Data Structures and Algorithms (DSA)

Lab Report 8

Name:	Iqra Fatima
Reg. Number:	23-CP-62
Semester:	4th
Department:	CPED

Submitted To:

Engineer Sheharyar Khan

Guided Tasks

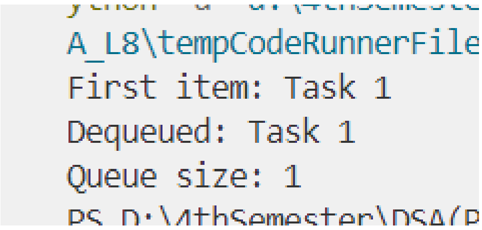
Example 1

Implementing a List-Based Queue

Code:

```
class ListQueue:
    def __init__(self):
        self.items = []

    def enqueue(self, data):
        """Add an item to the queue (end of list)."""
        self.items.insert(0, data)

    def dequeue(self):
        """Remove and return the first item from the queue."""
        if self.is_empty():
            print("Queue is empty!")
            return None
        return self.items.pop()

    def peek(self):
        """Return the first element without removing it."""
        return self.items[-1] if self.items else None

    def size(self):
        """Return the size of the queue."""
        return len(self.items)

    def is_empty(self):
        """Check if queue is empty."""
        return len(self.items) == 0

# Example Usage
queue = ListQueue()
queue.enqueue("Task 1")
queue.enqueue("Task 2")
print("First item:", queue.peek())
print("Dequeued:", queue.dequeue())
print("Queue size:", queue.size())
```

Output:



```
ython -u d:\4thSemeste
A_L8\tempCodeRunnerFile
First item: Task 1
Dequeued: Task 1
Queue size: 1
PS D:\4thSemester\DSA(P
```

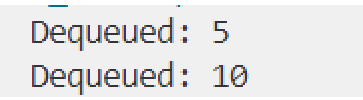
Example 2

Implementing a Stack-Based Queue

Code:

```
class StackQueue:
    def __init__(self):
        self.inbound_stack = []
        self.outbound_stack = []

    def enqueue(self, data):
        """Push data to inbound stack."""
        self.inbound_stack.append(data)

    def dequeue(self):
        """Move elements from inbound to outbound stack, then
pop."""
        if not self.outbound_stack:
            while self.inbound_stack:
                self.outbound_stack.append(self.inbound_stack.pop())
        return self.outbound_stack.pop() if self.outbound_stack else
None

# Example Usage
sq = StackQueue()
sq.enqueue(5)
sq.enqueue(10)
sq.enqueue(15)
print("Dequeued:", sq.dequeue())  # 5
print("Dequeued:", sq.dequeue())  # 10
```

Output:



```
Dequeued: 5
Dequeued: 10
```

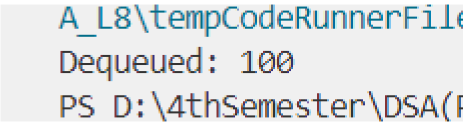
Example 3

Implementing a Node-Based Queue (Linked List)

Code:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedQueue:
    def __init__(self):
        self.head = None
        self.tail = None
        self.size = 0
    def enqueue(self, data):
        """Add an element to the tail of the queue."""
        new_node = Node(data)
        if self.tail:
            self.tail.next = new_node
            self.tail = new_node
        if self.head is None:
            self.head = new_node
        self.size += 1

    def dequeue(self):
        """Remove the front node."""
        if self.head is None:
            return None
        removed_data = self.head.data
        self.head = self.head.next
        if self.head is None:
            self.tail = None
        self.size -= 1
        return removed_data

# Example Usage
lq = LinkedQueue()
lq.enqueue(100)
lq.enqueue(200)
print("Dequeued:", lq.dequeue())  # 100
```

Output:



```
A_L8\tempCodeRunnerFil
Dequeued: 100
PS D:\4thSemester\DSA(
```

Exercise Questions

Easy Problems

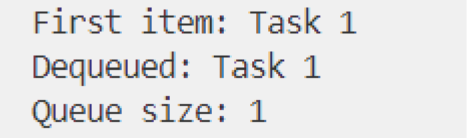
1. Basic Queue Operations:

Implement enqueue, dequeue, peek, and size functions.

Code:

