
    #stores the results for each bay in dictionary so each one can be analyzed␣
 ↪seperatly
    results = {}
    #running sim 1000 times for each bay set up through the whole range of bays
    for num_bays in range(1, range_of_bays + 1):
        bay_results = []
        #list where each bays results are stored
        for run in range(num_runs):
            global system_times
            system_times = []
            #for each run seperate system times are stored, runs for 1000␣
 ↪different and seperate operational days of 12 hours
            env = simpy.Environment()
            env.process(run_Carwash(env, num_bays))
            env.run(until=720)
```

```
            #this helps handle unfinsihed cars
            finalize_times(env)

            #Stores system times for this run separately then loops to next
            bay_results.append(system_times.copy())

        #This stores the detailed run results for the current number of bays,␣
↪then once this bay is done 1000 times, moves onto the next
        results[num_bays] = bay_results

    return results
```

[ ]:
```
#Running sim with for bays 1-15
sim_results = run_simulations(15)
```

Here I am just testing the logic of my set up, the goal was to measure the system times of cars even if they have not finished the car wash and were in line while it closed. This means if the arrival rate is set up properly for each car in each simulation, then no matter the amount bays, the number of cars that were in the simulation should be the same even if they did not finish. If cars that do not finish the simulation are not tracked and thier system times not recorded, then you would see decline in the number of cars recorded in the system if the bay number is low enough that the rate in which cars arrive is faster then the rate in which the bays can process them, this results in cars that enter the system to not complete it before the 12 hour day is done and if those are not recorded, then systems with lower bay amounts will see understated cars in their system, and thus very understated loss of goodwill costs.

[ ]:
```
for num_bays, all_runs in sim_results.items():
    average_length_per_run = sum(len(run) for run in all_runs) / len(all_runs)␣
↪if all_runs else 0
    print(f"Average number of cars processed for {num_bays} bays across all␣
↪runs: {average_length_per_run:.2f}")
```

```
Average number of cars processed for 1 bays across all runs: 961.42
Average number of cars processed for 2 bays across all runs: 959.21
Average number of cars processed for 3 bays across all runs: 959.22
Average number of cars processed for 4 bays across all runs: 959.37
Average number of cars processed for 5 bays across all runs: 959.38
Average number of cars processed for 6 bays across all runs: 961.78
Average number of cars processed for 7 bays across all runs: 959.36
Average number of cars processed for 8 bays across all runs: 959.98
Average number of cars processed for 9 bays across all runs: 961.91
Average number of cars processed for 10 bays across all runs: 959.50
Average number of cars processed for 11 bays across all runs: 960.30
Average number of cars processed for 12 bays across all runs: 959.56
Average number of cars processed for 13 bays across all runs: 959.44
Average number of cars processed for 14 bays across all runs: 959.55
Average number of cars processed for 15 bays across all runs: 960.55
```

[ ]:
```
def calculate_average_system_times(sim_results):
    average_system_times = {}
    for num_bays, all_runs in sim_results.items():

        combined_system_times = []

        for run in all_runs:
            combined_system_times.extend(run)

        if combined_system_times:
            average_system_times[num_bays] = sum(combined_system_times) /␣
 ↪len(combined_system_times)
        else:
            average_system_times[num_bays] = 0

    return average_system_times
```

[ ]:
```
average_system_times = calculate_average_system_times(sim_results)

for num_bays, avg_time in average_system_times.items():
    print(f"Average System Time for {num_bays} bays: {avg_time:.2f} minutes")
```

```
Average System Time for 1 bays: 315.87 minutes
Average System Time for 2 bays: 271.16 minutes
Average System Time for 3 bays: 226.27 minutes
Average System Time for 4 bays: 182.27 minutes
Average System Time for 5 bays: 138.36 minutes
Average System Time for 6 bays: 94.78 minutes
Average System Time for 7 bays: 51.69 minutes
Average System Time for 8 bays: 17.23 minutes
Average System Time for 9 bays: 8.04 minutes
Average System Time for 10 bays: 6.65 minutes
Average System Time for 11 bays: 6.26 minutes
Average System Time for 12 bays: 6.10 minutes
Average System Time for 13 bays: 6.03 minutes
Average System Time for 14 bays: 6.00 minutes
Average System Time for 15 bays: 5.99 minutes
```

Question 2

[ ]:
```
def calculate_costs(sim_results):
    cost_per_bay = 10000
    overtime_rate = 2
    overtime_threshold = 10

    total_costs = {}

    for num_bays, all_runs in sim_results.items():
```

```
        run_fixed_costs = num_bays * cost_per_bay


        run_variable_costs = []
        run_total_costs = []

        for run in all_runs:
            #This calcs variable cost for this run
            variable_cost = sum(max(0, time - overtime_threshold) *␣
 ↪overtime_rate for time in run)
            run_variable_costs.append(variable_cost)

            total_cost = run_fixed_costs + variable_cost
            run_total_costs.append(total_cost)

