National University of Computer and Emerging Sciences, Lahore Campus

Course:	**Data Structures**	**Course Code:**	**CS 201**
Program:	**BS(Computer Science)**	**Semester:**	**Fall 2018**
Duration:	**180 Minutes**	**Total Marks:**	**80**
Paper Date:	**21-Dec-2018**	**Page(s):**	**8**
Section:	**ALL**	**Section:**	
Exam:	**Final Exam**	**Roll No:**	

Instruction/Notes: Answer in the space provided
You can ask for rough sheets but **they will not be graded or marked**
In case of confusion or ambiguity make a reasonable assumption.
Questions are not allowed

Good luck!

Question 1: **(Marks: 5 * 7)**

a. Suppose, you are going to design an application for stock exchange, where prices of different shares (stocks) will be stored and processed. Select the best suitable and most efficient data structure for following requirements. Also, provide time complexity in big (Oh) form, for all required operations.

1. User can buy a share, which price is closest to the price provided by user.
2. User can view a list of all the shares, which prices lie in provided range.
3. User can search and sale five of those shares, which price is less than or equal to the given price.

AVL maintained on attribute price

1. Search for a price greater or equal to the given price and when you found one .. recur back and find a price on the back path that is just less then the given price …O(lgn)
2. Find price in O (lg n) and user can view them in O(s lgn) where s is the no of shares between the given range.
3. Similar to first

1. Closest means equal, smaller or larger value, which are floor and ceil values actually, so this operation can be performed in worst case $O(\log_2 n)$
2. Perform in order traversal of tree and print all data in provided range worst case O(n)
3. Find floor of given value which can be searched in $O(\log_2 n)$, this operation will be performed 5 times, so worst case is $O(\log_2 n)$

b. WhatsApp is a mobile app that allow its users to send and receive text, audio and video messages. The WhatsApp team wish to find whether a messages reach back to its sender or not?
For example, a person P sends a message to his friends and they forward the message to their friends and so on. Given the users of WhatsApp and the information of all the messages sent by its users to other users, we need to determine that a message is sent back to its user or not for a particular user. What data structure is most suitable to represent this data and why? How would you solve this problem with your suggested data structure? Briefly explain your idea.

Graph …. Where node is user and edge represent a msg.
Detect cycles in the users -message graph using DFS or BFS.

c. Suppose the department of computer science is offering n sections of the course Linear Algebra. The timetable has already been setup by the computer science department. The department of mathematics nothat 2 teachers are sufficient to teach all n sections given the time table of all Linear algebra classes? You can assume that a teacher can teach multiple sections of the course as long as their class timings does not overlap in the time table. What data structure is most suitable to represent this data and why? How would you solve this problem with your suggested data structure? Explain your idea (not code) in 3-4 lines

Graphs … node represent a course and an edge connect two nodes if their timings clash

Check if the graph is bipartite or not …. If its bipartite then 2 teachers are enough ….

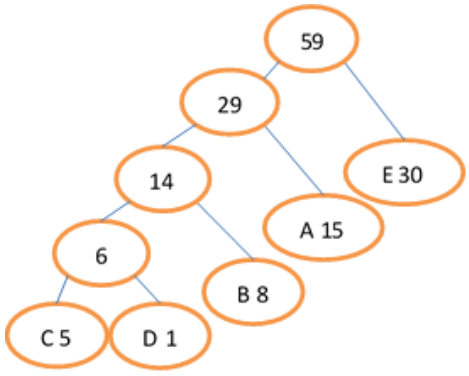
d. Suppose you are implementing a web browser that has a collection of malicious websites and whenever a user try to visit a web page, your browser must check whether that website is malicious or not. What data structure is most suitable to represent this data and why? How would you solve this problem with your suggested data structure? Briefly explain your idea.

Hash table as it is very efficient in searching
Hash all malicious website addresses and when a new website is to be checked we simply try to locate it in hash table if found then its malicious.

e. For which character frequencies of encoding data, Huffman algorithm always generates a skewed binary tree.

