Name:_____________ Roll #: __________

National University of Computer and Emerging Sciences, Lahore Campus

Course:	**Data Structures**	**Course Code:**	**CS 218**
Program:	**BS(Computer Science)**	**Semester:**	**Fall 2020**
Duration:	**180 Minutes**	**Total Marks:**	**100**
Paper Date:	**8-Feb-2021**	**Page(s):**	**9**
Section:	**ALL**	**Section:**	
Exam:	**Final Exam**	**Roll No:**	

Instruction/Notes: Answer in the space provided
Provide properly commented the code for Question 1 and 2
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: 2+3+8+2)**

Given a string **S** consists of n lower case characters {a-z}, we want to count the total number of substrings that contain exactly k distinct characters. For example let S= abbca and k = 3. Set of all possible substrings of S is {a, ab, abb, abbc, abbca, b, bb, bbc, bbca, b, bc, bca, c, ca, a}. The ones with 3 distinct characters are only 4 {abbc, abbca, bbca, bca}. A simple way is to solve this problem is to count number of distinct characters in all possible substrings and add only the ones that have exactly k distinct characters. There are total n(n+1)/2 total substrings. So this method will require O(n^3) time in the worst case.

Another efficient way to solve the same problem is to list only those substrings that have k distinct characters in it. This can be done as follows. Start with the i^{th} character of S and keep on adding subsequent characters until the number of unique characters are less than or equal to k. We can check whether a character **c** is already present in the substring or not by constructing an initially empty hash table. If the character **c** is found in the hash table then it is already present. Otherwise increment the count of distinct characters in the substring by 1 and add the character c in the hash table. This can be done in O(1) time in the worst case. Do the same procedure for $0 \leq i \leq n - k$.

What should be the size of hash table and why?

26

What do you think is the most appropriate hash function for this problem?

Use direct addressing. H(x) = x-97

Write a C++ function ***CountSubStrk(char* S, int k)*** *that takes a string* ***S*** *of* ***n*** *characters all in lower case and an integer* ***k*** *as input.* ***CountSubStrk(char* S, int k)*** *must count the total number of substrings of S that contains exactly k unique characters using the method explained above. You can assume that insert and search functions of the hash table are already implemented.*

```
int CountSubStrk(char* S, int k) {
    int n = strlen(S);
    int count = 0;
    int distinct_char = 0;
    for (int i = 0; i <= n - k; i++) {
        int Hash[26] = { 0 };
        distinct_char = 0;
        for (int j = i; j<n && distinct_char <= k; j++) {
            if (Hash[S[j]-97] == 0) {
                Hash[S[j]-97] = 1;
                distinct_char++;
            }
            if (distinct_char == k)
                count++;

        }
    }
    return count;
}
```

What is the time complexity of your function?

O(n^2)

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

Write a recursive method called *flatten* which takes the root pointer of a BST as its input and transforms the tree into a completely right-skewed tree, i.e. a binary tree in which each node except the leaf node has only a right child. Such a tree is often called a backbone tree and is easier to restructure again into a perfectly balanced BST.

Your function *flatten* is a private member of the class BST and may accept any number of parameters. Carefully specify these parameters. A public overloaded method *flatten*() then uses this private method to create the backbone.

```
…
public:
void flatten(){
void flatten(root, …); //your job is to write this recursive method.
}
```

Hint: Recursively flatten left and right subtrees and then connect them with the root node to flatten the entire tree. You need to find the maximum of left subtree and minimum of right subtree to flatten the entire tree.

```
//public wrapper
void flatten(){
  treeNode * lend, *rend;
  flatten(root, lend, rend);
  root=lend;//the new root is the left most node of the flat list
}

//recursive function
 void flatten(treeNode*n,treeNode*& lend, treeNode*& rend)
{
  if(n==nullptr){
       lend=rend=nullptr;
   }else{
     //flatten in LNR fashion
      treeNode * llend=nullptr,*lrend=nullptr,*rlend=nullptr,*rrend=nullptr;

      if(n->lchild==nullptr)//nothing to flatten on left
              lend=n;//this node itself is the left end
       else{
              flatten(n->lchild, llend, lrend);//flatten left side
              lend=llend;
              lrend->rchild=n;//make connection
       }

       if(n->rchild==nullptr){//nothing to flatten on the right
             rend=n;//if right is null then this node itself is the right end
       }else{
             flatten(n->rchild, rlend, rrend);//flatten right side
             rend=rrend;
             n->rchild=rlend;//make connection
       }

      n->lchild=nullptr; //nullify all left children
  }
}//end of function
```