        #This calcs avg cost over all 1000 sims for each bay set up
        avg_fixed_cost = run_fixed_costs  #Fixed cost is constant, no need to␣
 ↪avg
        avg_variable_cost = sum(run_variable_costs) / len(run_variable_costs)␣
 ↪if run_variable_costs else 0
        avg_total_cost = sum(run_total_costs) / len(run_total_costs) if␣
 ↪run_total_costs else 0

        total_costs[num_bays] = {
            "total_cost": avg_total_cost,
            "fixed_cost": avg_fixed_cost,
            "variable_cost": avg_variable_cost,
        }

    return total_costs

total_costs = calculate_costs(sim_results)

for num_bays, costs in total_costs.items():
    print(f"{num_bays} bays: AVG Total Cost = ${costs['total_cost']:.2f}, "
          f"AVG Fixed Cost = ${costs['fixed_cost']:.2f}, "
          f"AVG Variable (Loss of goodwill) Cost = ${costs['variable_cost']:.
 ↪2f}")
```

```
1 bays: AVG Total Cost = $598275.52, AVG Fixed Cost = $10000.00, AVG Variable
(Loss of goodwill) Cost = $588275.52
2 bays: AVG Total Cost = $521164.87, AVG Fixed Cost = $20000.00, AVG Variable
(Loss of goodwill) Cost = $501164.87
3 bays: AVG Total Cost = $445068.38, AVG Fixed Cost = $30000.00, AVG Variable
(Loss of goodwill) Cost = $415068.38
4 bays: AVG Total Cost = $370729.45, AVG Fixed Cost = $40000.00, AVG Variable
(Loss of goodwill) Cost = $330729.45
```

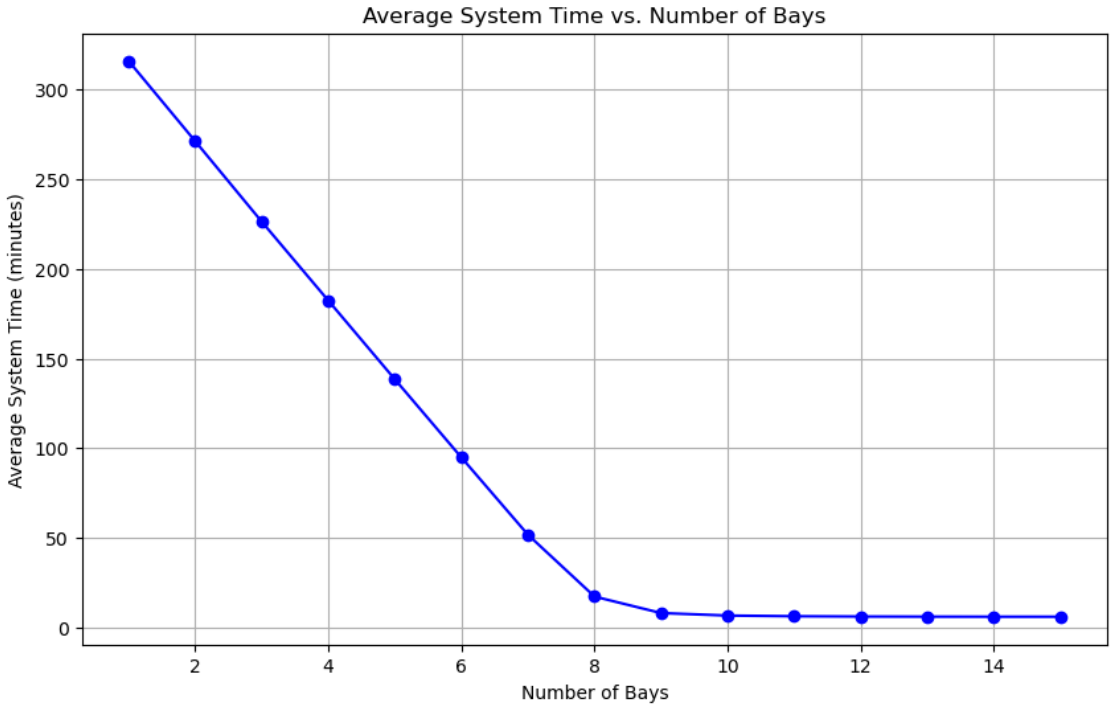
```
5 bays: AVG Total Cost = $296492.17, AVG Fixed Cost = $50000.00, AVG Variable
