Satya Pradhan
231000049
CSE

1. WAP to print preorder in-order and postorder traversal of a tree using Recursion.

Code:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
	int data;
	struct Node* left;
	struct Node* right;
} Node;

Node* createNode(int data) {
	Node* newNode = (Node*)malloc(sizeof(Node));
	newNode->data = data;
	newNode->left = NULL;
	newNode->right = NULL;
	return newNode;
}

void preOrder(Node* root) {
	if (root == NULL) return;
	printf("%d ", root->data);
	preOrder(root->left);
	preOrder(root->right);
}

void inOrder(Node* root) {
	if (root == NULL) return;
	inOrder(root->left);
	printf("%d ", root->data);
	inOrder(root->right);
}

void postOrder(Node* root) {
	if (root == NULL) return;
	postOrder(root->left);
```

```
	postOrder(root->right);
	printf("%d ", root->data);
}

int main() {
	Node* root = createNode(1);
	root->left = createNode(2);
	root->right = createNode(3);
	root->left->left = createNode(4);
	root->left->right = createNode(5);

	printf("Pre-order Traversal: ");
	preOrder(root);
	printf("\n");

	printf("In-order Traversal: ");
	inOrder(root);
	printf("\n");

	printf("Post-order Traversal: ");
	postOrder(root);
	printf("\n");

	return 0;
}
```

2. WAP to print preorder and in-order traversal of a tree without using recursion.

Code:

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

typedef struct Node {
	int data;
	struct Node* left;
	struct Node* right;
} Node;

Node* createNode(int data) {
	Node* newNode = (Node*)malloc(sizeof(Node));
```

```
	newNode->data = data;
	newNode->left = NULL;
	newNode->right = NULL;
	return newNode;
}

void preOrderNonRecursive(Node* root) {
	if (root == NULL) return;

	Node* stack[MAX];
	int top = -1;
	stack[++top] = root;

	while (top != -1) {
	Node* node = stack[top--];
	printf("%d ", node->data);

	if (node->right) stack[++top] = node->right;
	if (node->left) stack[++top] = node->left;
	}
}

void inOrderNonRecursive(Node* root) {
	Node* stack[MAX];
	int top = -1;
	Node* current = root;

	while (current != NULL || top != -1) {
	while (current != NULL) {
		stack[++top] = current;
		current = current->left;
	}
	current = stack[top--];
	printf("%d ", current->data);
	current = current->right;
	}
}

int main() {
	Node* root = createNode(1);
	root->left = createNode(2);
	root->right = createNode(3);
	root->left->left = createNode(4);
```

```
	root->left->right = createNode(5);

	printf("Pre-order Traversal Without Recursion: ");
	preOrderNonRecursive(root);
	printf("\n");

	printf("In-order Traversal Without Recursion: ");
	inOrderNonRecursive(root);
	printf("\n");

	return 0;
}
```

3. WAP to create a Binary Search Tree (BST).

Code:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
	int data;
	struct Node* left;
	struct Node* right;
} Node;

Node* createNode(int data) {
	Node* newNode = (Node*)malloc(sizeof(Node));
	newNode->data = data;
	newNode->left = NULL;
	newNode->right = NULL;
	return newNode;
}

Node* insert(Node* root, int data) {
	if (root == NULL) return createNode(data);

	if (data < root->data) {
	root->left = insert(root->left, data);
	} else if (data > root->data) {
	root->right = insert(root->right, data);
	}
	return root;
}
```

```c
void inOrder(Node* root) {
      if (root == NULL) return;
      inOrder(root->left);
      printf("%d ", root->data);
      inOrder(root->right);
}

int main() {
      Node* root = NULL;
      int n, data;

      printf("Enter number of nodes: ");
      scanf("%d", &n);

      for (int i = 0; i < n; i++) {
      printf("Enter node data: ");
      scanf("%d", &data);
      root = insert(root, data);
      }

      printf("In-order Traversal of BST: ");
      inOrder(root);
      printf("\n");

