GLA University, Mathura

Data Structure and Algorithm Lab BCSC 0805

(Lab Record)

Submitted by: Priyanshu Vishwakarma

191500611

B.tech

CSE

2nd Year

4th Sem

Section-B

14

----------Github(Codes)----------
-

Experiment-1

Write a program to print all the duplicate elements present in the given array.
for example 1,8,1,9,2,3,8,6,9,1
output 1,8,9

```
public class DuplicateElements {
    public void Duplicate(int arr[]){
```

```
        System.out.println("Printing Duplicate Elements");
        for(int i = 0; i < arr.length; i++) {
            for(int j = i + 1; j < arr.length; j++) {
                if(arr[i] == arr[j]) {
                    System.out.println(arr[j]);
                }
            }
        }
    }

    public static void main(String[] args) {
        int [] arr = {1, 2, 3, 4, 2, 7, 8, 8, 3};
        DuplicateElements obj=new DuplicateElements();
        obj.Duplicate(arr);
    }
}
```

Experiment-2

Consider a Singly linked list, where each node can store an integer value. You task is to form a number, whose digits are stored in given singly linked list. The start node contain the first digit and last node contain the last digit
Method
public int getNumber(SinglyLinkedList list){
}
For example:
1→ 8 → 6 → 2 → null
Then you have to give output
1862

```
public class SinglyLinkedList {
        class Node{
            int data;
            Node next;

            public Node(int data) {
                this.data = data;
                this.next = null;
            }
        }

        public Node head = null;
        public Node tail = null;


        public void addAtEnd(int data) {
            //Create a new node
```

```
            Node newNode = new Node(data);

            //Checks if the list is empty
            if(head == null) {
                //If list is empty, both head and tail will point to new node
                head = newNode;
                tail = newNode;
            }
            else {
                //newNode will be added after tail such that tail's next will
point to newNode
                tail.next = newNode;
                //newNode will become new tail of the list
                tail = newNode;
            }
        }

        //display() will display all the nodes present in the list
        public void display() {
            //Node current will point to head
            Node current = head;
            if(head == null) {
                System.out.println("List is empty");
                return;
            }
            while(current != null) {
                //Prints each node by incrementing pointer
                System.out.print(current.data);
                current = current.next;
            }
            System.out.println();
        }

        public static void main(String[] args) {

            SinglyLinkedList sList = new SinglyLinkedList();


            sList.addAtEnd(1);
            sList.addAtEnd(2);
            sList.addAtEnd(3);
            sList.addAtEnd(4);
            sList.display();
        }
    }
```

Experiment-3

Write a program to Implement Singly linked list.

```java
package Implementation;

import myinterfaces.MySinglyLinkedListADT;

public class MySinglyLinkedList implements MySinglyLinkedListADT {
    private Node head;
    private Node tail;
    // No. of Elements in linked list
    private int size;

    // Constructor
    public MySinglyLinkedList() {
        head = null;
        tail = null;
        size = 0;
    }

    @Override
    public void addFirst(int element) {
        Node node = new Node(element);
        if (isEmpty()) {
            head = node;
            tail = node;
            size++;
        } else {
            node.setNext(head);
            head = node;
            size++;
        }

    }

    @Override
    public boolean isEmpty() {
        return head == null;
    }

    @Override
    public int size() {
        return size;
    }

    @Override
    public void addLast(int element) {
        Node node = new Node(element);
        if (isEmpty()) {
            addFirst(element);
        } else {
            tail.setNext(node);
            tail = node;
            size++;
        }
```

```
    }

    @Override
    public void addLastWithoutUsingTail(int element) {
        Node node = new Node(element);
        if (isEmpty()) {
            head = node;
            tail = node;
        } else {
            // traverse till you find the next node
            Node temp = head;
            while (temp.getNext() != null) {
                //  Update the value of temp

                //this process is known as link hopping or pointer hopping'
                temp = temp.getNext();

            }
            temp.setNext(node);
            tail = node;
        }
    }

    @Override
    public void traverse() {
        System.out.println();
        if (!isEmpty()) {
            Node temp = head;
            while (temp != null) {
                System.out.print(temp.getData() + "-->");
                temp = temp.getNext();
            }
            System.out.println("null");
        } else {
            System.out.println("empty list");
        }
    }

    @Override
    public int removeFirst() {
        int response = 0;
        if (!isEmpty()) {
            // Single node
            response = head.getData();
            // size--;
            if (head == tail) {
                head = null;
                tail = null;
            }
            // multiple node
            else {
                head = head.getNext();
            }
```

```
            size--;
        }
        return response;
    }

    @Override
    public int removeLast() {
        // Removing Last node using 2 Pointer method
        int response = 0;
        if (!isEmpty()) {
            // Single node
            response = tail.getData();
            if (head == tail) {
                head = null;
                tail = null;
            }
            // Multiple Nodes
            else {
                Node temp = head;
                Node previous = null;
                while (temp.getNext() != null) {
                    previous = temp;
                    temp = temp.getNext();
                }
                previous.setNext(null);
                tail = previous;
            }
            size--;
        }


        return response;
    }

    @Override
    public int first() {
        if (isEmpty()) {
            return 0;// Considering 0 is invalid
        } else {
            return head.getData();
        }

        // Another Way
//        int response = 0;
//        if(!isEmpty()){
//            response= head.getData();
//        }
//        return 0;

    }

    @java.lang.Override
    public int Last() {
        int response = 0;
```

```
        if (!isEmpty()) {
            response = tail.getData();
        }
        return response;
    }

    @java.lang.Override
    public boolean search(int searchElement) {
        boolean response = false;
        Node temp = head;
        while (temp != null) {
            if (temp.getData() == searchElement) {
                response = true;
                break;
            } else {
                temp = temp.getNext();
            }
        }

        return response;
    }

    @Override
    public void addBefore(int element, int beforeNode) {
        Node temp = head;
        Node previous = null;
        while (temp != null) {
            if (temp.getData() == beforeNode) {
                break;
            } else {
                previous = temp;
                temp = temp.getNext();
            }
        }
        if (temp != null) {
            Node node = new Node(element);
            node.setNext(temp);
            if (previous == null) {
                //adding at head
                head = node;
            } else {
                previous.setNext(node);
            }
            size++;
        }
    }

    @Override
    public void addAfter(int element, int afterNodeData) {
        Node temp = head;
        while (temp != null) {
            if (temp.getData() == afterNodeData) {
                break;
            }
```

```
            temp = temp.getNext();
        }
        if (temp != null) {
            Node node = new Node(element);
            node.setNext(temp.getNext());
            temp.setNext(node);

            size++;
            //check if temp is last node, then tail must refer to node
            if (temp == tail) {
                tail = node;
            }
        }
    }

    public Node getLastNodeDataWithoutUsingTail() {
        Node temp = head;
        if (isEmpty()) {
            return null;
        } else {
            while (temp.getNext() != null) {
                temp = temp.getNext();
            }
            // System.out.println(temp.getData());
            return temp.getNext();
        }
    }

    public int getLastNodeWithoutUsingTail() {
        Node temp = head;
        if (isEmpty()) {
            return 0;
        } else {
            while (temp.getNext() != null) {
                temp = temp.getNext();
            }
            // System.out.println(temp.getData());
            return temp.getData();
        }


    }
}
```