(Loss of goodwill) Cost = $246492.17
6 bays: AVG Total Cost = $223334.15, AVG Fixed Cost = $60000.00, AVG Variable
(Loss of goodwill) Cost = $163334.15
7 bays: AVG Total Cost = $150388.92, AVG Fixed Cost = $70000.00, AVG Variable
(Loss of goodwill) Cost = $80388.92
8 bays: AVG Total Cost = $95517.04, AVG Fixed Cost = $80000.00, AVG Variable
(Loss of goodwill) Cost = $15517.04
9 bays: AVG Total Cost = $91197.43, AVG Fixed Cost = $90000.00, AVG Variable
(Loss of goodwill) Cost = $1197.43
10 bays: AVG Total Cost = $100142.22, AVG Fixed Cost = $100000.00, AVG Variable
(Loss of goodwill) Cost = $142.22
11 bays: AVG Total Cost = $110025.49, AVG Fixed Cost = $110000.00, AVG Variable
(Loss of goodwill) Cost = $25.49
12 bays: AVG Total Cost = $120005.92, AVG Fixed Cost = $120000.00, AVG Variable
(Loss of goodwill) Cost = $5.92
13 bays: AVG Total Cost = $130001.34, AVG Fixed Cost = $130000.00, AVG Variable
(Loss of goodwill) Cost = $1.34
14 bays: AVG Total Cost = $140000.34, AVG Fixed Cost = $140000.00, AVG Variable
(Loss of goodwill) Cost = $0.34
15 bays: AVG Total Cost = $150000.07, AVG Fixed Cost = $150000.00, AVG Variable
(Loss of goodwill) Cost = $0.07
```

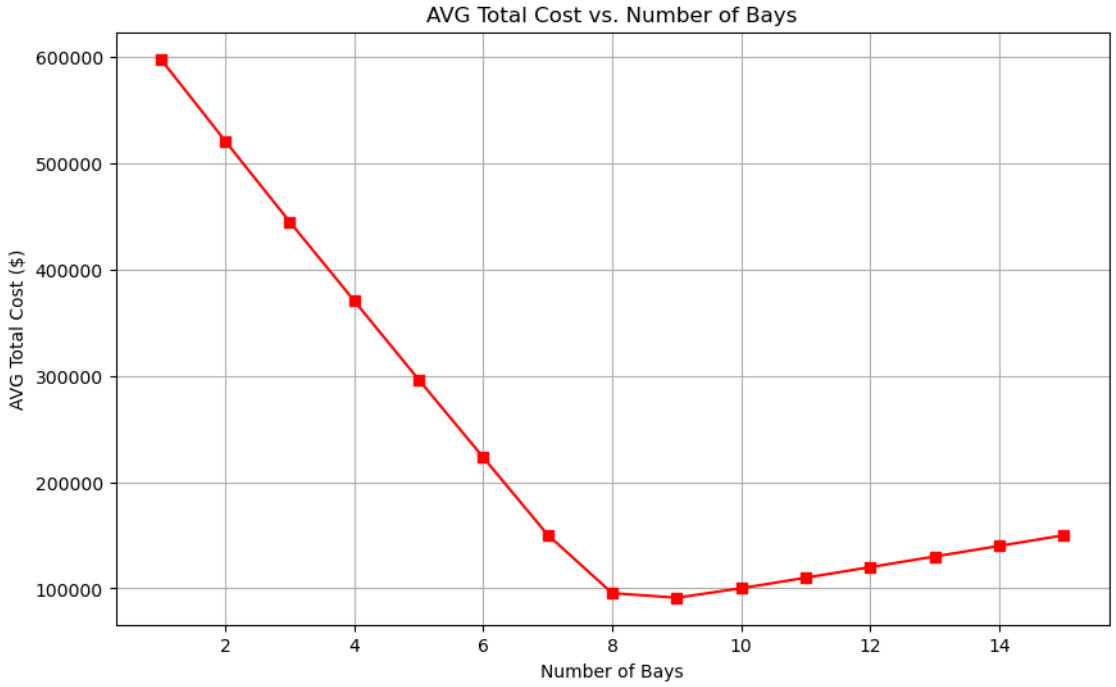
The optimal number of bays to open from a minimzing total cost perspective which takes into account both the fixed bay opening costs and loss of goodwill costs is 9 bays.

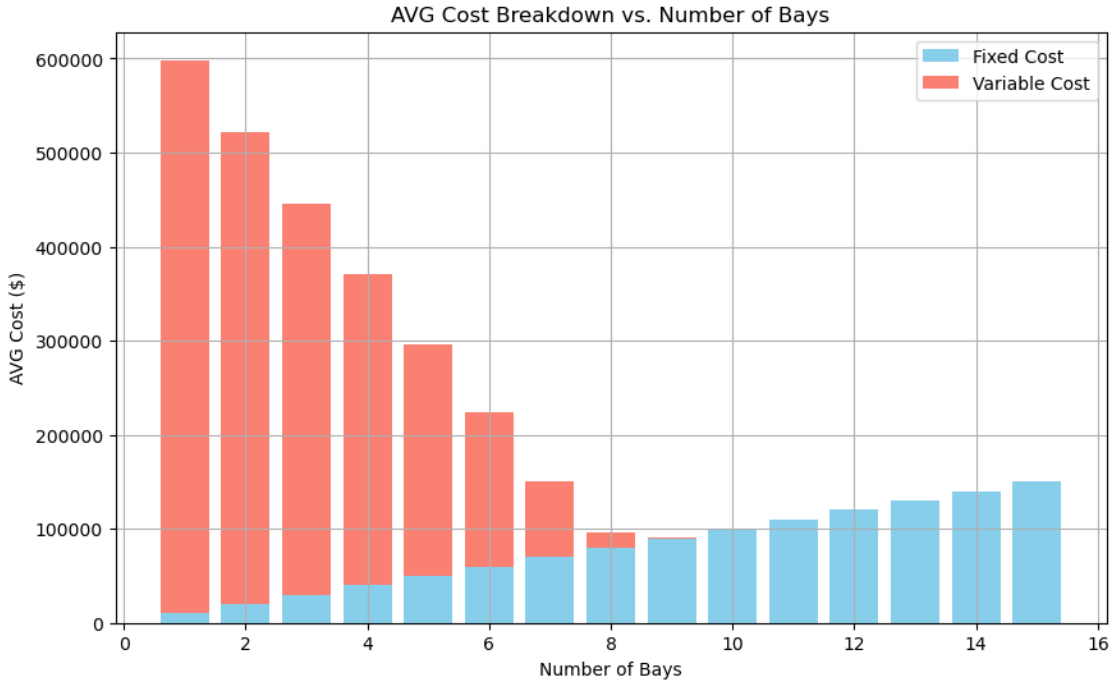
Question 3

[ ]:
```python
import matplotlib.pyplot as plt

