

**ABBOTTABAD UNIVERSITY OF SCIENCE AND**

**TECHNOLOGY**

SUBMITTED BY: WAJDAN ALI

SUBMITTED TO: MR JAMAL ABDUL AHAD

SECTION: C

ROLL NO: 12408

ASSIGNMENT: ASSIGNMENT 2

SUBJECT: DATA STRUCTURES AND ALGORITHMS

**QUS#1:** Design a python program that simulates a web server handling incoming reqs using queue. Model df types of reqs with varying processing times and simulate their processing order.

**ANSWER:**

import queue

import time

import random

# Define a simple request class

class Request:

def \_init\_(self, request\_type, processing\_time):

self.type = request\_type

self.processing\_time = processing\_time

# Function to simulate web server

def simulate\_web\_server(requests):

q = queue.Queue()

for request in requests:

q.put(request)

while not q.empty():

current\_request = q.get()

print(f"Processing {current\_request.type} request...")

time.sleep(current\_request.processing\_time)

print(f"{current\_request.type} request processed.")

# Example usage

requests = [

Request("GET", 2),

Request("POST", 4),

Request("PUT", 3),

# Add more requests as needed

]

simulate\_web\_server(requests)

**QUS#2:** In what scenarios would you choose a linked list implementation over an array implementation for a queue and vice versa.

**ANSWER:**

Use a linked list when the size of the queue is dynamic, and you want to efficiently handle insertions and deletions.

Use an array when the queue size is fixed or known in advance, and you need constant-time random access.

**QUS#3:** Discuss the time complexity of enqueue and dequeue operations in a basic queue. How cann you optimize these operations for specific use case?

**ANSWER:**

Enqueue: O(1)

Dequeue: O(1)

Optimizations:

For a dynamic queue, use a linked list to avoid resizing overhead.

Use a circular buffer for an array implementation to efficiently manage the queue size.

**QUS#4:** How can you use stacks to implement a queue. Provide a step by step explanation of the enqueue and dequeue operations in this scenario.

**ANSWER:**

class TwoStackQueue:

def \_init\_(self):

self.stack1 = []

self.stack2 = []

def enqueue(self, item):

self.stack1.append(item)

def dequeue(self):

if not self.stack2:

if not self.stack1:

return None # Queue is empty

while self.stack1:

self.stack2.append(self.stack1.pop())

return self.stack2.pop()

# Example usage

queue\_example = TwoStackQueue()

queue\_example.enqueue(1)

queue\_example.enqueue(2)

queue\_example.enqueue(3)

print(queue\_example.dequeue()) # Output: 1

print(queue\_example.dequeue()) # Output: 2