DATA7202
Semester 1, 2025
Assignment 3

Branden Lee 44789813

INTRODUCTION

This project addresses the challenges of managing an e-commerce system by analysing the performance of a simplified online retailer. Currently, the retailer processes orders through distinct picking and packing stages and are then collected by a courier. There are several key elements that influence the system's performance including order arrival rates, order priority, service rates, the number of workers, and courier schedule. The primary objective of this study is to evaluate an online retailer system's performance and efficiency with current staffing levels and policies. By the end of the report, this study will allow us to measure:

1. the average total speed of the system (from arrival to packing),
2. the wait times for express orders,
3. the utilisation of all workers, and
4. the proportion of time the courier leaves empty.

By developing a discrete event simulation (DES) in Python, this system can be simulated to allow us to identify current performances. It will also provide insights into any critical issues, such as bottlenecks in services and under-utilisation or over-utilisation of workers, that can be problematic for operational efficiency and customer satisfaction. But first, the next section will detail the model of the simulation.

MODEL DESCRIPTION

System space and events

The system's state changes at specific points in time. When an event is processed, current time advances, one or more queue lengths can change, and one or more worker statuses can change. Thus, let the state space be $\{\boldsymbol{X}_t\}_{\{t\geq 0\}}$ where

$$\boldsymbol{X}_t = \left(t, Q_{express}, Q_{standard}, Q_{pack}, Q_{courier}\ Pick_{busy}, Pack_{busy}\right)$$

where:

- t is current time
- $Q_{express}$ is the number of orders in the picking queue that are express orders
- $Q_{standard}$ is the number of orders in the picking queue that are standard orders
- Q_{pack} is the number of orders in the packing queue
- $Q_{courier}$ is the number of orders ready to be couriered
- $Pick_{busy}$ represents the picking workers' status, e.g. (true, false, false) or (1, 0, 0)
- $Pack_{busy}$ represents the packing workers' status

The events that change the system state include:

1. Arrival – where an order arrives in the system and is added to the picking queue
2. Picking complete – where an order is picked and added to the packing queue
3. Packing complete – where an order is packed and added to the courier-ready queue
4. Courier arrival – where the courier arrives to pick up all orders in the courier-ready queue

It should be noted that all queues are first in first out (FIFO) but the picking queue can contain priority orders where they must be picked first. To simulate this, two separate picking queues are created for each type of order.

Distributions

The system has services that are determined by varying distributions. Orders arrive based on the Poisson process with rate $\lambda = 0.8$, or 0.8 orders per time unit:

$$\text{Inter-arrival time} \sim \text{Exp}(\lambda = 0.8).$$

This is equivalent to

$$\text{Inter-arrival time} \sim \text{Exp}\left(\text{scale} = \frac{1}{0.8}\right)$$

using the scale parameter $\beta = 1/\lambda$. In Python and using the numpy package, this is simulated by running `numpy.random.exponential(1/0.8)`. For the services, picking follows an exponential distribution with mean 3 time units and packing follows an exponential distribution with mean 2 time units.

$$S_{pick} \sim \text{Exp}(\mu_{pick} = 3), \qquad S_{pack} \sim \text{Exp}(\mu_{pack} = 2)$$

This means on average, picking will take 3 time units and packing will take 2 time units. This is also simulated by running `numpy.random.exponential(mu)`.

Parameters

`T = 6000`	Total simulation time
`burn_in = 1000`	Burn-in period
`N = 40`	Number of batches for analysis
`LAMBDA = 0.8`	Rate that orders arrive
`mu_picking = 3`	Picking time mean
`mu_packing = 2`	Packing time mean
`picking_workers = 3`	Number of workers for Picking
`packing_workers = 2`	Number of workers for Packing
`p = 0.25`	25% of orders are Express
`courier_interval = 20`	Courier arrives every 20 time units