bays = list(average_system_times.keys())
avg_times = [average_system_times[bay] for bay in bays]
total_costs_values = [total_costs[bay]['total_cost'] for bay in bays]
fixed_costs = [total_costs[bay]['fixed_cost'] for bay in bays]
variable_costs = [total_costs[bay]['variable_cost'] for bay in bays]

#Plot for Average System Time vs. Number of Bays
plt.figure(figsize=(10, 6))
plt.plot(bays, avg_times, marker='o', linestyle='-', color='b')
plt.title('Average System Time vs. Number of Bays')
plt.xlabel('Number of Bays')
plt.ylabel('Average System Time (minutes)')
plt.grid(True)
plt.show()

#Plot for Total Cost vs. Number of Bays
plt.figure(figsize=(10, 6))
plt.plot(bays, total_costs_values, marker='s', linestyle='-', color='r')
plt.title('AVG Total Cost vs. Number of Bays')
```

```
plt.xlabel('Number of Bays')
plt.ylabel('AVG Total Cost ($)')
plt.grid(True)
plt.show()

#Plot for Breakdown of Total Cost vs. Number of Bays
plt.figure(figsize=(10, 6))
plt.bar(bays, fixed_costs, color='skyblue', label='Fixed Cost')
plt.bar(bays, variable_costs, bottom=fixed_costs, color='salmon',␣
 ↪label='Variable Cost')
plt.title('AVG Cost Breakdown vs. Number of Bays')
plt.xlabel('Number of Bays')
plt.ylabel('AVG Cost ($)')
plt.legend()
plt.grid(True)
plt.show()
```




AVG Total Cost vs. Number of Bays
AVG Total Cost ($)
600000
500000
400000
300000
200000
100000
2
4
6
8
10
12
14
Number of Bays


AVG Cost Breakdown vs. Number of Bays
Fixed Cost
Variable Cost
AVG Cost ($)
600000
500000
400000
300000
200000
100000
0
0
2
4
6
8
10
12
14
16
Number of Bays

Question 4

[ ]:
```python
#remaking function for to incorpate busy period in the middle of the day

def run_Carwash(env, num_bays):
    carwash = Carwash(env, num_bays)
    global car_id, car_arrivals

    while True:
        current_hour = (env.now / 60) % 24  # this just makes env.now time into␣
↪24 hour clock, making it easier to set busy time
        # defining busy hour as three hour period in the middle of operating␣
↪day, since env.now sets time to 0:00 when sim starts thats why 5 and 8 are␣
↪chosen, the sim ends at 12:00 each day
        if 5 <= current_hour < 8:
            arrival_rate = 200 / 60 #busy arrival rate
        else:
            arrival_rate = 40 / 60  #non-busy arrival rate

        yield env.timeout(random.expovariate(arrival_rate))
        car_id += 1
        env.process(goToCarwash(env, car_id, carwash))
```

[ ]:
```python
sim_results_q4 = run_simulations(20)
```

[ ]:
```python
#same as q1, just testing that sim is working as anticipated
for num_bays, all_runs in sim_results_q4.items():
    average_length_per_run_q4 = sum(len(run) for run in all_runs) /␣
↪len(all_runs) if all_runs else 0
    print(f"Average number of cars processed for {num_bays} bays across all␣
↪runs: {average_length_per_run_q4:.2f}")
```

