General Instructions:

Carefully read the following instructions before attempting the paper.

- The Exam paper consists of 3 questions on 2 printed sides of 1 page.
- In case of any ambiguity, you may make assumptions, but your assumption must not contradict any statement in the question paper. Also mention your assumptions.
- **DON'T** share your program, if your code is matched to any member of your class, both will get **straight F** in the course without asking who shared or who magically copied.
- All your files must be named by your roll number along with question number e.g. **K23-XXXX_Q1.cpp.**

Question # 01 (LLO #: 1) ***[20 Points, 15 Weightage]***

FAST - NUCES is organizing a golf tournament for its faculty, where players are ranked based on their performance scores, with lower scores indicating better performance. The tournament organizers utilize a dynamic ranking system to efficiently manage player standings and adjust rankings as new scores are recorded. You are tasked with developing the ranking system using heap data structures. Initially, the following scores are recorded: 45, 20, 35, 10, 50, 30, and 25. To manage the rankings, you need to organize the scores to ensure that the player with the lowest score (best performance) is always at the top. As new scores are added, the data should be adjusted to maintain the integrity of the ranking system. After all initial scores are inserted, extract the top three performers, ensuring that the heap structure is maintained. Afterward, transfer the remaining players' scores to focus on the players with the highest scores (worst performance), allowing for easy extraction of the worst performers.

Example Output:

Top 3 Performers (Best Scores): 10 20 25
Worst Performers (Highest Scores): 50 45 35 30

National University of Computer and Emerging Sciences

Question # 02 (LLO #: 2) ***[40 Points, 20 Weightage]***

The Galactic Pizza Delivery Service maintains a database of delivery drivers, including their average delivery time, vehicle speed, and name. The GPDS wants to develop a program that allows for efficient searching of drivers based on their average delivery time or vehicle speed using a Balanced Binary Search Tree (BST).
The program should be designed to store driver records in the BST, where each record includes the driver's average delivery time, vehicle speed, and name. The program must support several key features: the ability to add new drivers into the tree with their respective attributes, the ability to remove driver records based on delivery time or vehicle speed, and a search functionality to find drivers who fall within a given range of delivery time or vehicle speed (both cases to be handled). The program should return a list of drivers whose delivery time or vehicle speed matches the specified search criteria, enabling GPDS officials to easily analyze and filter drivers for quick delivery optimization, peak hour management, and country-wide pizza delivery campaigns.

Question # 03 (LLO #: 3) ***[40 Points, 15 Weightage]***

A publishing company is looking to implement a system for analyzing the frequency of words in novels to help identify the most frequently used words. The company handles a vast amount of textual data, and with the increasing number of novels and manuscripts, manually counting and analyzing word frequencies is becoming inefficient and time-consuming. To address this, the company needs an automated system that can efficiently process large volumes of text and provide accurate frequency counts.
The system should use a hash table to store the words and their corresponding frequencies. Each word in the novel will serve as a key, and its frequency (how many times it appears) will serve as the value. The system should include a mechanism to resize when more items are added. This ensures that the hash table maintains optimal performance.
Once the hash table has recorded the frequency of all words in the novel, the system should then proceed to sort the words based on their frequency in descending order. Words with higher frequencies should appear first in the sorted list, providing a clear view of the most commonly used terms. If two or more words share the same frequency, the system should then sort them in alphabetical order in descending order ('z' appears first & 'a' appears last). This ensures that the output is both meaningful and easy to interpret.

Note: In case of collisions, resolve them using the double hashing method only. For sorting, you can only use merge sort.