COMP 202: Data Structures and Algorithms
Mini Projects

Students are required to carry out one mini project for the course COMP 202. The project will carry 5 (five) marks for internal grading. Students must choose one project from the provided list of projects (see Section 3), and work in a group of at most 3 students. At the end of the project, each group is required to provide a demo of the project, and take a viva-voce examination.

1 Important Dates

All groups must respect the following deadlines:

Project selection deadline Dec 6, 2019

Project submission deadline Jan 15, 2020

Demo and viva will be between Jan 19, 2020 and Jan 24, 2020.

2 Project Guidelines

1. No more than 2 groups may work on the same project. To avoid conflicts in project selection, all groups are required to choose 3 projects from Section 3 and email the chosen projects along with their preferences to rajani.chulyadyo@ku.edu.np. If there are projects selected by more than 2 groups, projects will be assigned to the groups based on their preferences.

2. Each group is required to submit the following by the date mentioned in Section 1:

   (a) Well-documented code. To be submitted through Gitlab. Naming convention for the Git repository is as follows:
   $CE2018\_MP\_\langle RollNoStudent1\rangle\_\langle RollNoStudent2\rangle\_\langle RollNoStudent3\rangle$

   (b) Report containing a brief explanation of the submitted implementation, relevant screenshots, and answers to the questions, if any, posed in the task.

3 Projects

For all projects below, you are required to submit your implementation along with a test program to verify that your implementation works correctly.

1. Write a program to convert prefix expressions to postfix. What is the time complexity of your program?

   You are supposed to use the stack implementation done in Lab 1.

2. Write a program to convert infix expressions to postfix. What is the time complexity of your program?

   You are supposed to use the stack implementation done in Lab 1.

3. Write a program to evaluate postfix expressions. What is the time complexity of your program?

   You are supposed to use the stack implementation done in Lab 1.

4. Write a program to simulate a queue of customers in a bank. Customers must be served one at a time and the customer that has been waiting the longest must be served first.

5. Implement a stack using a circularly linked list with the following operations:

   - *push* – Inserts an element into the stack.
   - *pop* – Displays and deletes an element at the top of the stack.
   - *top* – Displays the element at the top of the stack.
   - *isEmpty* - Returns true if the stack is empty and false otherwise.
   - *isFull* - Returns true if the stack is full and false otherwise.

   What are the time complexities of these operations in your implementation?

   You are supposed to use the linked list implementation done in Lab 2.

6. Implement a queue using a circularly linked list with the following operations:

   - *enqueue* – Inserts an element at the end of the queue
   - *dequeue* – Displays and deletes an element from the front of the queue
   - *rear* – Displays the last element of the queue
   - *front* – Displays the first element of the queue
   - *isEmpty* – Returns true if the queue is empty and false otherwise
   - *isFull* – Returns true if the queue is full and false otherwise

What are the time complexities of these operations in your implementation?

You are supposed to use the linked list implementation done in Lab 2.

7. Implement the following tree traversal techniques:

   - Inorder traversal
   - Preorder traversal
   - Postorder traversal

   You are allowed to use the tree implementation done in Lab 3. Also, discuss on the time complexities of your implementation.

8. Implement a binary search tree (BST). Your implementation must support the following operations:

   - *searchBST(root, targetKey)* : Search a BST for the requested key
   - *addBST(root, newNode)* : Add a node to a BST
   - *deleteBST(root, keyToDelete)* : Delete a node from a BST

   What are the time complexities of these operations in your implementation?

9. Implement heap data structure and use it to implement an ascending priority queue. Discuss on the time complexity of your implementation.

10. Implement graph data structure. Use your implementation to create some graphs, and then perform depth-first search (DFS) and breadth-first search (BFS) on them.

    Your report must contain visual representation of the graphs used in your program along with the results of DFS and BFS traversals.

11. Implement Kruskal's algorithm for finding a minimum spanning tree. Apply it on at least 3 graphs. Discuss on the time complexity of your implementation.

    Your report must contain visual representation of the graphs used in your program along with the results of Kruskal's algorithm.

12. Implement Prim's algorithm for finding a minimum spanning tree. Apply it on at least 3 graphs. Discuss on the time complexity of your implementation.

    Your report must contain visual representation of the graphs used in your program along with the results of Prim's algorithm.

13. Implement graph data structure and use it to create a network of friends. Use graph traversal methods to find friends of friends in this network.

14. Implement linear search and binary search algorithms. Compare the performance of these algorithms in your implementation by applying these algorithms on at least 10 different lists of varying length. Also, discuss on the time complexity of these algorithms.

15. Implement insertion sort and bubble sort algorithms. Plot a graph depicting the worst case running times for these algorithms.

16. Implement selection sort and bubble sort algorithms. Plot a graph depicting the worst case running times for these algorithms.

17. Implement selection sort and merge sort algorithms. Plot a graph depicting the worst case running times for these algorithms.

18. Implement quick sort and merge sort algorithms. Plot a graph depicting the worst case running times for these algorithms.

19. Implement merge sort and heap sort algorithms. Plot a graph depicting the worst case running times for these algorithms.

20. Implement merge sort and bubble sort algorithms. Plot a graph depicting the worst case running times for these algorithms.