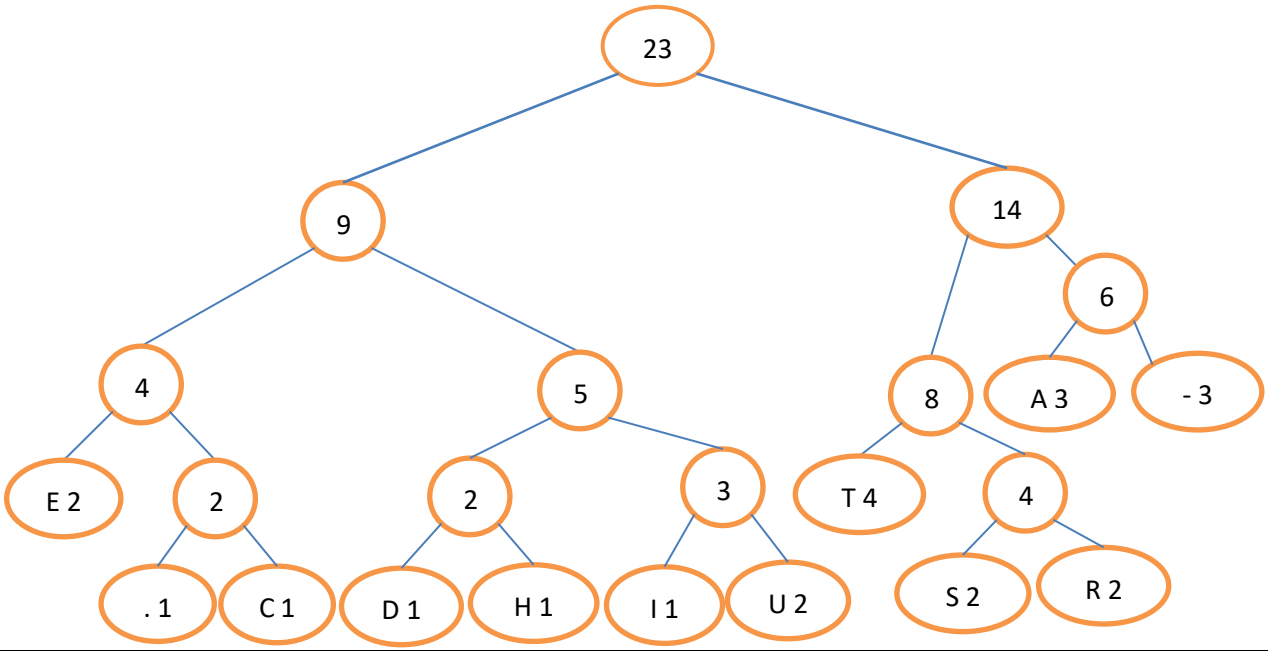
Fibonacci nos, or Frequency sum of two characters is smaller than the next frequency of character picked from queue. For Example:

E	A	B	C	D
30	15	8	5	1



Roll Number:_____ Section:_____

f. Decode the string of binary data given below using following tree generated by Huffman Algorithm.



0110 111 1010 100 1011 0111 0011 100 0111 1011 000 1010 111 0100 110 100 110 111 0101 110 100 000 0010
Decoded Message: **l1-3s2T4R2U2C1T4U2R2E2S2-3D1A3T4A3-3H1A3T4E2.1** **I-STRUCTURES-DATA-HATE.**

g. Suppose we want to reverse first k elements in a FIFO queue. For example if the input queue is: {10, 20, 30, 40, 50, 60} and k=4. Then output queue must be: {40, 30, 20, 10, 50, 60}. The only functions available for the Queue are:

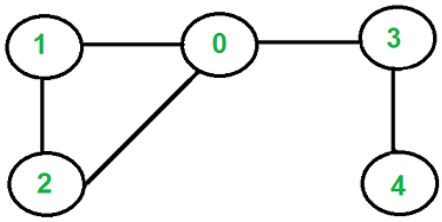
Enque() – inserts a data item at the end of the queue
Dequeu() – Removes an element from the start of the queue and return the removed element
Isempyt() – Return true if queue is empty and false otherwise
getSize() – returns the number of elements in the queue
How would you reverse the first k elements of the queue using the above functions only? Explain your idea in 3-4 lines
Hint: You may use some other standard data structure to solve this problem

First dequeu k elements from the queue and push in the stack.
Then for all k element, pop from the stack and enque in the queue
Lastly, for first n-k elements of the queue deque one element at a time and enque at the end of the list.