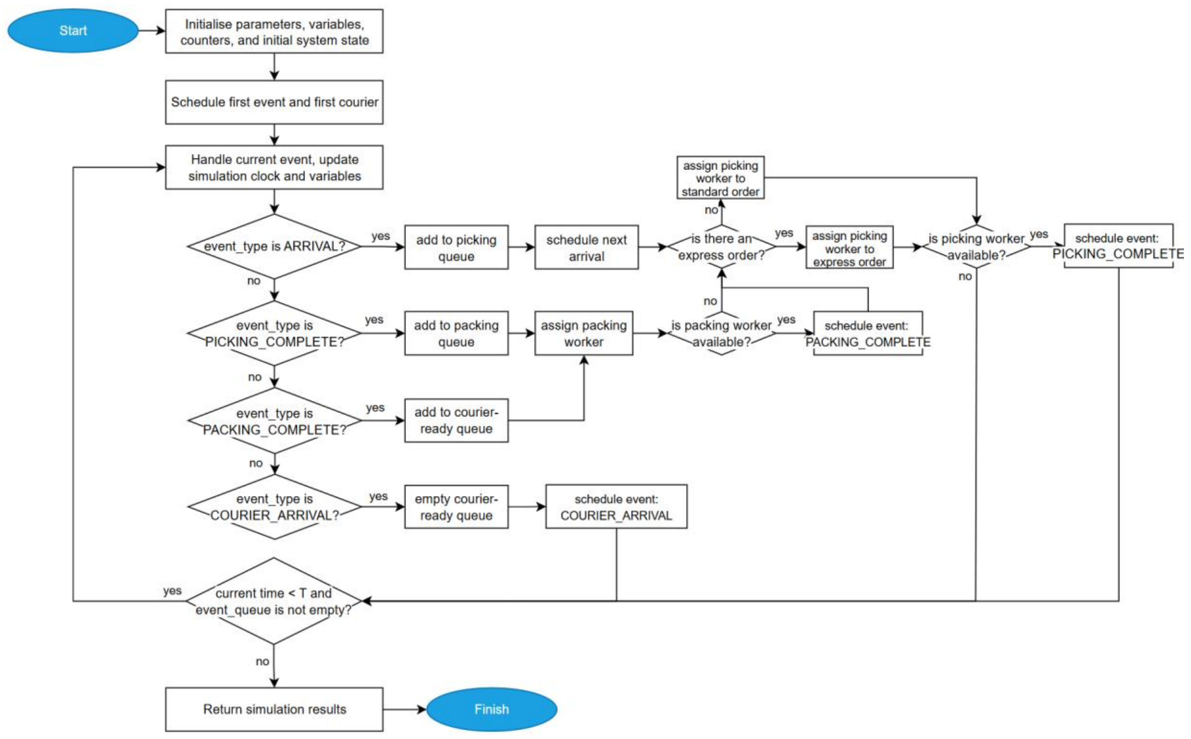
Variables

`current_time`	Current simulation time
`event_queue`	Priority queue for events
`picking_queue_express`	Picking queue for express orders
`picking_queue_standard`	Picking queue for standard orders

`packing_queue`	Packing queue after picking is done
`ready_for_courier`	Queue for packed orders ready to be couriered
`picking_worker_status`	Status of available picking workers
`packing_worker_status`	Status of available packing workers

DES Process

The diagram below demonstrates the general process of the simulation in Python.



Start
Initialise parameters, variables, counters, and initial system state
Schedule first event and first courier
Handle current event, update simulation clock and variables
event_type is ARRIVAL?
yes
add to picking queue
schedule next arrival
is there an express order?
yes
assign picking worker to express order
no
assign picking worker to standard order
is picking worker available?
yes
schedule event: PICKING_COMPLETE
no
event_type is PICKING_COMPLETE?
yes
add to packing queue
assign packing worker
is packing worker available?
yes
schedule event: PACKING_COMPLETE
no
event_type is PACKING_COMPLETE?
yes
add to courier-ready queue
no
event_type is COURIER_ARRIVAL?
yes
empty courier-ready queue
schedule event: COURIER_ARRIVAL
current time < T and event_queue is not empty?
yes
no
Return simulation results
Finish

This process includes initialising the parameters and variables as well as scheduling the first arrival and courier. Following this is the while loop which is the primary section that simulates the system. The while loop contains the following steps:

1. Updating simulation time/clock and variables
2. Handling the current event – depending on the event type, the process will change:
    a. Arrival – order has arrived
        i. Add order to picking queue
        ii. Schedule the next order arrival
        iii. Assign picking worker and if queue has express orders, pick them first
        iv. If picking worker is available, order picking is complete
    b. Picking complete – order has been picked
        i. Add order to packing queue
        ii. Assign packing worker
        iii. If worker is available, order packing is complete
        iv. Picking worker for this order is now free, so assign a new order in the picking queue to the picking worker
    c. Packing complete – order has been packed
        i. Add to courier-ready queue

        ii. Packing worker is now free, so assign a new order in the packing queue to the picking worker
    d. Courier arrival – courier has arrived
        i. Take/clear all orders in courier-ready queue
        ii. Schedule next courier arrival
3. Repeat steps 1 and 2 until time is greater than T (6,000)

Once the loop finishes, we will be able to generate the results and analyse them.

RESULTS AND ANALYSIS

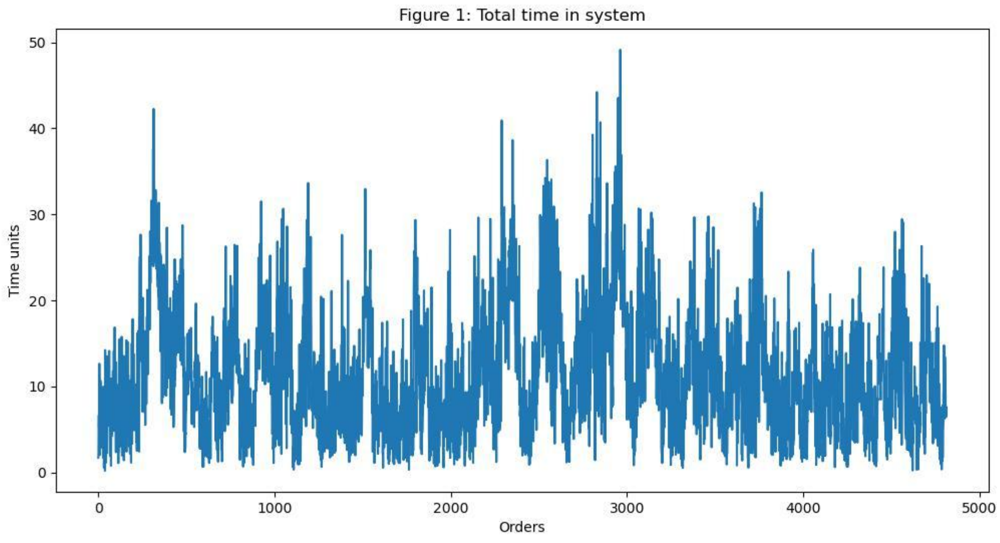
This section will display the results of the DES model simulated in Python and address the four key evaluation questions during the analysis. These questions are:

1. What is the average total time in the system (from arrival to packing complete)?
2. What proportion of express orders wait more than 6 time units before picking begins?
3. What is the utilisation of all workers?
4. What proportion of time does the courier leave empty?