```
class ListQueue:
    def __init__(self):
        self.items = []

    def enqueue(self, data):
        """Add an item to the queue (end of list)."""
        self.items.insert(0, data)

    def dequeue(self):
        """Remove and return the first item from the queue."""
        if self.is_empty():
            return "Queue is empty!"
        return self.items.pop()

    def peek(self):
        """Return the front element without removing it."""
        return self.items[-1] if not self.is_empty() else "Queue is
empty!"

    def size(self):
        """Return the size of the queue."""
        return len(self.items)

    def is_empty(self):
        """Check if the queue is empty."""
        return len(self.items) == 0

# Example Usage
queue = ListQueue()
queue.enqueue("Task 1")
queue.enqueue("Task 2")
print("First item:", queue.peek())  # Task 1
print("Dequeued:", queue.dequeue())  # Task 1
print("Queue size:", queue.size())  # 1
```

Output:



```
First item: Task 1
Dequeued: Task 1
Queue size: 1
```

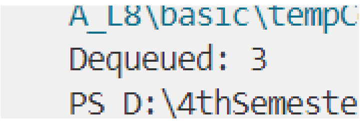
2. Reverse a Queue:

Implement a function to reverse a queue using a stack.

Code:

```
class StackQueue:
    def __init__(self):
        self.inbound_stack = []
        self.outbound_stack = []

    def enqueue(self, data):
        """Push data to inbound stack."""
        self.inbound_stack.append(data)

    def dequeue(self):
        """Move elements from inbound to outbound stack, then
pop."""
        if not self.outbound_stack:
            while self.inbound_stack:
                self.outbound_stack.append(self.inbound_stack.pop())
        return self.outbound_stack.pop() if self.outbound_stack else
None

def reverse_queue(q):
    stack = []
    while True:
        item = q.dequeue()
        if item is None:
            break
        stack.append(item)
    for item in reversed(stack):
        q.enqueue(item)

# Example Usage
sq = StackQueue()
sq.enqueue(1)
sq.enqueue(2)
sq.enqueue(3)
reverse_queue(sq)
print("Dequeued:", sq.dequeue())  # 1 (reversed order)
```

Output:



```
    A_L8\basic\tempC
    Dequeued: 3
    PS D:\4thSemeste
```

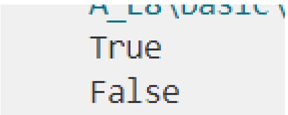
3. Check Palindrome using Queue:

Use a queue to check if a string is a palindrome.

Code:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedQueue:
    def __init__(self):
        self.head = None
        self.tail = None
        self.size = 0

    def enqueue(self, data):
        new_node = Node(data)
        if self.tail:
            self.tail.next = new_node
        self.tail = new_node
        if self.head is None:
            self.head = new_node
        self.size += 1

    def dequeue(self):
        if self.head is None:
            return None
        removed_data = self.head.data
        self.head = self.head.next
        if self.head is None:
            self.tail = None
        self.size -= 1
        return removed_data

def is_palindrome(string):
    q = LinkedQueue()
    for char in string:
        q.enqueue(char)

    stack = []
    temp = q.head
    while temp:
        stack.append(temp.data)
        temp = temp.next

    temp = q.head
    while temp:
        if temp.data != stack.pop():
            return False
        temp = temp.next
    return True

# Example Usage
print(is_palindrome("racecar"))  # True
print(is_palindrome("hello"))  # False
```

Output:



```
True
False
```

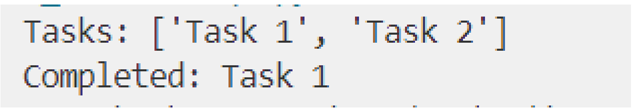
4. Queue-based Task Manager:

Implement a simple task manager using a queue.

Code:

```
class TaskManager:
    def __init__(self):
        self.tasks = []

    def add_task(self, task):
        self.tasks.insert(0, task)

    def complete_task(self):
        if self.tasks:
            return f"Completed: {self.tasks.pop()}"
        return "No tasks remaining."
    def view_tasks(self):
        return self.tasks[::-1]

# Example Usage
tm = TaskManager()
tm.add_task("Task 1")
tm.add_task("Task 2")
print("Tasks:", tm.view_tasks())  # ['Task 1', 'Task 2']
print(tm.complete_task())  # Completed: Task 1
```

<u>Output:</u>



```
Tasks: ['Task 1', 'Task 2']
Completed: Task 1
```

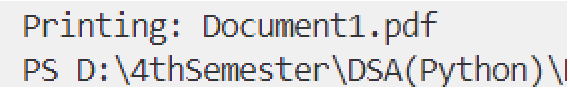
5. Print Jobs Simulation:

Simulate print job handling using a queue.