Roll Number:_____ Section:_____

Question 2: **(Marks: 12+3)**

The diameter of a graph is the maximum of shortest paths between any pair of vertices (u, v). The diameter of an unweighted disconnected graph is infinity. Consider the graph on left. The possible vertex pair and their distance (shortest paths) are Dist(0,1) = 1, Dist(0,2) = 1, Dist(0,3)=1, Dist(0,4)=2, Dist(1,2)=1, Dist(1,3)=2, Dist(1,4)=3, Dist(2,3)=2, Dist(2,4)=3, Dist(3,4)=1. Hence, the diameter of following graph is three (3).



Write a C++ function ***ComputeDiameter()*** that takes an unweighted, undirected graph G as parameter and returns the diameter of G.

```
#include <iostream>
#include <queue>
using namespace std;
const int V = 5;
int findDiameter(int G[][V], int src)
{
	bool visited[V]; //save visited nodes information
	int dist[V]; //save distance information
	int diameter = 0;
	queue <int> q;

	for (int i = 0; i < V; i++){
		dist[i] = -1;
		visited[i] = false;
	}
	dist[src] = 0; //dist of src to itself will be zero
	visited[src] = true; //visited true
	q.push(src);
	diameter = dist[src];
	while (!q.empty())
	{
		int u = q.front(); q.pop();
		// Find all non-visited adjacent vertices and compute distance to them.
		for (int v = 0; v < V; ++v){
			if ( G[u][v] && !visited[v]){
				visited[v] = true;
				dist[v] = 1 + dist[u];
				if (dist[v]> diameter) //update diameter
					diameter = dist[v];
				q.push(v);
			}
		}
	}
	return diameter;
}
```

```
//find diameter to all nodes.
int computeDiameter(int G[][V])
{
	int diameter = -1;
	for (int i = 0; i < V; i++){
		int d = findDiameter(G, i);
		if (d > diameter)
			diameter = d;
	}
	return diameter;
}
```

```
void main() {
	int G[][V] = {
	{ 0, 1, 1, 1, 0 },
	{ 1, 0, 1, 0, 0 },
	{ 1, 1, 0, 0, 0 },
	{ 1, 0, 0, 0, 1 },
	{ 0, 0, 0, 1, 0 }, };
	cout << "Diameter of Graph is :  "
		<< computeDiameter(G) <<endl;
	system("Pause");
}
```

What is the time complexity of ***ComputeDiameter()? We have to run V times BFS***

Adjacency Matrix = $O(V*(V^2))$

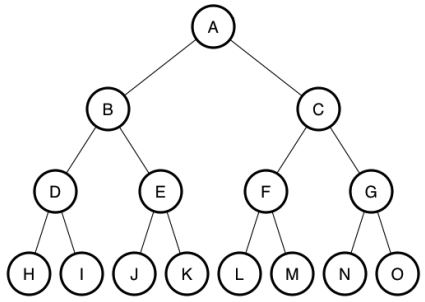
Adjacency List = $O(V*(V+E))$

Roll Number:_____ Section:_____

Question 3: **(Marks: 12+3)**

A node **Z** in a BST is called common ancestor of nodes **X** and **Y** iff both **X** and **Y** exist in the subtree rooted at node **Z**. For example in the tree below B is a common ancestor of both D and K. Write an ***efficient*** C++ function **CommonAncestor()** (member function of class BST) that take the root of a binary tree two integers **X, Y** as parameter and returns all the common ancestors (pointers to the nodes of common ancestors) of **X** and **Y** if both **X** and **Y** exist in the tree. Also note that order of common ancestors must be from nearest ancestor to farthest. For example the common ancestors of L and M are the nodes with data A, C and F. So you should order the ancestors such that F should come first then C and then A.

```
class BSTNode
{
      BSTNode (int );
      BSTNode  * left;
      BSTNode * right;
      int data;
}
```



Create an empty stack S
Min = minimum(x,y)
Max = maximum(x,y)
Done is false
While not done
If Max is less than root's data, push root in S and update root=root->left
If Min is greater than root->data, push root in S and update root=root->right
If Min is less than root->data and Max is greater than root->data, push root in S and done is true
If either Min or Max is equal to root->data, done is true
If both Min and Max found in tree return S
Otherwise return an empty stack

Roll Number:_____ Section:_____

What is the time complexity of your function?