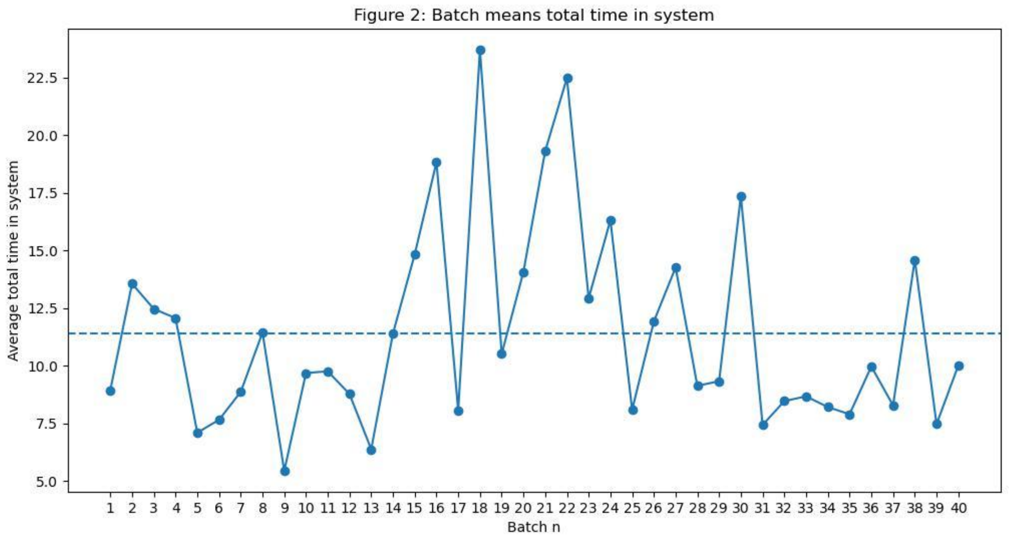
Answering these questions will provide insights into the performance and the speed of the system and may also highlight potential areas of improvement.

1. Average total time in the system

To check if the system is in a stable state, the total time taken for each completed order is collected and visualised onto a plot. Figure 1 demonstrates that the system shows no visible trends. Also, the time taken for each order does not continuously increase or decrease. Instead, any significant changes to the time taken are swiftly brought back to normal levels. With this finding, the batch means method can be used to be find the estimated average and 95% confidence intervals. Rather than executing multiple simulation runs, the batch means method only requires one long simulation run to discard an initial amount as burn-in or warm-up and then divide it into batches. This will provide a more accurate mean and confidence interval. One advantage to using this method is the fact it uses one long simulation run to find long-run averages. One disadvantage is that batches may show signs of correlation.



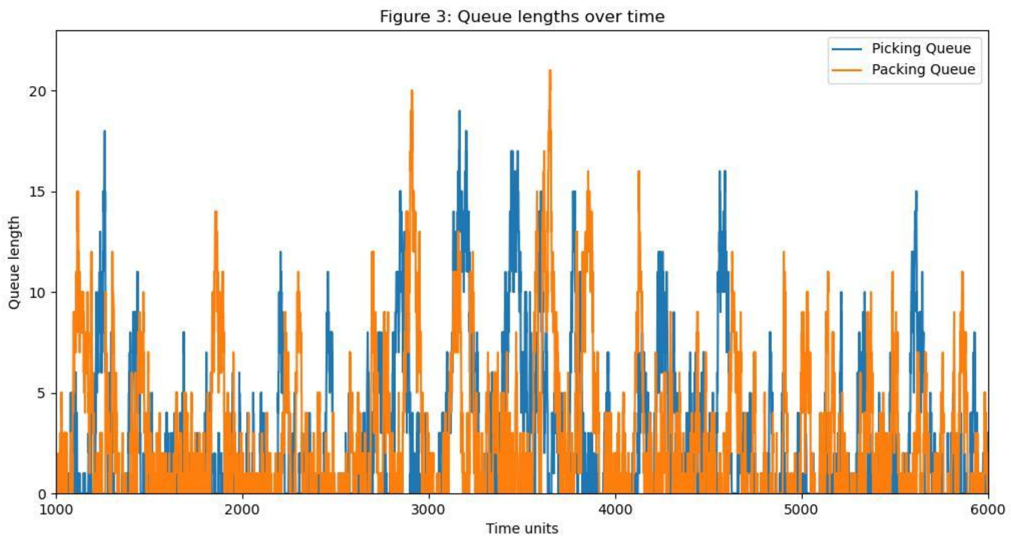
After discarding the initial 1000 time units as burn-in, the remainder of the simulation is divided into 40 non-overlapping batches. By computing the mean average for each batch, it allowed us to find that the estimated average total time in the system for an order was 11.4004 time units with a 95% confidence interval of (10.0565, 12.7442). Figure 2 visualises the means found for each batch with the dotted blue line representing the overall mean.