<u>Code:</u>

```
import time

class PrintQueue:
    def __init__(self):
        self.queue = []

    def add_job(self, job):
        self.queue.insert(0, job)

    def process_job(self):
        if self.queue:
            print(f"Printing: {self.queue.pop()}")
            time.sleep(1)
        else:
            print("No print jobs in queue.")

# Example Usage
pq = PrintQueue()
pq.add_job("Document1.pdf")
pq.add_job("Image.png")
pq.process_job()  # Printing: Document1.pdf
```

<u>Output:</u>



```
Printing: Document1.pdf
PS D:\4thSemester\DSA(Python)\
```

Intermediate Problems

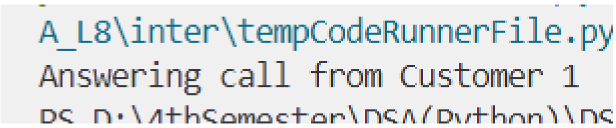
1. Call Center Simulation:

Implement a queue where customer service calls are answered in FIFO order.

Code:

```
class CallCenter:
    def __init__(self):
        self.calls = []

    def receive_call(self, caller):
        self.calls.insert(0, caller)

    def answer_call(self):
        if self.calls:
            return f"Answering call from {self.calls.pop()}"
        return "No calls in the queue."

# Example Usage
cc = CallCenter()
cc.receive_call("Customer 1")
cc.receive_call("Customer 2")
print(cc.answer_call())  # Customer 1
```

Output:



```
A_L8\inter\tempCodeRunnerFile.py
Answering call from Customer 1
PS D:\4thSemester\DSA(Python)\DS
```

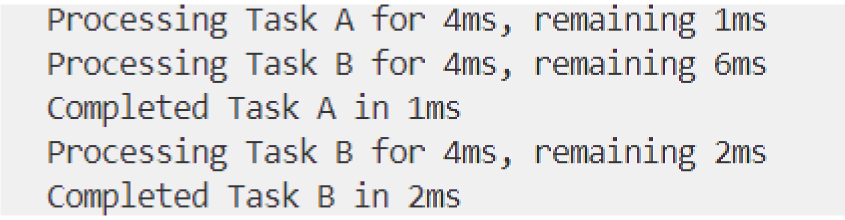
2. CPU Task Scheduling:

Implement Round Robin scheduling for CPU tasks using a queue.

Code:

```
def round_robin(tasks, quantum):
    queue = tasks[:]
    while queue:
        task, time = queue.pop(0)
        if time > quantum:
            print(f"Processing {task} for {quantum}ms, remaining
{time - quantum}ms")
            queue.append((task, time - quantum))
        else:
            print(f"Completed {task} in {time}ms")

# Example Usage
tasks = [("Task A", 5), ("Task B", 10)]
round_robin(tasks, 4)
```

Output:



```
Processing Task A for 4ms, remaining 1ms
Processing Task B for 4ms, remaining 6ms
Completed Task A in 1ms
Processing Task B for 4ms, remaining 2ms
Completed Task B in 2ms
```

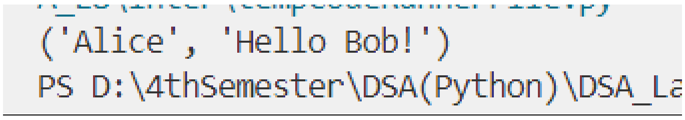
3. Message Queue System:

Implement a simple message-passing queue system between users.

Code:

```
class MessageQueue:
    def __init__(self):
        self.queue = []

    def send_message(self, sender, message):
        self.queue.insert(0, (sender, message))

    def receive_message(self):
        return self.queue.pop() if self.queue else "No messages."

# Example Usage
mq = MessageQueue()
mq.send_message("Alice", "Hello Bob!")
print(mq.receive_message())  # ('Alice', 'Hello Bob!')
```

Output:



```
('Alice', 'Hello Bob!')
PS D:\4thSemester\DSA(Python)\DSA_La
```

4. Queue-based Chat System:

Implement a chat message queue where messages are

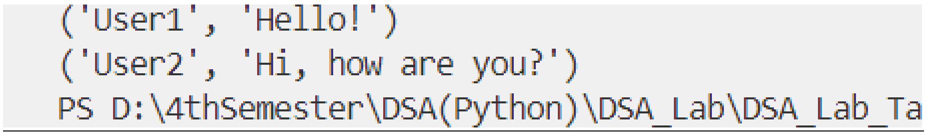
displayed in order.

