EAST WEST UNIVERSITY
Department of Computer Science and Engineering
B.Sc. in Computer Science and Engineering Program
Final Examination, Spring 2021 Semester, Set: B

Course: CSE 207- Data Structures, Section-2
Instructor: Dr. Maheen Islam, Associate Professor, CSE Department
Full Marks: 50
Time: 1 Hour and 30 Minutes

Note: There are FOUR questions, answer ALL of them.

Problem-1 [10 Marks]

a. Suppose you are a coach of an intra-school football team. You have selected your entire team member. But some more students want to play football. So, you have decided to prepare a waiting list containing name, age, number of matches played, and number of total goals missed of the students. To determine the priority of a student the following formula shown is used.

$$Priority\ number = A + (B-C)/2$$

Now **build** a Binary Heap using the priority number where the smallest number will have the highest priority.

Name	Age (A)	No. of matches played (B)	No. of goals missed (C)
Rony	15	20	30
Jack	13	16	22
Smith	13	12	10
Paul	12	10	8
Sophia	17	28	18
Jorge	12	10	2
Dave	16	12	6
Eva	14	12	4
pinky	13	14	8
Brown	14	22	14

b. Now consider five students don't want play football anymore and you have decided to select top five students from the waiting list constructed in question no 1(a) according to their priority number. Delete the top five priority students from the Heap and perform necessary operations to **rebuild** the heap after deletion.

Problem-2: [10 Marks]

Construct an AVL tree using the following data entered as a sequential set. Show the balance factors in the resulting tree. **(Show detailed steps).**

45, 32, 8, 52, 47, 57, 6, 26, 29

Next, delete the nodes containing 52 and 47 from the constructed tree. **Draw each deleted scenario** while you delete from the tree and make sure that, the tree created after each delete operation must be balanced.

Problem-3: [10 Marks]

Given two arrays that represent preorder and postorder traversals of a full binary tree, construct the binary tree. A Full Binary Tree is a binary tree where every node has either 0 or 2 children

The C code given below creates a Binary tree from the given traversal sequences.

Find the output of the following programs. Show all the calculations (Step-by-step) performed by the function

- `struct node* constructTreeUtil (int pre[], int post[], int* preIndex, int l, int h, int size)`

```
/* program for construction of full binary tree */
#include <stdio.h>
#include <stdlib.h>

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct node
{
    int data;
    struct node *left;
    struct node *right;
};

// A utility function to create a node
struct node* newNode (int data)
{
    struct node* temp = (struct node *) malloc( sizeof(struct node) );

    temp->data = data;
    temp->left = temp->right = NULL;

    return temp;
}

// A recursive function to construct Full from pre[] and post[].
// preIndex is used to keep track of index in pre[].
// l is low index and h is high index for the current subarray in post[]
```

```
struct node* constructTreeUtil (int pre[], int post[], int* preIndex,
                                int l, int h, int size)
{
    // Base case
    if (*preIndex >= size || l > h)
        return NULL;

    // The first node in preorder traversal is root. So take the node at
    // preIndex from preorder and make it root, and increment preIndex
    struct node* root = newNode ( pre[*preIndex] );
    ++*preIndex;

    // If the current subarry has only one element, no need to recur
    if (l == h)
        return root;

    // Search the next element of pre[] in post[]
    int i;
    for (i = l; i <= h; ++i)
        if (pre[*preIndex] == post[i])
            break;

    // Use the index of element found in postorder to divide
    // postorder array in two parts. Left subtree and right subtree
    if (i <= h)
    {
        root->left = constructTreeUtil (pre, post, preIndex,
                                        l, i, size);
        root->right = constructTreeUtil (pre, post, preIndex,
                                         i + 1, h, size);
    }

    return root;
}

// The main function to construct Full Binary Tree from given preorder and
// postorder traversals. This function mainly uses constructTreeUtil()
struct node *constructTree (int pre[], int post[], int size)
{
    int preIndex = 0;
    return constructTreeUtil (pre, post, &preIndex, 0, size - 1, size);
}

// A utility function to print inorder traversal of a Binary Tree
void printInorder (struct node* node)
{
    if (node == NULL)
        return;
    printInorder(node->left);
    printf("%d ", node->data);
    printInorder(node->right);
}

// Driver program to test above functions
```

```
int main ()
{
    int pre[] = {1, 2, 4, 8, 9, 5, 3, 6, 7};
    int post[] = {8, 9, 4, 5, 2, 6, 7, 3, 1};
    int size = sizeof( pre ) / sizeof( pre[0] );

    struct node *root = constructTree(pre, post, size);

    printf("Inorder traversal of the constructed tree: \n");
    printInorder(root);

    return 0;
}
```

Problem-4: [3 +5 + 2 + 5 + 5 = 20 Marks]

(a) **Draw** an undirected connected graph with eight vertices and at least15 edges. The vertices should be called u1, u2, u3, u4, u5, u6, u7 and u8. There must be at least 2 paths of length three from u2 to u5 and a path of length two from u3 to u8.

(b) **Show** depth-first traversal of the graph constructed in part (a), where u5 is the starting node. Visit adjacent nodes in a clockwise ordering from a particular node (12 o'clock position). Show each step to find the traversal sequence using the appropriate data structure.

(c) Convert the graph of part (a) into a weighted graph by assigning weights to each of the edges of the graph; ranging 5 - 15 (randomly pick any value). Do not assign the same weight to more than 2 edges.

(d) **Construct** the minimum spanning tree for the weighted graph constructed in part (c). Also compute the weight. Show the steps you applied to construct the minimum spanning tree.

(e) Use Dijkstra's algorithm to **calculate** minimum cost path (Show detail steps), for the weighted graph constructed in part (c), to travel from node u4 to all other nodes and show the corresponding routes.