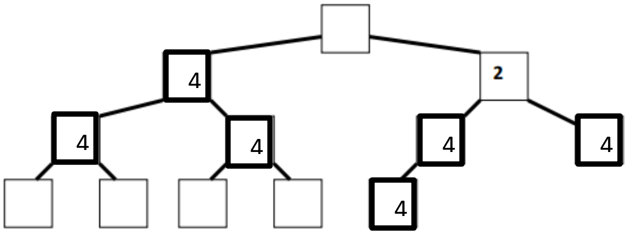
What is the time complexity of your function?

O(n)

Question 3: **(Marks: 5+5)**

a

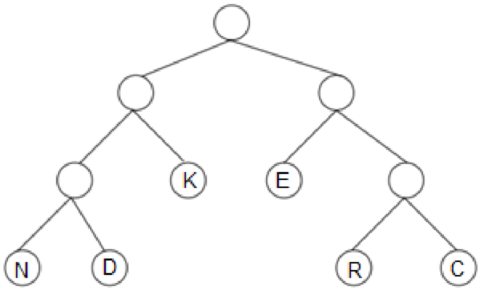
In the following sketch of a Binary Min Heap, the node marked 2 is the one containing the 2nd smallest value in the heap. Mark a 4 for each node that can possibly contain the 4th smallest element in the heap. Assume that there are no duplicate keys.



b.

Decode the message: **1001010001011111001000**

Using the following Huffman tree



EKKNECRKN

Question 4 **(Marks: 5+5)**

a.

Is the following statement true or false? First encircle the correct choice, then justify your answer.

After an element x has been inserted into a Binary Search Tree, both its successor (element that must come after x in sorted order) and predecessor (element that must come before x in sorted order) nodes must be its ancestors in the tree.

TRUE / FALSE

Every new node x is inserted as leaf node. If x is left child of its parent then parent is successor and predecessor y will be the closest ancestor of x such that x lies in the right subtree of y. If no such ancestor exists then x is the minimum element (no predecessor). Similar argument applies if x is right child of its parent.

b.

We have implemented the following Mystery function in the Binary Tree class. What is the functionality of Mystery()? Give a 2-3 line description.

```
bool BT:: Mystery(BTNode * r1, BTNode * r2) {

        if (r1 & r2) {
                if (r1->data == r2->data) {
                        return (Mystery(r1->left, r2->right) && Mystery(r1->right, r2->left));
                }
                else
                    return false;
        }
       else
            return false;

       return true;
}
```

Checking if both the trees are mirror of one another or not

Question 5 **(Marks: 5*4)**

In this course, you have studied and implemented different data structures, including the following:

1. Linked List (Singly, Doubly, Circular)	2. Stack
3. Queues (Linear, Circular)	4. Arrays
5. Binary Trees	6. Heaps
7. Graphs (Directed, Un-Directed)	8. Hash Table

For each of the following applications, indicate which of these data structures would be most suitable and give a brief justification for your choice. For data structures like trees and graphs, describe what information is stored in the vertices and edges, and, if the edges are weighted, describe what information is stored in the weights.

a. Map of the Motorway highway system used to display traffic travel times on a web page. The map displays principle cities, intersections, and major landmarks, the roads that connect them, and the travel times between them along those roads. Travel times along the same road may be different in different directions.

Weighted Directed Graph: principle cities, intersections, and major landmarks are nodes and roads connected them are edges. Travel time is edge weight.

b. A set of the legal words in a WORD game. We want to be able to quickly check whether words used by players do, in fact, exist in the set or not.

Hash Table for efficient and frequent searching

c. The record/history of web sites visited by the user of a web browser. As new sites are visited, they are added to the history. The data structure must also supports the operation of going back to the web page that was previously visited before the current page and going forward to the next page visited.