Code:

```
class ChatQueue:
    def __init__(self):
        self.queue = []

    def send_message(self, sender, message):
        self.queue.insert(0, (sender, message))

    def receive_messages(self):
        while self.queue:
            print(self.queue.pop())
```

```
# Example Usage
chat = ChatQueue()
chat.send_message("User1", "Hello!")
chat.send_message("User2", "Hi, how are you?")
chat.receive_messages()
```

Output:



```
('User1', 'Hello!')
('User2', 'Hi, how are you?')
PS D:\4thSemester\DSA(Python)\DSA_Lab\DSA_Lab_Ta
```

5. Ride-Sharing Queue:

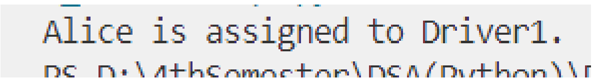
Simulate a queue system where passengers are assigned to

rides based on first-come, first-served.

Code:

```
class RideQueue:
    def __init__(self):
        self.queue = []

    def request_ride(self, passenger):
        self.queue.insert(0, passenger)

    def assign_ride(self, driver):
        if self.queue:
            return f"{self.queue.pop()} is assigned to {driver}."
        return "No passengers waiting."

# Example Usage
rq = RideQueue()
rq.request_ride("Alice")
print(rq.assign_ride("Driver1"))  # Alice is assigned to Driver1
```

Output:



```
Alice is assigned to Driver1.
PS D:\4thSemester\DSA(Python)\
```

Advanced Problems

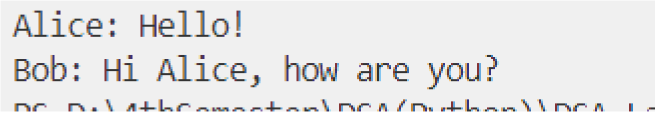
1. Facebook Messenger Chat Queue:

Implement a queue system where chat messages are stored and displayed in order.

Code:

```
class ChatQueue:
    def __init__(self):
        self.queue = []

    def send_message(self, sender, message):
        """Add a new message to the queue."""
        self.queue.insert(0, (sender, message))

    def receive_messages(self):
        """Retrieve messages in FIFO order."""
        while self.queue:
            sender, message = self.queue.pop()
            print(f"{sender}: {message}")

# Example Usage
chat = ChatQueue()
chat.send_message("Alice", "Hello!")
chat.send_message("Bob", "Hi Alice, how are you?")
chat.receive_messages()
```

Output:



```
Alice: Hello!
Bob: Hi Alice, how are you?
PS D:\4thSemester\DSA(Python)\DSA
```

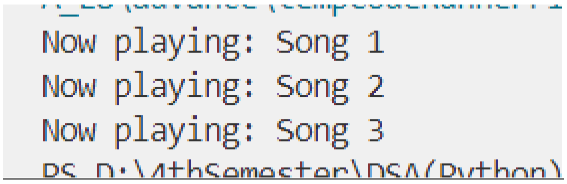
2. Spotify Playlist Queue:

Implement a circular queue to cycle through songs in a playlist.

Code:

```
class CircularQueue:
    def __init__(self, size):
        self.queue = [None] * size
        self.max_size = size
        self.front = self.rear = -1

    def enqueue(self, song):
        """Add a song to the queue."""
        if (self.rear + 1) % self.max_size == self.front:
            print("Playlist is full!")
            return
        if self.front == -1:
            self.front = 0
        self.rear = (self.rear + 1) % self.max_size
        self.queue[self.rear] = song

    def dequeue(self):
        """Remove a song from the queue."""
        if self.front == -1:
            print("No songs in playlist!")
            return None
        song = self.queue[self.front]
        if self.front == self.rear:
            self.front = self.rear = -1
        else:
            self.front = (self.front + 1) % self.max_size
        return song

    def play_all(self):
        """Play all songs in circular order."""
        while self.front != -1:
            print("Now playing:", self.dequeue())

# Example Usage
playlist = CircularQueue(5)
playlist.enqueue("Song 1")
playlist.enqueue("Song 2")
playlist.enqueue("Song 3")
playlist.play_all()
```

Output:



```
Now playing: Song 1
Now playing: Song 2
Now playing: Song 3
PS D:\4thSemester\DSA(Python)\
```

3. Operating System Process Queue:

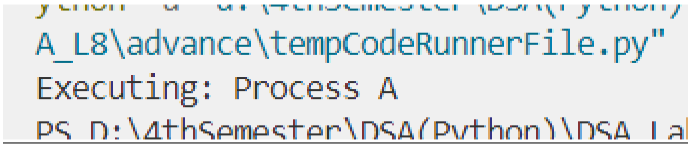
Simulate how an operating system manages

processes using a priority queue.

