DS2030 Data Structures and Algorithms for Data Science
Mid-Semester Lab Exam

September 30th, 2025

Lab Instructions

- Create a folder named **"DS2030_<RollNo.>"** (all letters in capital) in **"home"** directory. Eg- **DS2030_142402022**
- Code template files will be provided to you.
- Make sure the **folder, files, classes, functions and attributes** are named as instructed in the lab sheet.
- We will not be able to evaluate your work if the folder is not properly named or is not located in the home directory.
- Make sure to save your progress before leaving the lab.
- Do not shut down the system after completing the lab.
- You are not allowed to discuss or share code with your classmates.
- The exam will have a duration of **2 hours and 50 minutes** (2:00 - 4:50).

Q1. Emergency Task Scheduler (15 marks)

A hospital maintains an **emergency task queue** to manage patient cases. Each task is represented as a tuple (*task_name*, *priority*, *department*), where:

- `task_name` – Name of the task or patient case.
- `priority` – A number indicating urgency (smaller = higher priority). Each priority is unique.
- `department` – The hospital department responsible (e.g., "Cardiology", "Neurology").

The queue follows first-in-first-out (FIFO) order, but tasks may need to be rearranged based on urgency and department rules.

Queue Implementation

We implement a queue using a list of tuples. The queue supports:

- `enqueue(task)` – Add a task at the rear.
- `dequeue()` – Remove and return the front task.
- `is_empty()` – Check if the queue is empty.
- `peek()` – Return the front task without removing it.

Required Functionalities (Recursive Only)

Using only queue operations and recursion, implement:

1. **Print tasks in reverse order** (5 marks) Display tasks from latest to earliest without changing the queue order.
2. **Find highest priority task for a specific department** (5 marks) Recursively find the task with the smallest priority number within a given department. If no task exists for that department, return `None`.
3. **Reorder queue by department** (5 marks) Recursively move all tasks of a given department to the front of the queue while preserving the relative order of all other tasks. For example, moving all "Cardiology" tasks to the front should keep them in the same order relative to each other, and all other tasks should remain in their original order after them.

Constraints

- No loops (`for`/`while`) are allowed.
- No additional data structures (lists, stacks, arrays) are permitted beyond the queue itself.
- All operations must restore the queue order where appropriate and use recursion only.

Expected Output

```
Tasks in reverse order:
Task5 (3, Neurology)
Task4 (4, Orthopedics)
Task3 (1, Cardiology)
Task2 (2, Neurology)
Task1 (5, Cardiology)

Highest priority task in Cardiology:
Task3 (1, Cardiology)

Queue after moving Cardiology tasks to front:
Task1 (5, Cardiology)
Task3 (1, Cardiology)
Task5 (3, Neurology)
Task4 (4, Orthopedics)
Task2 (2, Neurology)
```

Q2. BST and Doubly Linked List (10 marks)

You are given a set of integer values to construct a Binary Search Tree (BST). Your task is to perform the following operations:

Tasks

1. **Construct the BST**(5 marks)
    - Insert the given values one by one into a BST following the standard BST property:
        - Left subtree values < parent node value
        - Right subtree values > parent node value
2. **Convert the BST to a Doubly Linked List (DLL)**(3 marks)
    - Convert the BST into a sorted DLL using in-order traversal.
    - Reuse the BST node pointers:
        - `left` pointer acts as previous in DLL.
        - `right` pointer acts as next in DLL.
3. **Print the DLL**
    - Print all elements of the DLL from head to tail in a single line.
4. **Find a pair with a given sum in the DLL**(2 marks)
    - Given a target sum, determine if two nodes in the DLL sum up to the target.
    - Print all the posibble pairs; otherwise, indicate that no such pair exists.

Sample Output

```
Binary Search Tree:
    10
  6    14
 4  8  12  16

Doubly Linked List:
4 6 8 10 12 14 16

Pair found with sum 20: (4, 16)
Pair found with sum 20: (6, 14)
Pair found with sum 20: (8, 12)
```

Q3. Max-Heap Operations (5 marks)

You are given a set of integers. Your task is to perform the following operations using a Max-Heap:

Tasks

1. **Build a Max-Heap** Insert the given integers into a Max-Heap.
2. **Extract Maximum** Remove and print the maximum element (the root of the heap).

Sample Output

```
Heap after insertions (level order): [40, 30, 20, 5, 10, 15]
Extracted Maximum: 40
Heap after one extraction (level order): [30, 15, 20, 5, 10]
```