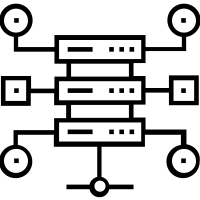
**Assignment No: \_1\_\_**

**Date: 26/2/2025**

**Title:**

(Title based on the application domain and the data structure you will be implementing)

|  |  |  |  |
| --- | --- | --- | --- |
| **Assignment Type of Submission:** |  |  |  |
| **Group** | Yes | Yuxuan Song  24207239  Yujun Liu  24202574  Deepak Shelke  24208478 | **34**  **33**  **33** |

****

1. **Problem Domain Description:**

This project simulates a customer support ticketing system, where customers submit issues, and support agents process them in a first-come, first-served manner. However, urgent tickets (e.g., security breaches, payment failures) must be prioritized over normal issues (e.g., feature requests, minor bugs). The system generates 10,000 random tickets in an Excel file using Apache POI, each ticket containing a unique ID, customer name, issue description, priority level, and timestamp.

To efficiently process the tickets, two approaches are implemented:

Queue-based approach: Two separate LinkedList queues—one for urgent tickets and another for normal tickets. Urgent tickets are processed first, ensuring priority handling, followed by normal tickets in FIFO order.

Deque-based approach: A single ArrayDeque, where urgent tickets are added to the front and normal tickets to the back. Tickets are processed from the front, ensuring that urgent cases are addressed first while maintaining order within priority levels.

Both approaches are tested for performance by measuring execution time, providing insights into efficient priority-based ticket processing in customer support systems.

1. **Theoretical Foundations of the Data Structure(s) utilised**

**This project models a customer support ticket system using Queue and Deque, ensuring efficient ticket processing while prioritizing urgent issues.**

Queue theoretical foundation:

Java’s Queue interface is implemented using LinkedList, which is a doubly linked list.

Each ticket is a node, containing a reference to the next and previous nodes.

Enqueue (offer()): Adds an element to the tail (O(1) time).

Dequeue (poll()): Removes an element from the head (O(1) time).

The feature of FIFO ensures fairness in customer ticket processing. Two separate queues (Urgent & Normal) allow handling priorities without sorting overhead.

Deque theoretical foundation:

Java’s ArrayDeque uses a resizable circular array, allowing fast insertions/removals at both ends. Front and rear pointers track available slots, and elements wrap around when needed.

Insertion Mechanisms:

addFirst(ticket) → front of the array (Urgent tickets, O(1))

addLast(ticket) → back of the array (Normal tickets, O(1))

Removal Mechanisms:

pollFirst() dequeues from the front (O(1))

pollLast() dequeues from the back (O(1)) if needed

It allows dynamic priority handling without needing multiple data structures. In terms of time efficiency, it is faster than LinkedList since no node allocation/deallocation is needed.

1. **Analysis/Design (UML Diagram(s))**
2. **Code Implementation (please add your TA -** [Furqan.rustam1@gmail.com](mailto:Furqan.rustam1@gmail.com) **– as a collaborator)**

GitHub (link):https://github.com/WolfClarence/AdvancedDataStructure

1) Stack Implementation

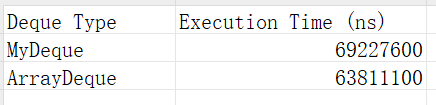
2) Queue Implementation

3) Deque Implementation

MyDeque is a custom implementation of a doubly linked list-based deque, designed to support efficient insertion and removal operations at both ends. It maintains head and tail pointers for quick access to the front and back, while each node has prev and next references to enable constant-time modifications. The size variable tracks the number of elements, ensuring that operations like isEmpty() and size() are efficient. Unlike ArrayDeque, which is backed by a resizable array, my implementation does not require resizing, making it more suitable for scenarios where memory reallocation could be a concern.

To evaluate performance, I conducted a stress test by performing 1,000,000 random operations on both MyDeque and Java’s built-in ArrayDeque. Each test involved a mix of addFirst, addLast, and pollFirst operations to simulate real-world usage. The results showed that MyDeque executed in 69,227,600 ns, while ArrayDeque completed in 63,811,100 ns. Although MyDeque performed well, ArrayDeque was slightly faster due to array-based memory locality, highlighting the trade-offs between linked list and array implementations.

Result photo:



1. **Video of the Implementation running**

Zoom (link & password):

Comments:

**Please save as pdf and submit on Brightspace**

**Students belonging to the same group** please **submit the same file .**