National University of Computer and Emerging Sciences

Lahore Campus

Data Structures (CS2001)

Final Exam

Date: August 7, 2024

Course Instructor

Ms. Hira Butt

Total Time (Hrs): 3

Total Marks: 50

Total Questions: 5

[redacted] Roll No

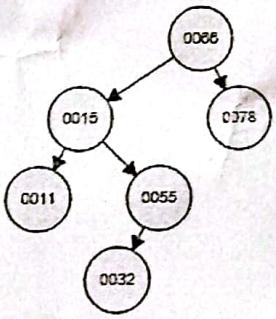
[redacted] Section

[redacted] Student Signature

Do not write below this line

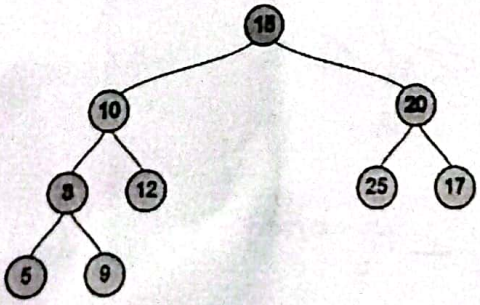
- **Attempt all the questions.**
- **In case of confusion or ambiguity, make a reasonable assumption. Questions are not allowed.**
- **You must attempt all questions on answer sheet NEATLY, in case of any overwriting/ambiguity from your side, marks will be deducted and no explanation will be entertained.**
- **Bonus 1 mark(s) for attempting your all questions in order.**

Question 1 (a): Is this an AVL tree? If yes, redraw the same tree, and if not, redraw the tree after rebalancing. You must show all working with the names of imbalance cases, nodes, and rotations performed. [3+3]



(b) *Now, delete 55 from your* **resultant** (re-drawn) AVL Tree and show all of your working. Don't *forget to highlight your final answer.*

Question 2: Build min heap from this given tree using BuildHeap method and show all of your working [5]



Question 3: Dry run and Guess the output for the following code snippet. Imagine you have already implemented display and insert functions. *[10]*

```
#include <iostream>
#include <queue>
using namespace std;

class Node {
   friend class LinkedList;
private:
   int data;
   Node* next;
   Node(int d) {
      this->data = d;
      this->next = nullptr;
   }
};

class LinkedList {
private:
   Node* head;
public:
  LinkedList() {
    head = nullptr;
  }

  void funcGuess() {
    if (!head) return;

    queue<int> nodeQueue;
    Node* curr = head;

    while (curr != nullptr) {
      nodeQueue.push(curr->data);
      curr = curr->next;
    }

    Node* newHead = nullptr;
    Node* tail = nullptr;

    while (!nodeQueue.empty()) {
      if (!nodeQueue.empty()) {
        int frontValue = nodeQueue.front();
        nodeQueue.pop();
        Node* frontNode = new Node(frontValue);
        if (newHead == nullptr) {
          newHead = frontNode;
          tail = frontNode;
```

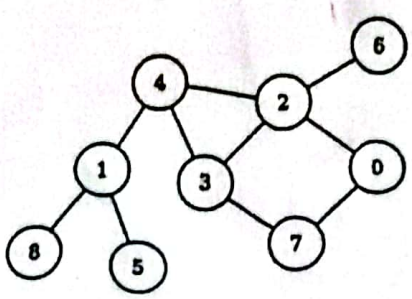
```
    } else {
        tail->next = frontNode;
        tail = tail->next;
    }
}

if (!nodeQueue.empty()) {
    queue<int> tempQueue;
    while (nodeQueue.size() > 1) {
        tempQueue.push(nodeQueue.front());
        nodeQueue.pop();
    }
    int backValue = nodeQueue.front();
    nodeQueue.pop();
    Node* backNode = new Node(backValue);
    tail->next = backNode;
    tail = tail->next;

    while (!tempQueue.empty()) {
        nodeQueue.push(tempQueue.front());
        tempQueue.pop();
    }
}
if (tail) {
    tail->next = nullptr;
}
head = newHead;
}
};
int main() {
    LinkedList l1;
    l1.insertAtEnd(1);
    l1.insertAtEnd(2);
    l1.insertAtEnd(3);
    l1.insertAtEnd(4);
    l1.insertAtEnd(5);
    l1.insertAtEnd(6);
    l1.insertAtEnd(7);
    l1.insertAtEnd(8);

    cout << "Original list:" << endl;
    l1.display();
    l1.funcGuess();
    cout << "Updated list:" << endl;
    l1.display();
}
```

Question 4 (a): For the Given graph [10+3]

- Apply Depth first search algorithm to find a path from node 5 to node 7.
- Apply Breadth first search algorithm to find a path from node 5 to node 7.

Note: While pushing nodes into stack or enqueing nodes into queue, push or enqueue minimum element first. Starting node is 5

(b) Adjacency matrix or adjacency list, which will be more efficient to represent the given graph, and why? Explain your answer.



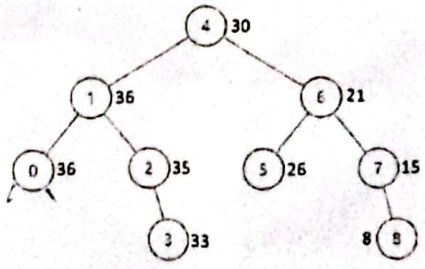
Question 5: *Suppose a file contains* the following characters along with their frequencies.
J : 22, *B* : 5, *D* : 11, C : 19, G : 2, E : 11, H : 25, F: 5

[6]

How many bytes will be required to store this file if Huffman encoding scheme is used. Show your complete working. (table and tree)

Question 6: Transform the given BST into a Greater Sum Tree where every key of the original BST is changed/replaced with the "original key plus the sum of all keys greater than the original key in BST". In short, given the root of a binary search tree, each node in the BST needs to be replaced with the sum of all greater nodes in the BST. [10]

Example: Here is the original BST, we have to replace its value with the sum of its all greater nodes including itself (representing in blue). Also, return the updated tree as a result.



Function signature should be this:

TreeNode* bstToGst(TreeNode* root);

//Suppose you have already implemented the rest of BST class functions. Just by adding this function to your class, it should work fine. Also, you can use any helper function inside it, if you want (Hint) to ease your task.

Good Luck ☺