Code:

```
import heapq

class ProcessQueue:
    def __init__(self):
        self.queue = []

    def add_process(self, priority, process_name):
        """Add a process with a priority (lower number = higher
priority)."""
        heapq.heappush(self.queue, (priority, process_name))

    def execute_process(self):
        """Execute the highest priority process."""
        if not self.queue:
            return "No processes to execute."
        return f"Executing: {heapq.heappop(self.queue)[1]}"

# Example Usage
pq = ProcessQueue()
pq.add_process(3, "Process C")
pq.add_process(1, "Process A")  # Highest priority
pq.add_process(2, "Process B")
print(pq.execute_process())  # Executing: Process A
```

Output:



```
ython -u d:\4thSemester\DSA(Python)
A_L8\advance\tempCodeRunnerFile.py"
Executing: Process A
PS D:\4thSemester\DSA(Python)\DSA La
```

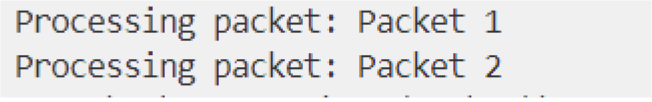
4. Network Packet Handling Queue:

Implement a queue to process packets sent over a network.

Code:

```
class PacketQueue:
    def __init__(self):
        self.queue = []

    def send_packet(self, packet):
        """Add a packet to the queue."""
        self.queue.insert(0, packet)

    def process_packet(self):
        """Process packets in FIFO order."""
        if self.queue:
            return f"Processing packet: {self.queue.pop()}"
        return "No packets to process."

# Example Usage
network = PacketQueue()
network.send_packet("Packet 1")
network.send_packet("Packet 2")
print(network.process_packet())  # Processing packet: Packet 1
print(network.process_packet())  # Processing packet: Packet 2
```

Output:



```
Processing packet: Packet 1
Processing packet: Packet 2
```

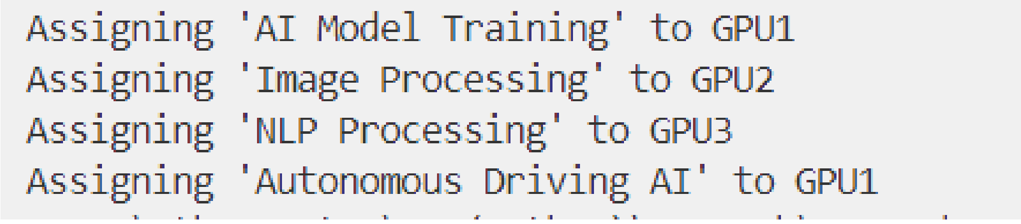
5. AI Task Processing Queue:

Implement a queue that assigns AI processing tasks to available GPU resources.

Code:

```
class AITaskQueue:
    def __init__(self, gpus):
        self.tasks = []  # Task queue
        self.gpus = gpus  # List of GPUs

    def add_task(self, task):
        """Add an AI task to the queue."""
        self.tasks.append(task)

    def process_tasks(self):
        """Assign tasks to GPUs in round-robin order."""
        if not self.tasks:
            print("No AI tasks in the queue.")
            return

        gpu_index = 0  # Track which GPU to assign next
        while self.tasks:
            task = self.tasks.pop(0)  # Get the first task (FIFO
order)
            gpu = self.gpus[gpu_index]  # Assign to GPU in round-
robin
            print(f"Assigning '{task}' to {gpu}")

            gpu_index = (gpu_index + 1) % len(self.gpus)  # Move to
next GPU

# Example Usage
ai_queue = AITaskQueue(["GPU1", "GPU2", "GPU3"])
ai_queue.add_task("AI Model Training")
ai_queue.add_task("Image Processing")
ai_queue.add_task("NLP Processing")
ai_queue.add_task("Autonomous Driving AI")
ai_queue.process_tasks()
```

Output:



```
Assigning 'AI Model Training' to GPU1
Assigning 'Image Processing' to GPU2
Assigning 'NLP Processing' to GPU3
Assigning 'Autonomous Driving AI' to GPU1
```