```
Average number of cars processed for 1 bays across all runs: 958.17
Average number of cars processed for 2 bays across all runs: 955.91
Average number of cars processed for 3 bays across all runs: 956.28
Average number of cars processed for 4 bays across all runs: 956.15
Average number of cars processed for 5 bays across all runs: 956.85
Average number of cars processed for 6 bays across all runs: 957.12
Average number of cars processed for 7 bays across all runs: 957.20
Average number of cars processed for 8 bays across all runs: 956.98
Average number of cars processed for 9 bays across all runs: 957.97
Average number of cars processed for 10 bays across all runs: 957.29
Average number of cars processed for 11 bays across all runs: 955.53
Average number of cars processed for 12 bays across all runs: 956.10
Average number of cars processed for 13 bays across all runs: 957.60
Average number of cars processed for 14 bays across all runs: 954.46
Average number of cars processed for 15 bays across all runs: 957.46
Average number of cars processed for 16 bays across all runs: 957.34
Average number of cars processed for 17 bays across all runs: 956.82
Average number of cars processed for 18 bays across all runs: 957.71
```

```
Average number of cars processed for 19 bays across all runs: 956.53
Average number of cars processed for 20 bays across all runs: 956.61
```

[ ]:
```python
average_system_times_q4 = calculate_average_system_times(sim_results_q4)

for num_bays, avg_time in average_system_times_q4.items():
    print(f"Average System Time for {num_bays} bays: {avg_time:.2f} minutes")
```

```
Average System Time for 1 bays: 300.25 minutes
Average System Time for 2 bays: 255.34 minutes
Average System Time for 3 bays: 211.72 minutes
Average System Time for 4 bays: 173.52 minutes
Average System Time for 5 bays: 152.77 minutes
Average System Time for 6 bays: 137.45 minutes
Average System Time for 7 bays: 122.12 minutes
Average System Time for 8 bays: 107.33 minutes
Average System Time for 9 bays: 92.37 minutes
Average System Time for 10 bays: 77.53 minutes
Average System Time for 11 bays: 62.49 minutes
Average System Time for 12 bays: 50.09 minutes
Average System Time for 13 bays: 40.56 minutes
Average System Time for 14 bays: 32.21 minutes
Average System Time for 15 bays: 26.07 minutes
Average System Time for 16 bays: 20.88 minutes
Average System Time for 17 bays: 16.38 minutes
Average System Time for 18 bays: 12.75 minutes
Average System Time for 19 bays: 10.04 minutes
Average System Time for 20 bays: 8.10 minutes
```

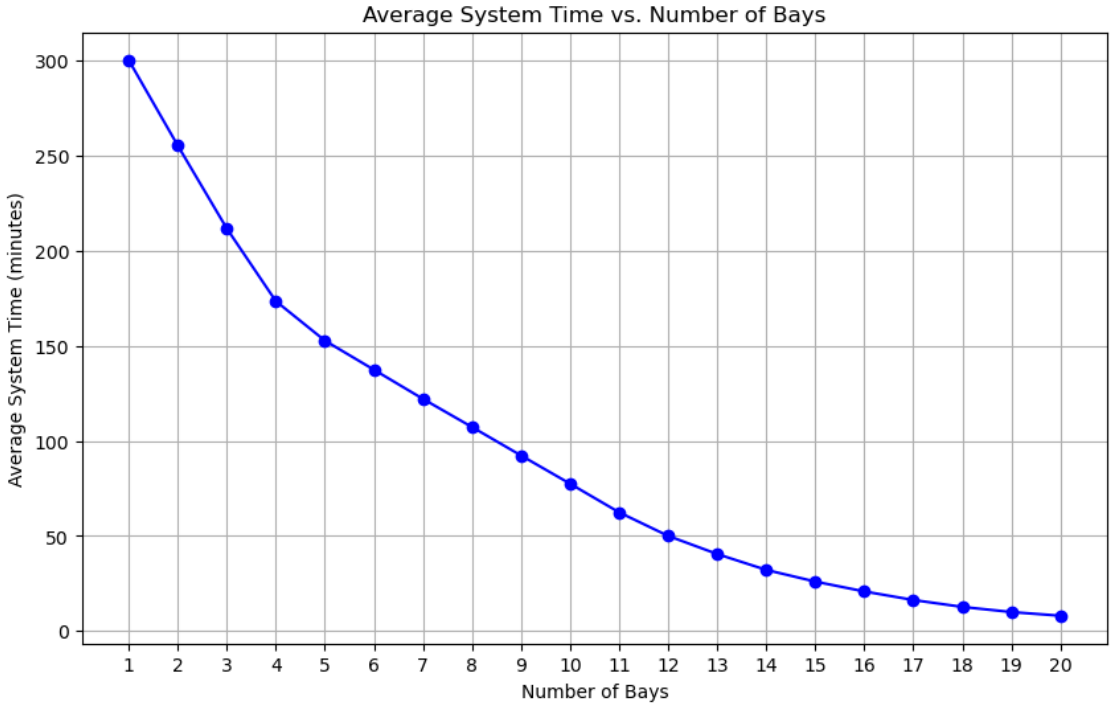
[ ]:
```python
total_costs_q4 = calculate_costs(sim_results_q4)