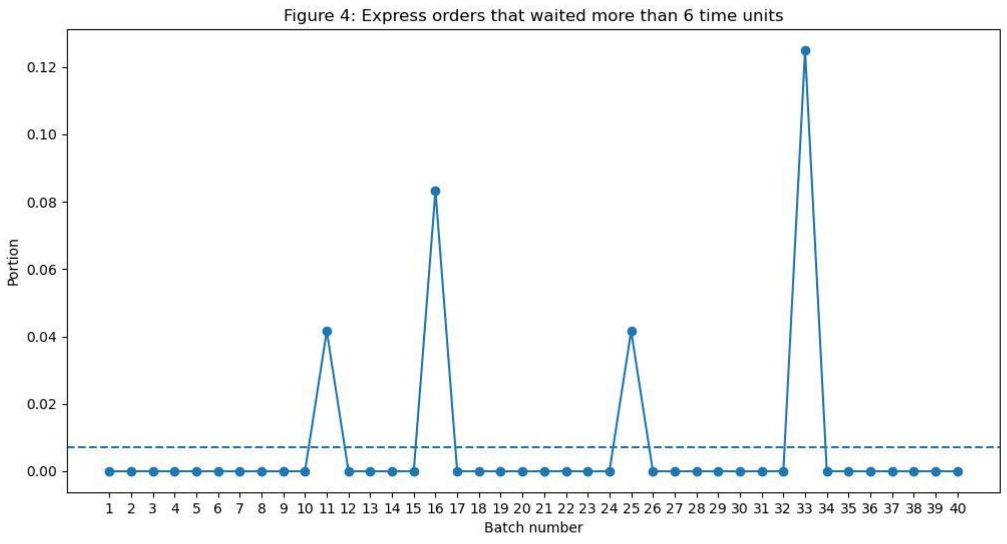
While unrealistic, without any queues, an order should only take a little over 5 time units to go through the system. This can be seen if the rate of arrival is changed to 1/3 to match the rate of service, which resulted in an average total time of 5.3 time units. However, in a realistic simulation like in this study, the average time taken is found to be over double this. Key factors to understand why this occurs was explored further.

2. Express order wait times

A key factor investigated was the queues in the system. Orders can potentially not flow through system smoothly if the queues suffer from congestion. Congestions can occur if the arrival rate of new orders to a service is greater than the rate at which it can be served. This becomes a critical issue if the arrival rate is consistently greater than the service rate. However, Figure 3 visualises the queue lengths over time and demonstrates that the queues are showing only temporary congestion due to the randomness and variability of the rates. It can be seen that picking and packing queue lengths are similar throughout.

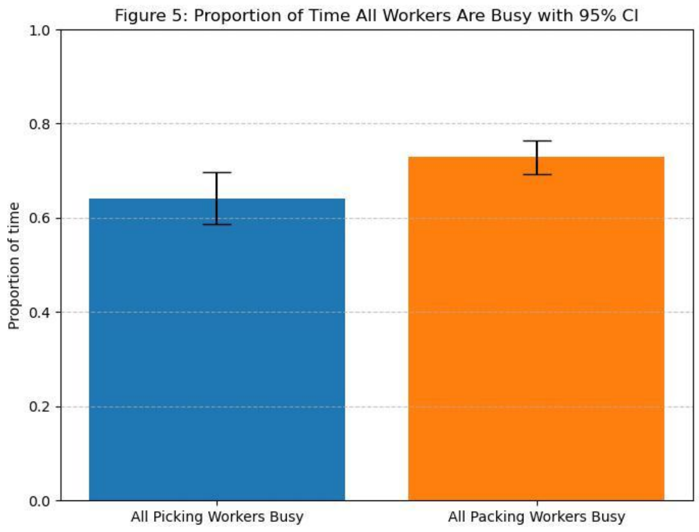


Drilling down into the queues, the lengths of the express queues can be found. Specifically, we find the estimated proportion of express orders that waited more than 6 time units is 0.00729 with a 95% confidence interval of (-0.00038, 0.01497). Figure 4 shows 4 out of the 40 batches had a small portion of express orders waiting more than 6 time units, but most batches showed no express orders waited that long. These results suggest that the likelihood of express orders waiting more than 6 time units is less than 1% and extremely unlikely.



3. Utilisation of all workers

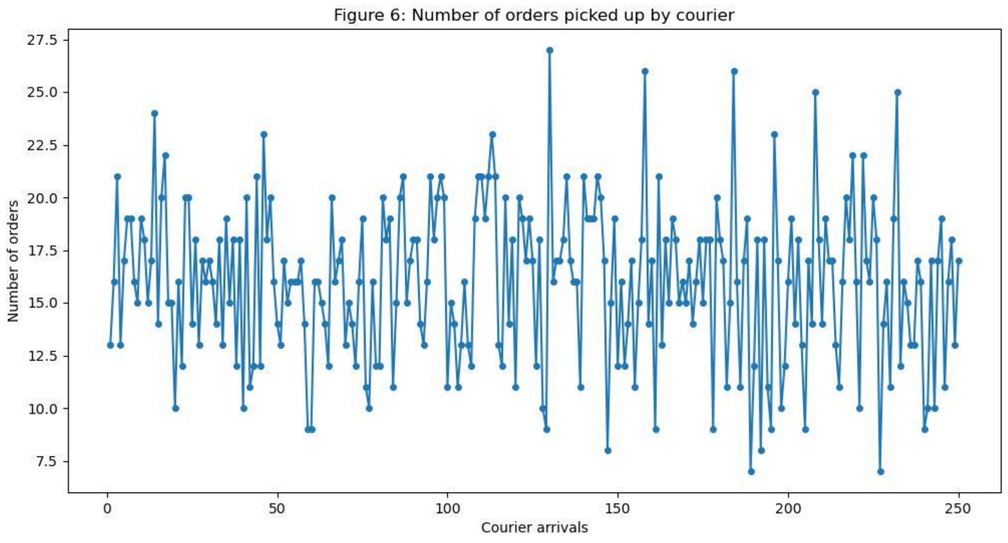
Another key factor are the workers for the services. For the queues to not become congested, the services need to be able to consistently handle the incoming orders and more workers will allow the system to do so. Currently, there are 3 workers for picking and 2 workers for packing. The proportion of time where all workers are busy for each service were found by using a step function estimator. Using this estimator is more appropriate here as the utilisation of the workers changes at discrete points in time and holds that value until the next change. Hence, the utilisation of workers is inherently a step function. With the current working rates, the estimated proportion of time where all picking workers were busy is 64.05% with a 95% confidence interval of (0.5850, 0.6961). For packing, the estimated proportion of time where all packing workers were busy is 72.75% with a confidence interval of (0.6924, 0.7627). The remaining proportion is the proportion of time where at least one worker is available or idle. Figure 5 visualises these results.



