Submitted To:

Engineer Sheharyar Khan

Name: Iqra Fatima

Reg. Number: 23-CP-62

Semester: 4th

Department: CPED

Data Structures and Algorithms

(DSA)  
Lab Report 8

# 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:**

A screen shot of a computer code

AI-generated content may be incorrect.

## 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:**

****

## 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:**

****

# 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")**

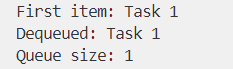
**queue.enqueue("Task 2")**

**print("First item:", queue.peek())  # Task 1**

**print("Dequeued:", queue.dequeue())  # Task 1**

**print("Queue size:", queue.size())  # 1**

**Output:**

****

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 close up of a text

AI-generated content may be incorrect.**

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:**

**A close up of a text

AI-generated content may be incorrect.**

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**

**Output:**

**A close up of numbers

AI-generated content may be incorrect.**

5. Print Jobs Simulation:

**Simulate print job handling using a queue.**

**Code:**

**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**

**Output:**

## 

## 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:**

****

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:**

A screenshot of a computer

AI-generated content may be incorrect.

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:**

**A close up of a computer screen

AI-generated content may be incorrect.**

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:**

**A black and white text

AI-generated content may be incorrect.**

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:**

****

## 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:**

A close up of a text

AI-generated content may be incorrect.

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:**

**A screen shot of a computer screen

AI-generated content may be incorrect.**

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:**

**A screenshot of a computer

AI-generated content may be incorrect.**

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:**



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")**

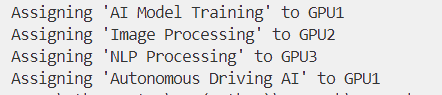
**ai\_queue.add\_task("Image Processing")**

**ai\_queue.add\_task("NLP Processing")**

**ai\_queue.add\_task("Autonomous Driving AI")**

**ai\_queue.process\_tasks()**

**Output:**

****