      return 0;
}
```

4. WAP to delete an element from the BST

Code:

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
   int data;
   struct Node* left;
   struct Node* right;
} Node;

Node* createNode(int data) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   newNode->data = data;
```

```
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

Node* insert(Node* root, int data) {
    if (root == NULL) return createNode(data);
    if (data < root->data) root->left = insert(root->left, data);
    else if (data > root->data) root->right = insert(root->right, data);
    return root;
}

Node* findMin(Node* root) {
    while (root->left != NULL) root = root->left;
    return root;
}

Node* deleteNode(Node* root, int data) {
    if (root == NULL) return root;

    if (data < root->data) root->left = deleteNode(root->left, data);
    else if (data > root->data) root->right = deleteNode(root->right, data);
    else {
        if (root->left == NULL) {
            Node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            Node* temp = root->left;
            free(root);
            return temp;
        }

        Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}

void inOrder(Node* root) {
    if (root == NULL) return;
    inOrder(root->left);
```

```
    printf("%d ", root->data);
    inOrder(root->right);
}

int main() {
    Node* root = NULL;
    int n, data;

    printf("Enter number of nodes: ");
    scanf("%d", &n);

    printf("Enter node data: ");
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        root = insert(root, data);
    }

    printf("In-order Traversal before deletion: ");
    inOrder(root);
    printf("\n");

    printf("Enter value to delete: ");
    scanf("%d", &data);
    root = deleteNode(root, data);

    printf("In-order Traversal after deletion: ");
    inOrder(root);
    printf("\n");

    return 0;
}
```

5. WAP to print the In-order successor of a given node in the BST
Code:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
      int data;
      struct Node* left;
      struct Node* right;
```

```
} Node;

Node* createNode(int data) {
	Node* newNode = (Node*)malloc(sizeof(Node));
	newNode->data = data;
	newNode->left = NULL;
	newNode->right = NULL;
	return newNode;
}

Node* insert(Node* root, int data) {
	if (root == NULL) return createNode(data);
	if (data < root->data) root->left = insert(root->left, data);
	else if (data > root->data) root->right = insert(root->right, data);
	return root;
}

Node* findMin(Node* root) {
	while (root->left != NULL) root = root->left;
	return root;
}

Node* inOrderSuccessor(Node* root, Node* n) {
	if (n->right != NULL) return findMin(n->right);

	Node* succ = NULL;
	while (root != NULL) {
	if (n->data < root->data) {
		succ = root;
		root = root->left;
	} else if (n->data > root->data) {
		root = root->right;
	} else break;
	}
	return succ;
}

int main() {
	Node* root = NULL;
	int n, data;

	printf("Enter number of nodes: ");
	scanf("%d", &n);
```

```
	for (int i = 0; i < n; i++) {
	printf("Enter node data: ");
	scanf("%d", &data);
	root = insert(root, data);
	}

	printf("Enter node value to find successor: ");
	scanf("%d", &data);
	Node* node = root;
	while (node != NULL && node->data != data) {
	if (data < node->data) node = node->left;
	else node = node->right;
	}

	if (node != NULL) {
	Node* succ = inOrderSuccessor(root, node);
	if (succ != NULL) printf("In-order Successor of %d is %d\n", data,
succ->data);
	else printf("No In-order Successor for %d\n", data);
	} else {
	printf("Node not found.\n");
	}

	return 0;
}
```

6. WAP to find the following in the BST
Height of a BST
To find the number of nodes
To find the no of leaves

Code:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
	int data;
	struct Node* left;
	struct Node* right;
} Node;
```

```c
Node* createNode(int data) {
	Node* newNode = (Node*)malloc(sizeof(Node));
	newNode->data = data;
	newNode->left = NULL;
	newNode->right = NULL;
	return newNode;
}

Node* insert(Node* root, int data) {
	if (root == NULL) return createNode(data);
	if (data < root->data) root->left = insert(root->left, data);
	else if (data > root->data) root->right = insert(root->right, data);
	return root;
}

int height(Node* root) {
	if (root == NULL) return 0;
	int leftHeight = height(root->left);
	int rightHeight = height(root->right);
	return (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;
}

int nodeCount(Node* root) {
	if (root == NULL) return 0;
	return 1 + nodeCount(root->left) + nodeCount(root->right);
}

int leafCount(Node* root) {
	if (root == NULL) return 0;
	if (root->left == NULL && root->right == NULL) return 1;
	return leafCount(root->left) + leafCount(root->right);
}

int main() {
	Node* root = NULL;
	int n, data;

	printf("Enter number of nodes: ");
	scanf("%d", &n);

	for (int i = 0; i < n; i++) {
	printf("Enter node data: ");
```

```
        scanf("%d", &data);
        root = insert(root, data);
        }

        printf("Height of BST: %d\n", height(root));
        printf("Number of nodes: %d\n", nodeCount(root));
        printf("Number of leaves: %d\n", leafCount(root));

        return 0;
}
```