Overall, packing workers seem to be slightly busier than packing workers. Furthermore, all picking workers are busy only about 64% of the time. Ideally, these proportions should be higher. This suggests that the rate of arrival is currently not high enough to fully utilise all picking workers or the service rate of the picking workers is too fast. However, in a scenario where the number of picking workers is reduced to 2, the system becomes unstable as the picking workers become too busy to handle the incoming orders, resulting in a bottleneck.

4. Proportion of time courier leaves empty

We also want to investigate the courier and its efficiency. Is the system processing enough orders for the courier to be efficiently transporting enough orders? Figure 6 visualises the number of orders that the courier picks up each time it arrives. It shows the courier rarely picks up a low number of orders, with the average being approximately 16 orders. More importantly, no couriers left empty. Thus, the proportion of time the courier leaves empty is simply 0, with a confidence interval of (0, 0).



Summary

	Mean	*95% Confidence Interval*
Average total time in the system	11.4003	(10.0565, 12.7442)
Proportion of express orders waiting more than 6 time units	0.0073	(-0.0004, 0.0150)
Utilisation of all workers		
1. Picking	0.6405	(0.5850, 0.6961)
2. Packing	0.7275	(0.6924, 0.7627)
Proportion of time courier leaves empty	0	(0, 0)

CONCLUSIONS

In conclusion, with the current rate of arrivals for orders, the system is found to be stable with no critical issues present. There are several reasons why we found the system is stable and working well. Firstly, an order will take a reasonable 11.4 time units to go through the system on average. Secondly, waiting times for orders are low, with only 0.73% of express orders waiting more than 6 time units. Thirdly, workers are moderately utilised for both services. 64.09% of the total system time sees all picking workers busy, and 72.86% of the total system time has all packing workers busy. Finally, the courier never leaves empty. While the system is stable and has no indefinitely growing queues, it can be improved by reducing the average time in the system. This can be done by improving the rates for both picking and packing services. Even a small 10 percent improvement in the rates will reduce the time in the system to approximately 8.2 time units, which is about a 30 percent improvement. Worker utilisation can be potentially optimised, especially if orders arrive at a faster rate, but this requires careful management. If worker utilisation is too high, it suggests the system is taking in more orders than it can handle. If it’s too low, the workers are not being utilised enough.