for num_bays, costs in total_costs_q4.items():
    print(f"{num_bays} bays: AVG Total Cost = ${costs['total_cost']:.2f}, "
          f"AVG Fixed Cost = ${costs['fixed_cost']:.2f}, "
          f"AVG Variable (Loss of goodwill) Cost = ${costs['variable_cost']:.
 ↪2f}")
```

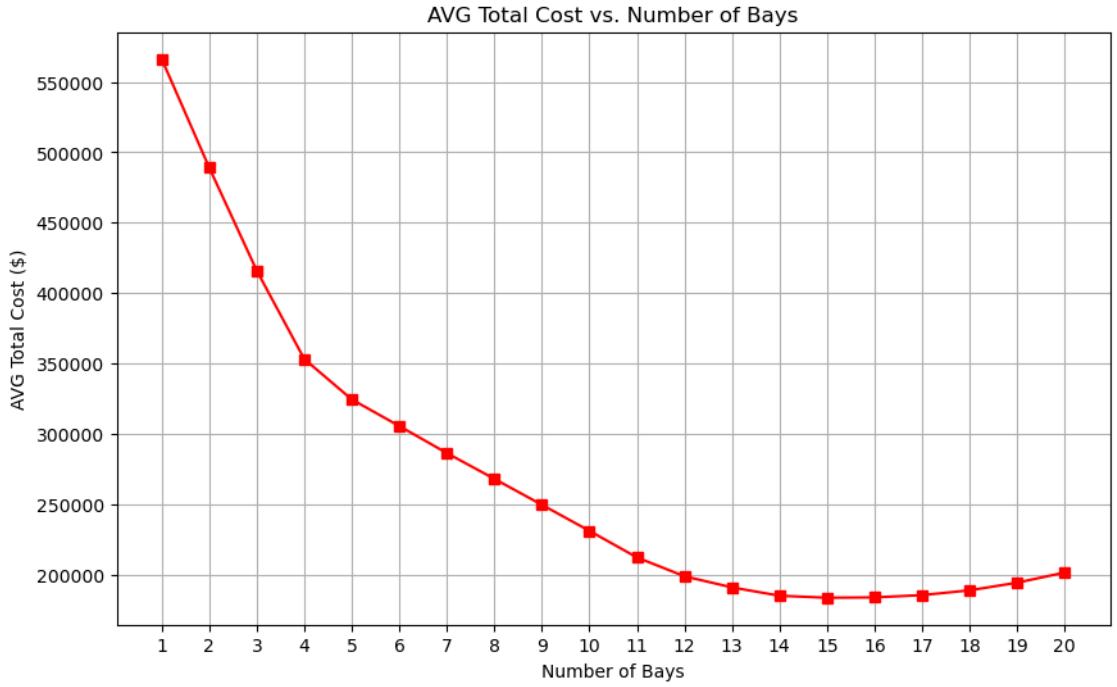
```
1 bays: AVG Total Cost = $566300.98, AVG Fixed Cost = $10000.00, AVG Variable
(Loss of goodwill) Cost = $556300.98
2 bays: AVG Total Cost = $489130.55, AVG Fixed Cost = $20000.00, AVG Variable
(Loss of goodwill) Cost = $469130.55
3 bays: AVG Total Cost = $415925.23, AVG Fixed Cost = $30000.00, AVG Variable
(Loss of goodwill) Cost = $385925.23
4 bays: AVG Total Cost = $353153.99, AVG Fixed Cost = $40000.00, AVG Variable
(Loss of goodwill) Cost = $313153.99
5 bays: AVG Total Cost = $324420.55, AVG Fixed Cost = $50000.00, AVG Variable
(Loss of goodwill) Cost = $274420.55
6 bays: AVG Total Cost = $305519.03, AVG Fixed Cost = $60000.00, AVG Variable
(Loss of goodwill) Cost = $245519.03
```

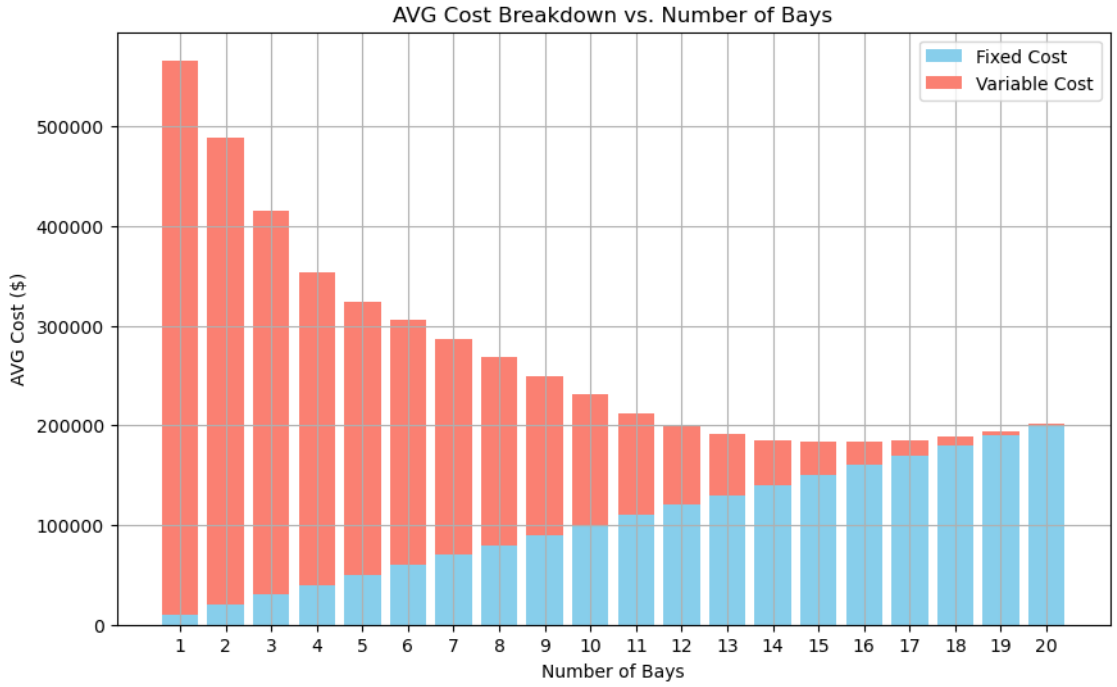
```
7 bays: AVG Total Cost = $286309.63, AVG Fixed Cost = $70000.00, AVG Variable
(Loss of goodwill) Cost = $216309.63
8 bays: AVG Total Cost = $268004.04, AVG Fixed Cost = $80000.00, AVG Variable
(Loss of goodwill) Cost = $188004.04
9 bays: AVG Total Cost = $249562.08, AVG Fixed Cost = $90000.00, AVG Variable
(Loss of goodwill) Cost = $159562.08
10 bays: AVG Total Cost = $231063.50, AVG Fixed Cost = $100000.00, AVG Variable
(Loss of goodwill) Cost = $131063.50