Either two stacks one for forward and one for backward navigation or doubly linked list

d. Police department maintains a collection of DNAs of all criminals. Whenever they get a new record (a DNA) they have to determine if it is of some existing criminal or not? Which 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 for efficient and frequent searching

Question 6 **(Marks: 5+5)**

Part a

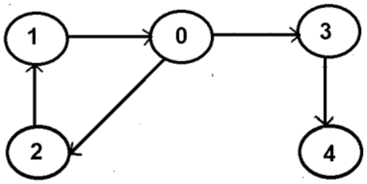
What are the minimum and maximum number of edges in a simple undirected graph G having |V| vertices.

Minimum edges = 0

Maximum edges = |V|*(|V|-1)/2

Part b

A mother vertex in a graph G = (V,E) is a vertex v such that all other vertices in G can be reached by a path from v. There can be more than one mother vertices in a graph. For example, in the below graph, vertices 0, 1 and 2 are mother vertices.



Explain your idea on how to determine if there is mother vertex within a given directed graph or not in 3-4 lines.

For each vertex v, apply BFS or DFS as start vertex. If all the other vertices are visited then v is the mother vertex. If no such vertex exists then there is no mother vertex in the entire graph.

Question 7 **(Marks: 7+3)**

Suppose you have a hash table of size 13, the keys are words, and the hash map is defined as follows: Each letter is assigned a number according to its position in the alphabet, i.e.

A	b	c	d	E	F	G	h	i	j	K	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12
N	O	P	q	R	S	T	u	v	W	X	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

and the primary hash function is “x modulo 13”, where x is the sum of the numbers corresponding to all the letters in the key word.

a. Insert the following list of words into an initially empty hash table using linear probing:

[computer, science, in, lahore, dates, finalterm, to, the, pandemic]

For each word mention the index(s) probed.

H(computer) = (2+14+12+15+20+19+4+17) mod 13 = 103%13=12
H(science) = (18+2+8+4+13+2+4)%13=51%13=12 collision, 0
H(in) = (8+13)%13=21%13=8
H(lahore) = (10+0+7+14+17+4)%13=53%13=1
H(dates) = (3+0+19+4+18)%13 = 44%13=5
H(fianlterm) = (5+8+13+0+11+19+4+17+12)%13=89%13=11
H(to)=(19+14)%13=7
H(the) = (19+7+4)%13=30%13=4
H(pandemic) = (15+0+13+3++12+8+2)%13=57%13=5 collision, 6

0	**Science**
1	**Lahore**
2	
3	
4	**The**
5	**Dates**
6	**Pandemic**
7	**To**
8	**In**
9	
10	
11	**Finalterm**
12	**computer**

b. What is the load factor of the resulting table, and how many collisions occurred?

Load Factor = 9/13 and # Collisions = 2

Question 8 **(Marks: 2*5)**

Give the worst-case running time for each of the following in terms of N. You MUST choose your answer from the following, each of which could be re-used (could be the answer for more than one of (a-e)):

$O(N^2)$, $O(N \log N)$, $O(N)$, $O(N^2 \log N)$, $O(2^N)$, $O(N^3)$, $O(\log N)$, $O(N^N)$, $O(1)$, $O(N^4)$

For any credit, you must explain how you got your answer – be specific as to why the bound you give is appropriate. Assume that the most time-efficient implementation is used and that **all keys are distinct**. Use **N** to represent the total number of elements. **Bases of logarithms are assumed to be 2** unless otherwise specified.

a)	Print out all values in an AVL tree containing N elements from smallest to largest. **Explanation:** **In order traversal prints the data of BST in sorted order and time complexity of in order raversal is O(N)**	O(N)
b)	Insert a value into a binary min heap (implemented using an array) containing N elements. **Explanation:** **New element is added as leaf and then heap-Up function checks the heap property for ancestor node. Height of tree in case of heap is lgN so time for insertion is O(lgN)**	O(lgN)
c)	Finding the minimum value in an AVL tree containing N elements. **Explanation:**	O(lgN)
d)	Given a FIFO queue implemented as a linked list currently containing N values, enqueue N more values so that when you are finished the queue will contain 2N values (Give the total time for enqueueing N more values) **Explanation:**	O(N)
e)	In simple uniform hashing, the search complexity is ____________ [*Considering you are using linear probing for resolving clashes*] **Explanation:**	O(N)