APPENDIX

```
import heapq
import numpy as np
from collections import deque
import matplotlib.pyplot as plt
from scipy.stats import norm

# === Parameters / Constants ===
T = 6000
burn_in = 1000
N = 40

LAMBDA = 0.8 # order arrival rate
p = 0.25 # express rate

picking_workers = 3 # number of picking workers
mu_picking = 3 # picking worker rate

packing_workers = 2 # number of packing workers
mu_packing = 2 # packing worker rate

courier_interval = 20 # courier arrives every 20 time units

# --- Order Class ---
class Order:
    def __init__(self, arrival_time, order_type):
        self.arrival_time = arrival_time
        self.order_type = order_type
        self.picking_start_time = None
        self.picking_complete_time = None
        self.packing_start_time = None
        self.packing_complete_time = None
        self.time_in_system = None
        self.picking_wait_time = None

    def __repr__(self):
        return f"Order(Type={self.order_type}, Arrived={self.arrival_time:.2f})"

# === Variables ===
current_time = 0.0

event_queue = [] # (time, event_type, event_data_dict)

# Queues
picking_queue_express = deque()
picking_queue_standard = deque()
packing_queue = deque()
ready_for_courier = deque()

# Worker Status (True = Busy, False = Idle)
picking_workers_status = [False] * picking_workers
packing_workers_status = [False] * packing_workers

# -- Statistics --
total_orders_completed = 0
total_time_in_system_sum = 0.0
express_orders_picking_wait_count = 0
express_orders_picking_wait_gt_6_count = 0

picking_worker_busy_time = [0.0] * picking_workers
packing_worker_busy_time = [0.0] * packing_workers

courier_empty_count = 0 # number of times courier leaves empty
```

```
courier_arrival_count = 0 # number of times courier arrives

# For Question 3 -- Step function estimator
picking_all_busy_values = []
picking_all_busy_steps = []
packing_all_busy_values = []
packing_all_busy_steps = []

# For Question 4
courier_data = []

completed_orders = []

# For plotting
time_arr = []
picking_express_queue_len = []
picking_standard_queue_len = []
packing_queue_len = []


# Helper Functions
def schedule_event(time, event_type, event_data=None):
    if event_data is None:
        event_data = {}  # will contain data like worker ID and order ID
    heapq.heappush(event_queue, (time, event_type, event_data))

def get_idle_worker(worker_status_list):
    for i, status in enumerate(worker_status_list):
        # if worker is availble, return their id
        if not status:
            return i
    # worker is not available
    return -1

def assign_picking_worker():
    global express_orders_picking_wait_count, express_orders_picking_wait_gt_6_count

    # if express queue is not empty
    if picking_queue_express:
        order = picking_queue_express[0] # get first order
        # checks if workers are available
        worker_id = get_idle_worker(picking_workers_status)
        if worker_id != -1: # if worker is available
            picking_workers_status[worker_id] = True
            order = picking_queue_express.popleft()
            order.picking_start_time = current_time

            # update wait time to get picked
            order.picking_wait_time = order.picking_start_time - order.arrival_time

            # for TASK 2
            if order.order_type == "EXPRESS":
                express_orders_picking_wait_count += 1
                if order.picking_wait_time > 6:
                    express_orders_picking_wait_gt_6_count += 1

            picking_time = np.random.exponential(scale = mu_picking)
            schedule_event(current_time + picking_time, "PICKING_COMPLETE",
                           {"worker_id": worker_id, "order": order})
            return True
    # if express queue is empty
    elif picking_queue_standard:
        order = picking_queue_standard[0]
        worker_id = get_idle_worker(picking_workers_status)
        if worker_id != -1:
            picking_workers_status[worker_id] = True
            order = picking_queue_standard.popleft()
            order.picking_start_time = current_time

            picking_time = np.random.exponential(mu_picking)
            schedule_event(current_time + picking_time, "PICKING_COMPLETE",
                           {"worker_id": worker_id, "order": order})
            return True
    # else no workers available
    return False

def assign_packing_worker():
    # if packing queue is not empty
    if packing_queue:
        # get first order in queue
        order = packing_queue[0]
        # get available packing worker
        worker_id = get_idle_worker(packing_workers_status)
        # if a worker is available, start packing
        if worker_id != -1:
            packing_workers_status[worker_id] = True
            order = packing_queue.popleft()

            # track packing time
            order.packing_start_time = current_time
            packing_time = np.random.exponential(mu_packing)
            schedule_event(current_time + packing_time, "PACKING_COMPLETE",
                           {"worker_id": worker_id, "order": order})
            return True
    # else, no workers available or packing queue is empty
    return False

# === Event Handlers ===
def handle_arrival(event_data):
    # For simulation, randomly assign express or standard type
    order_type = "EXPRESS" if np.random.rand() < p else "STANDARD"
    new_order = Order(current_time, order_type)

    # add order to express queue or standard queue
    if new_order.order_type == "EXPRESS":
        picking_queue_express.append(new_order)
    else:
        picking_queue_standard.append(new_order)

    # schedule next arrival
    next_arrival_time = current_time + np.random.exponential(1/LAMBDA)
    schedule_event(next_arrival_time, "ARRIVAL")

    # assign 1 of 3 workers for PICKING
    assign_picking_worker()

def handle_picking_complete(event_data):
    # get picking worker
    worker_id = event_data["worker_id"]
    order = event_data["order"]

    # set picking worker to available
    picking_workers_status[worker_id] = False
    order.picking_complete_time = current_time

    # add order to packing queue
    packing_queue.append(order)
    # assign packing worker
    assign_packing_worker()

    # since worker is available, assign worker to another order
    assign_picking_worker()

def handle_packing_complete(event_data):
    global total_orders_completed, total_time_in_system_sum
    # get info from order
    worker_id = event_data["worker_id"]
    order = event_data["order"]

    # set worker to be available
    packing_workers_status[worker_id] = False
    # set packing completion time
    order.packing_complete_time = current_time

    # calculate total time in system
    order.time_in_system = order.packing_complete_time - order.arrival_time
    ready_for_courier.append(order)
    completed_orders.append(order)

    # statistics
    if current_time >= burn_in:
        total_time_in_system_sum += order.time_in_system
        total_orders_completed += 1

    # packing worker is now free, so assign new order
    assign_packing_worker()

def handle_courier_arrival(event_data):
    global courier_empty_count, courier_arrival_count

    # number of orders ready to be couriered
    orders_shipped = len(ready_for_courier)

    if current_time > burn_in:
        # store data for plotting
        courier_data.append(orders_shipped)

    # track number of courier arrivals
    courier_arrival_count += 1

    if not ready_for_courier: # if no orders are ready
        courier_empty_count += 1
    else: # else, order(s) are ready
        ready_for_courier.clear()

    # schedule the next courier to arrive
    schedule_event(current_time + courier_interval, "COURIER_ARRIVAL")

# --- Simulation Run ---
def run_simulation():
    global current_time

    # Initial events
    schedule_event(np.random.exponential(LAMBDA), "ARRIVAL")
    schedule_event(courier_interval, "COURIER_ARRIVAL")

    while event_queue and current_time < T:
        # get next event
        time, event_type, event_data = heapq.heappop(event_queue)

        time_diff = time - current_time
        # to track worker utilisation
        if time_diff > 0:
            for i in range(picking_workers):
                if picking_workers_status[i] and current_time >= burn_in:
                    picking_worker_busy_time[i] += time_diff
            for i in range(packing_workers):
                if packing_workers_status[i] and current_time >= burn_in:
                    packing_worker_busy_time[i] += time_diff

            # Tracks when all workers are busy after burn in
            if time >= burn_in:
                if all(picking_workers_status):
                    picking_all_busy_values.append(1)
                else:
                    picking_all_busy_values.append(0)
                picking_all_busy_steps.append(time)
                if all(packing_workers_status):
                    packing_all_busy_values.append(1)
                else:
                    packing_all_busy_values.append(0)
                packing_all_busy_steps.append(time)

        current_time = time

        # for statistics
        time_arr.append(time)
        picking_express_queue_len.append(len(picking_queue_express))
        picking_standard_queue_len.append(len(picking_queue_standard))
        packing_queue_len.append(len(packing_queue))

        # handle events based on type
        if event_type == "ARRIVAL":
            handle_arrival(event_data)
        elif event_type == "PICKING_COMPLETE":
            handle_picking_complete(event_data)
        elif event_type == "PACKING_COMPLETE":
            handle_packing_complete(event_data)
        elif event_type == "COURIER_ARRIVAL":
            handle_courier_arrival(event_data)

    # final update for worker busy times at the end of the simulation
    time_diff_final = T - current_time
    if time_diff_final > 0:
        for i in range(picking_workers):
            if picking_workers_status[i] and current_time >= burn_in:
                picking_worker_busy_time[i] += time_diff_final
        for i in range(packing_workers):
            if packing_workers_status[i] and current_time >= burn_in:
                packing_worker_busy_time[i] += time_diff_final


# based on batchmeans.py file
# assumes data burn in already discarded
def batch_means_ci(data, N, alpha=0.05):
    batch_size = len(data) // N
    batch_means = []
    for i in range(N):
        start = i * batch_size
        end = start + batch_size
        batch = data[start:end]
        batch_means.append(np.mean(batch))
    batch_means = np.array(batch_means)
    # compute confidence interval
    mu = np.mean(batch_means)
    s = np.std(batch_means, ddof=1)
    z = norm.ppf(1 - alpha / 2)
    ci = (mu - z * s / np.sqrt(N), mu + z * s / np.sqrt(N))
    return mu, ci, batch_means

# function to find estimate and CI for Q2
def batch_means_q2(data, N, alpha=0.05):
    batch_size = len(data) // N
    batch_means = []
    for i in range(N):
        start = i * batch_size
        end = start + batch_size
        batch = data[start:end]
        batch = np.array(batch)
        batch_means.append(np.mean(batch > 6)) # find P(t > 6)
    batch_means = np.array(batch_means)
    # compute estimate and CI
    mu = np.mean(batch_means) # estimate
    s = np.std(batch_means, ddof=1)
    z = norm.ppf(1 - alpha / 2)
    ci = (mu - z * s / np.sqrt(N), mu + z * s / np.sqrt(N))
    return mu, ci, batch_means

# function to find estimate and CI for Q4
def batch_means_q4(data, N, alpha=0.05):
    batch_size = len(data) // N
    batch_means = []
    for i in range(N):
        start = i * batch_size
        end = start + batch_size
        batch = data[start:end]
        batch = np.array(batch)
        # find when courier leaves empty
        batch_means.append(np.mean(batch == 0))
    batch_means = np.array(batch_means)
    # compute estimate and CI
    mu = np.mean(batch_means) # estimate
    s = np.std(batch_means, ddof=1)
    z = norm.ppf(1 - alpha / 2)
    ci = (mu - z * s / np.sqrt(N), mu + z * s / np.sqrt(N))
    return mu, ci, batch_means

    # function to find estimate and CI for Q4
    def batch_means_q4_2(data, N, alpha=0.05):
        batch_size = len(data) // N
        batch_means = []
        for i in range(N):
            start = i * batch_size
            end = start + batch_size
            batch = data[start:end]
            batch = np.array(batch)
            # find when courier leaves empty
            batch_means.append(np.mean(batch))
        batch_means = np.array(batch_means)
        # compute estimate and CI
        mu = np.mean(batch_means) # estimate
        s = np.std(batch_means, ddof=1)
        z = norm.ppf(1 - alpha / 2)
        ci = (mu - z * s / np.sqrt(N), mu + z * s / np.sqrt(N))
        return mu, ci, batch_means

# from mm1queue.py file, assumes burn-in is discarded
def step_func_ci(values, steps, N):
    batch_size = len(values) // N
    estimates = []
    for i in range(N):
        x = values[i*batch_size:(i+1)*batch_size] # step values
        dt = steps[i*batch_size:(i+1)*batch_size] # delta
        estimates.append(np.sum(x * dt) / np.sum(dt))
    mu = np.mean(estimates)
    s = np.std(estimates, ddof=1)
    z = norm.ppf(0.975)
    ci = (mu - z * s / np.sqrt(N), mu + z * s / np.sqrt(N))
    return mu, ci, estimates

if __name__ == "__main__":
    np.random.seed(1) # for reproducibility
    run_simulation()

    print("Simulation Results (after burn-in)")

    # (1) Average total time in the system
    avg_total_time_in_system = total_time_in_system_sum / total_orders_completed
    print(f"(1) Average total time in the system (from arrival to packing complete):
{avg_total_time_in_system:.2f} time units")

    # (2) Proportion of Express orders wait more than 6 time units before picking begins