11 bays: AVG Total Cost = $212170.70, AVG Fixed Cost = $110000.00, AVG Variable
(Loss of goodwill) Cost = $102170.70
12 bays: AVG Total Cost = $198784.53, AVG Fixed Cost = $120000.00, AVG Variable
(Loss of goodwill) Cost = $78784.53
13 bays: AVG Total Cost = $190883.14, AVG Fixed Cost = $130000.00, AVG Variable
(Loss of goodwill) Cost = $60883.14
14 bays: AVG Total Cost = $184971.92, AVG Fixed Cost = $140000.00, AVG Variable
(Loss of goodwill) Cost = $44971.92
15 bays: AVG Total Cost = $183545.82, AVG Fixed Cost = $150000.00, AVG Variable
(Loss of goodwill) Cost = $33545.82
16 bays: AVG Total Cost = $183806.21, AVG Fixed Cost = $160000.00, AVG Variable
(Loss of goodwill) Cost = $23806.21
17 bays: AVG Total Cost = $185404.35, AVG Fixed Cost = $170000.00, AVG Variable
(Loss of goodwill) Cost = $15404.35
18 bays: AVG Total Cost = $188827.39, AVG Fixed Cost = $180000.00, AVG Variable
(Loss of goodwill) Cost = $8827.39
19 bays: AVG Total Cost = $194240.24, AVG Fixed Cost = $190000.00, AVG Variable
(Loss of goodwill) Cost = $4240.24
20 bays: AVG Total Cost = $201458.64, AVG Fixed Cost = $200000.00, AVG Variable
(Loss of goodwill) Cost = $1458.64
```

[ ]:
```python
bays_q4 = list(average_system_times_q4.keys())
avg_times_q4 = [average_system_times_q4[bay] for bay in bays_q4]
total_costs_values_q4 = [total_costs_q4[bay]['total_cost'] for bay in bays_q4]
fixed_costs_q4 = [total_costs_q4[bay]['fixed_cost'] for bay in bays_q4]
variable_costs_q4 = [total_costs_q4[bay]['variable_cost'] for bay in bays_q4]