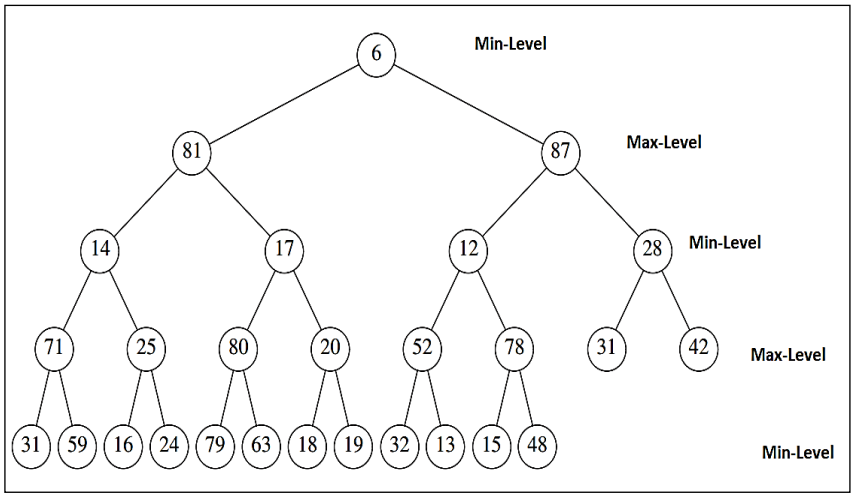
It is O(h) where h is the height of the tree

Question 4: **(Marks: 8+7)**

Min-Max Heap is a data-structure that allow us to find both the minimum and the maximum element of the heap in constant time and perform Insert, Delete-Min and Delete-Max operations in O(lgn) time.

The structure is identical to a binary heap, but the heap-order property is that: root is at depth 0, and for any node, *X*, at **even depth**, the element stored at *X* is smaller than parent but larger than the grandparent

For any node, *Y*, at **odd depth**, the element stored at *Y* is larger than the parent but smaller than the grandparent.



It is now obvious that the minimum element in the heap can be found at the root, and that the maximum element is one of the root's two children.

As Min-Max heap is a complete binary tree so we implement it using **arrays.**

```
class MinMaxHeap{
 public:
      void MinMaxHeap(int capacity = 100);
      void deleteMin();
      void PrintSecondMax();
 private:
      int currentSize; // Number of elements in Min-Max heap
      int * data; // The Min-Max heap array
      int capacity;
};
```

a. Write a C++ function deleteMin to extract minimum element from the Min-Max heap in O(lgn). Note the deleteMin should not violate the Min-Max heap property described above.

Hint: the above DeleteMin operation is more or less similar to deletion in simple heaps.

The operation DeleteMin is analogous to deletion in conventional heaps. Specifically, the required element is extracted and the vacant position is filled with the last element of the heap. The minmax ordering is maintained after applying the TrickleDown procedure

Pseudo -code of Trickle down Min

```
void MinMaxHeap::TrickleDownMin(i) {
       if (A[i] has children) {
              m = index of smallest of the children and grandchildren(if any) of A[i]
              if A[m] is a grandchild of A[i] then
                     if A[m] < A[i]{
                            swap A[i] and A[m]
                            if A[m] > A[parent(m)]{
                                   swap A[m] and A[parent(m)]
                            }
                            TrickleDownMin(m)
                     }
                     else {// if (A[m] is a child of A[i]])
                                   if A[m] < A[i]{
                                          swap A[i] and A[m]
                     }
     }
}
```

b. Write a function with name PrintSecondMax to print the second maximum value in the Min-Max heap.

```
void PrintSecondMax()
{
    int m = minimum(15, currentsize);
    int min=data[2]
    for( int i=3; i<=m; i++){
        if(data[i]<min)
            min = data[i];
    }

    cout<<min<<endl;
}
```