    express_wait_gt_6_prop = express_orders_picking_wait_gt_6_count /
express_orders_picking_wait_count
    print(f"(2) Proportion of Express orders wait more than 6 time units before picking begins:
{express_wait_gt_6_prop:.4f}")

    # (3) Utilization of all workers
    effective_simulation_duration = T - burn_in

    picking_utilization = sum(picking_worker_busy_time) / (picking_workers *
effective_simulation_duration)
    packing_utilization = sum(packing_worker_busy_time) / (packing_workers *
effective_simulation_duration)

    print(f"(3) Utilization of all workers:")
    print(f"    - Picking workers average utilization: {picking_utilization:.4f}")
    print(f"    - Packing workers average utilization: {packing_utilization:.4f}")

    print("    Individual Picking Worker Utilization:")
    for i, busy_time in enumerate(picking_worker_busy_time):
        individual_util = busy_time / effective_simulation_duration
        print(f"        Worker {i+1}: {individual_util:.4f}")

    print("    Individual Packing Worker Utilization:")
    for i, busy_time in enumerate(packing_worker_busy_time):
        individual_util = busy_time / effective_simulation_duration
        print(f"        Worker {i+1}: {individual_util:.4f}")

    # (4) Proportion of time the Courier leaves empty
    if courier_arrival_count > 0:
        courier_empty_prop = courier_empty_count / courier_arrival_count
    else:
        courier_empty_prop = 0.0
    print(f"(4) Proportion of time the Courier leaves empty: {courier_empty_prop:.4f}")

    # === Plotting and Estimates ===
    print("\nPlotting and Esimates")
    # -- Figure 1 --
    # plotting to find total time taken in orders
    total_times = []
    for order in completed_orders:
        total_time = order.time_in_system
        total_times.append(total_time)

    index = np.array(list(range(len(total_times))))
    plt.figure(figsize=(12,6))
    plt.plot(index, total_times)
    plt.xlabel("Orders")
    plt.ylabel("Time units")
    plt.title("Figure 1: Total time in system")
    # plt.savefig("Fig1.jpg")
    plt.show()

    # -- Figure 2 --
    # get total time for each completed order in the system
    total_times = []
    for order in completed_orders:
        if order.arrival_time > burn_in: # check order is after burn in
            total_time = order.time_in_system
            total_times.append(total_time)

    mu, ci, batch_means = batch_means_ci(total_times, 40)
    print(f"(Q1) Average total time in the system {mu:.6f} CI=({ci[0]:.6f}, {ci[1]:6f})")

    batch_indices = np.arange(1, N+1)
    plt.figure(figsize=(12, 6))
    plt.plot(batch_indices, batch_means, marker="o", linestyle="-")
    plt.axhline(y=mu, linestyle="--", label=f"Mean = {mu:.2f}")
    plt.xticks(batch_indices)
    plt.xlabel("Batch n")
    plt.ylabel("Average total time in system")
    plt.title("Figure 2: Batch means total time in system")
    # plt.savefig("Fig2.jpg")
    plt.show()

    # -- Figure 3 --
    # plotting for queue lengths
    picking_queue_len = np.array(picking_express_queue_len) + np.array(picking_standard_queue_len)

    plt.figure(figsize=(12,6))
    plt.plot(time_arr, picking_queue_len, label="Picking Queue")
    plt.plot(time_arr, packing_queue_len, label="Packing Queue")
    plt.xlabel("Time units")
    plt.ylabel("Queue length")
    plt.xlim(1000, 6000)
    plt.ylim(0, 23)
    plt.title("Figure 3: Queue lengths over time")
    plt.legend()
    # plt.savefig("Fig3.jpg")
    plt.show()

    # -- Figure 4 --
    # find all express orders that was in the system and collect their waiting times
    express_order_wait_times = []
    for order in completed_orders:
        if order.arrival_time > 1000 and order.order_type == "EXPRESS":
            express_order_wait_times.append(order.picking_wait_time)

    # find the total estimate, CI, and estimates for each batch
    mu, ci, batch_means = batch_means_q2(express_order_wait_times, N)
    print(f"(Q2) Estimated portion of express orders that waited more than 6 time units: {mu:.6f}
CI=({ci[0]:.6f}, {ci[1]:6f})")

    # plotting
    batch_indices = np.arange(1, N+1)
    plt.figure(figsize=(12, 6))
    plt.plot(batch_indices, batch_means, marker="o", linestyle="-")
    plt.axhline(y=mu, linestyle="--", label=f"Mean = {mu:.4f}")
    plt.xticks(batch_indices)
    plt.xlabel("Batch number")
    plt.ylabel("Portion")
    # plt.xlim(1,40)
    # plt.ylim(0.00, 0.13)
    plt.title("Figure 4: Express orders that waited more than 6 time units")
    # plt.savefig("Fig4.jpg")
    plt.show()

    # -- Figure 5 --
    picking_busy_vals = np.array(picking_all_busy_values)
    picking_busy_vals = picking_busy_vals[:-1]
    picking_delta_times = np.diff(picking_all_busy_steps)

    packing_busy_vals = np.array(packing_all_busy_values)
    packing_busy_vals = packing_busy_vals[:-1]
    packing_delta_times = np.diff(packing_all_busy_steps)

    print("(Q3)")

    pick_est, pick_ci, pick_estimates = step_func_ci(picking_busy_vals, picking_delta_times, N)
    print(f"Proportion of time for all picking workers being utilized {pick_est:.5f}
CI=({pick_ci[0]:.5f}, {pick_ci[1]:.5f})")

    pack_est, pack_ci, pack_estimates = step_func_ci(packing_busy_vals, packing_delta_times, N)
    print(f"Average proportion of time for all packing workers being utilized {pack_est:.5f}
CI=({pack_ci[0]:.5f}, {pack_ci[1]:.5f})")

    # -- Plotting --
    categories = ["All Picking Workers Busy", "All Packing Workers Busy"]

    means = [pick_est, pack_est]

    # find errors with confidence intervals for plotting
    picking_error_lower = pick_est - pick_ci[0]
    picking_error_upper = pick_ci[1] - pick_est
    packing_error_lower = pack_est - pack_ci[0]
    packing_error_lower = pack_ci[1] - pack_est
    errors = [
        [picking_error_lower, packing_error_lower],
        [picking_error_upper, packing_error_lower]
    ]

    plt.figure(figsize=(8, 6))
    plt.bar(categories, means, yerr=errors, capsize=10, color=["C0", "C1"])
    plt.ylabel("Proportion of time")
    plt.title("Figure 5: Proportion of Time All Workers Are Busy with 95% CI")
    plt.ylim(0, 1) # Proportions are between 0 and 1
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    # plt.savefig("Fig5.jpg")
    plt.show()

    # -- Figure 6 --
    mu, ci, batch_means = batch_means_q4(courier_data, N)
    print(f"(Q4) Estimated portion of time Courier leaves empty: {mu:.6f} CI=({ci[0]:.6f},
{ci[1]:6f})")

    # Plotting for Question 4
    index = np.arange(1, len(courier_data)+1)
    plt.figure(figsize=(12, 6))
    plt.plot(index, courier_data, marker="o", markersize=4)
    plt.xlabel("Courier arrivals")
    plt.ylabel("Number of orders")
    plt.title("Figure 6: Number of orders picked up by courier")
    # plt.savefig("Fig6.jpg")
    plt.show()

    mu, ci, batch_means = batch_means_q4_2(courier_data, N)
    print(f"Estimated courier load: {mu:.6f} CI=({ci[0]:.6f}, {ci[1]:6f})")
```