7. WAP to implement Heap Sort using top-down approach

Code:

```
#include <stdio.h>

void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest]) largest = left;
    if (right < n && arr[right] > arr[largest]) largest = right;

    if (largest != i) {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;
        heapify(arr, n, largest);
    }
}

void heapSort(int arr[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--) heapify(arr, n, i);
    for (int i = n - 1; i >= 0; i--) {
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;
        heapify(arr, i, 0);
    }
}

int main() {
```

```
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    heapSort(arr, n);

    printf("Sorted array: ");
    for (int i = 0; i < n; i++) printf("%d ", arr[i]);
    printf("\n");

    return 0;
}
```

8. WAP to implement the Heap Sort using bottom-up approach

Code:

```
#include <stdio.h>

void heapify(int arr[], int n, int i) {
    int smallest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] < arr[smallest]) {
        smallest = left;
    }
    if (right < n && arr[right] < arr[smallest]) {
        smallest = right;
    }

    if (smallest != i) {
        int temp = arr[i];
        arr[i] = arr[smallest];
        arr[smallest] = temp;

        heapify(arr, n, smallest);
    }
}

void heapSort(int arr[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
```

```
    }

    for (int i = n - 1; i > 0; i--) {
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;

        heapify(arr, i, 0);
    }
}

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    heapSort(arr, n);

    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

9. WAP to print all the nodes and it's In-order successors of a BST

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
} Node;

Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = NULL;
```

```
    newNode->right = NULL;
    return newNode;
}

Node* insert(Node* root, int data) {
    if (root == NULL) return createNode(data);
    if (data < root->data) root->left = insert(root->left, data);
    else if (data > root->data) root->right = insert(root->right, data);
    return root;
}

Node* findMin(Node* root) {
    while (root->left != NULL) root = root->left;
    return root;
}

Node* inOrderSuccessor(Node* root, Node* n) {
    if (n->right != NULL) return findMin(n->right);

    Node* succ = NULL;
    while (root != NULL) {
        if (n->data < root->data) {
            succ = root;
            root = root->left;
        } else if (n->data > root->data) {
            root = root->right;
        } else break;
    }
    return succ;
}

void printSuccessors(Node* root) {
    if (root == NULL) return;

    Node* succ = inOrderSuccessor(root, root);
    printf("Node: %d, In-order Successor: %d\n", root->data, (succ ?
succ->data : -1));

    printSuccessors(root->left);
    printSuccessors(root->right);
}

int main() {
```

```
    Node* root = NULL;
    int n, data;

    printf("Enter number of nodes: ");
    scanf("%d", &n);

    for (int i = 0; i < n; i++) {
        printf("Enter node data: ");
        scanf("%d", &data);
        root = insert(root, data);
    }

    printf("In-order Successors of all nodes:\n");
    printSuccessors(root);

    return 0;
}
```

10. Student Instructor has to demonstrate the following
Bubble Sort:

```
#include <stdio.h>

int main() {
    int n;
    scanf("%d", &n);
    int arr[n];
    for (int i = 0; i <n ; i++) {
        scanf("%d", &arr[i]);
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n-1; j++) {
            if (arr[j+1] < arr[j]) {
                int t = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = t;
            }
        }
    }
    for (int i = 0; i <n ; i++) {
```

```
        printf("%d ", arr[i]);
    }

}
```

Insertion Sort:

```
#include <stdio.h>

int main() {
    int n;
    scanf("%d", &n);
    int arr[n];
    for (int i = 0; i <n ; i++) {
        scanf("%d", &arr[i]);
    }
    for (int i = 1; i < n; i++) {
        for (int j = i; j >= 1; j--) {
            if (arr[j-1] > arr[j]) {
                int t = arr[j-1];
                arr[j - 1] = arr[j];
                arr[j] = t;
            } else {
                break;
            }
        }
    }
    for (int i = 0; i <n ; i++) {
        printf("%d ", arr[i]);
    }
}
```

Selection Sort:

```
#include <stdio.h>

int main () {
    int n;
    scanf("%d", &n);
    int arr[n];
    for (int i = 0; i <n ; i++) {
        scanf("%d", &arr[i]);
```

```
    }
    for (int i = 0; i <n; i++){
        for (int j = i+1; j < n; j++) {
            if (arr[j] < arr[i]) {
                int t = arr[j];
                arr[j] = arr[i];
                arr[i] = t;
            }
        }
    }
    for (int i = 0; i <n ; i++) {
        printf("%d ", arr[i]);
    }
}
```