#Plot for Average System Time vs. Number of Bays
plt.figure(figsize=(10, 6))
plt.plot(bays_q4, avg_times_q4, marker='o', linestyle='-', color='b')
plt.title('Average System Time vs. Number of Bays')
plt.xlabel('Number of Bays')
plt.ylabel('Average System Time (minutes)')
plt.xticks(bays_q4)
plt.grid(True)
plt.show()

#Plot for Total Cost vs. Number of Bays
plt.figure(figsize=(10, 6))
```

```
plt.plot(bays_q4, total_costs_values_q4, marker='s', linestyle='-', color='r')
plt.title('AVG Total Cost vs. Number of Bays')
plt.xlabel('Number of Bays')
plt.ylabel('AVG Total Cost ($)')
plt.xticks(bays_q4)
plt.grid(True)
plt.show()

#Plot for Breakdown of Total Cost vs. Number of Bays
plt.figure(figsize=(10, 6))
plt.bar(bays_q4, fixed_costs_q4, color='skyblue', label='Fixed Cost')
plt.bar(bays_q4, variable_costs_q4, bottom=fixed_costs_q4, color='salmon',␣
 ↪label='Variable Cost')
plt.title('AVG Cost Breakdown vs. Number of Bays')
plt.xlabel('Number of Bays')
plt.ylabel('AVG Cost ($)')
plt.xticks(bays_q4)
plt.legend()
plt.grid(True)
plt.show()
```





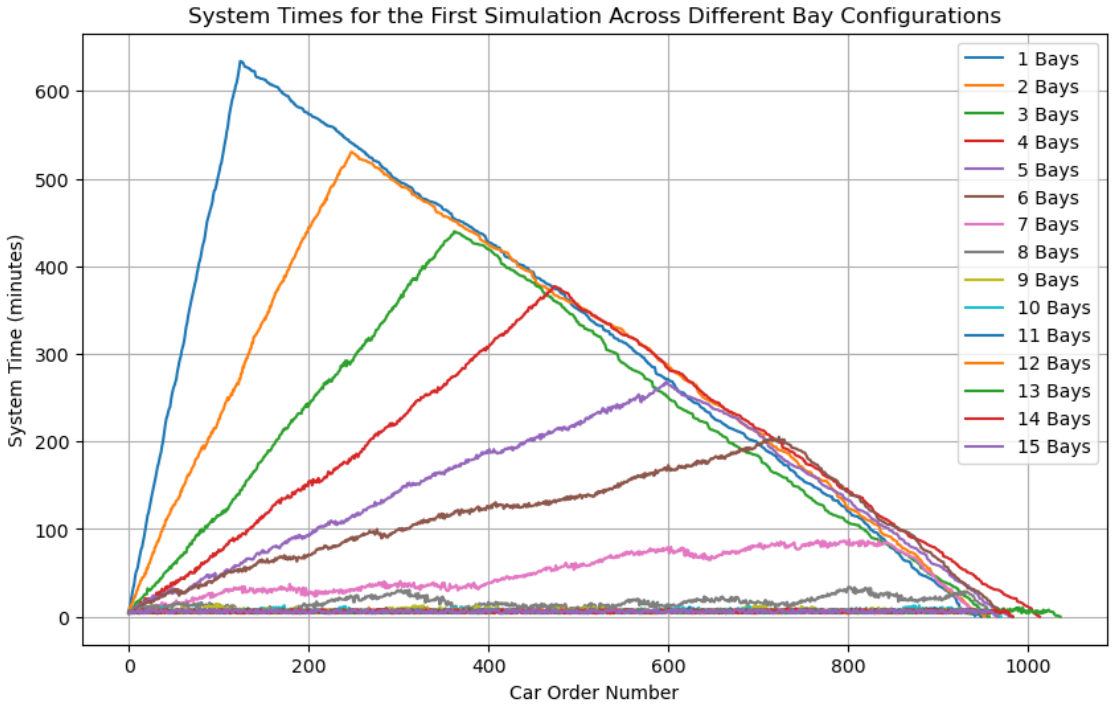


When encorperating a busy period for 3 hours in the middle of the 12 hour operation, and after simulating 1000 operational days, the best system set up which minimizes total cost is to have 15 bays open.

The below graphs look to show the difference in sysetem times for each new car as it enters and completes (or does not complete) a system for each variation of bays. Each graph is only of the first of 1000 simulations. The purpose is to see the operational difference and spike in system times when looking at steady arrival versus a system that has a busy period.

The below graph is for the original system. You can see that systems with low bays that cannot handle the arrival rate have sysetm times increase all the way until new cars that arrive cannot finish before closure, then the times tapper off at the rate of new car arrivals, these car do not finish the system but each of their system times decrease as they arrive closer and closer to the system closure as they have less time to wait until being kicked out.

[ ]:
```
plt.figure(figsize=(10, 6))

for num_bays, all_runs in sim_results.items():
    first_simulation_times = all_runs[0]
    plt.plot(range(len(first_simulation_times)), first_simulation_times,␣
 ↪label=f'{num_bays} Bays')

plt.title('System Times for the First Simulation Across Different Bay␣
 ↪Configurations')
plt.xlabel('Car Order Number')
plt.ylabel('System Time (minutes)')
plt.legend()
plt.grid(True)
```



Here, in the second simulation with a busy period, a spike can be seen when the busy period begins, the slope of the increase is lower for systems with more bays.

[ ]:
```
plt.figure(figsize=(10, 6))

for num_bays, all_runs in sim_results_q4.items():
    first_simulation_times_q4 = all_runs[0]
    plt.plot(range(len(first_simulation_times_q4)), first_simulation_times_q4,␣
  ↪label=f'{num_bays} Bays')

plt.title('System Times for the Second Simulation Across Different Bay␣
  ↪Configurations')
plt.xlabel('Car Order Number')
plt.ylabel('System Time (minutes)')
plt.legend()
plt.grid(True)
```