Experiment-4

Write a program to implement circular linked list.

```
public class CircularLinkedList {
    private Node head;
    private Node tail;

    public void insert(int data){
        Node newNode = new Node(data);
        if (head == null){
            head = newNode;
        }else {
            tail.setNextNode(newNode);
        }
        tail = newNode;
        tail.setNextNode(head);
    }
    public boolean search(int data){
        if (head == null){
            return false;
        }else {
            Node currentNode = head;
            while (currentNode.getNextNode() != head){
                if (currentNode.getData() == data){
                    return true;
                }
                currentNode = currentNode.getNextNode();
            }
        }
        return false;
    }
    public void delete(int data){
        Node currentNode = head;
        if (head != null){
            if (currentNode.getData() == data){
                head = currentNode.getNextNode();
                tail.setNextNode(head);
            }else {
                while(currentNode.getNextNode() != head){
                    if (currentNode.getNextNode().getData() == data){

currentNode.setNextNode(currentNode.getNextNode().getNextNode());
                        break;
                    }
                    currentNode = currentNode.getNextNode();
                }
            }
        }
    }
    public void traverse(){
        if (head != null){
            Node currentNode = head;
            while (currentNode.getNextNode() != head){
                System.out.print(currentNode.getData() + " ");
                currentNode = currentNode.getNextNode();
            }
```

```
        }
        System.out.print(tail.getData());
    }
}
```

Experiment-5

Write a program to implement Stack Using Array.

```
package implementation;

import MyInterface.MyStackADT;

public class MyStack implements MyStackADT {
    // Maximum of elements in stack
    private final int MAX_CAPACITY;
    // to store elements of stack
    private int[] arr;
    // top
    int top;

    public MyStack(int MAX_CAPACITY) {
        this.MAX_CAPACITY = MAX_CAPACITY;
        // constructor an array
        arr=new int[MAX_CAPACITY];
        top=-1;
    }

    @Override
    public void push(int element) {
        if(isFull()){
            top++;
            arr[top]=element;
            System.out.println("Element Inserted");
        }else{
            System.out.println("Stack Overflow");
        }
    }

    private boolean isFull() {
        return top != MAX_CAPACITY - 1;
    }

    @Override
    public int pop() {
        int response=0;
        if(!isEmpty()){
            response =arr[top];
```

```
            top--;
        }else{
            System.out.println("Stack Underflow");
        }
        return response;
    }

    @Override
    public int peek() {
        int response=0;
        if(!isEmpty()){
            response=arr[top];
        }else{
            System.out.println("Stack is Empty");
        }
        return response;
    }

    @Override
    public boolean isEmpty() {
        boolean response =false;
        if(top==-1){
            response =true;
        }
        return response;
    }

    @Override
    public int size() {
        return top + 1;
    }

    public void traverse(){
        for (int i = 0; i <=top; i++) {
            System.out.print(arr[i]+", ");
        }
        System.out.println();
    }
    public boolean search(int searchElement){
        boolean response=false;
        for (int i  = 0; i <=top ; i++) {
            if(arr[i]==searchElement){
                response=true;
                break;
            }
        }
        return response;
    }
}
```

Experiment-6

Write a program to implement Stack using linked list.

```
package implementation;

import myinterface.StackADT;

public class MyStack implements StackADT {
    Node top;
    int size=0;

    public MyStack() {
        this.size = 0;
        this.top = null;
    }

    @Override
    public void push(int element) {
        Node node=new Node(element);
            node.setNext(top);
            top=node;
            size++;

    }

    @Override
    public int pop() {
        int response=0; // 0 is invalid data
        if(!isEmpty()){
            response=top.getData();
           top= top.getNext();
           size--;
        }else{
            System.out.println("Stack is Underflow");
        }
        return response;
    }

    @Override
    public int peek() {
      int response=0;
      if(!isEmpty()){
          response=top.getData();
      }else{
          System.out.println("Stack is Empty");
      }
      return response;
    }

    @Override
    public boolean isEmpty() {
```

```
        return top==null;
    }

    @Override
    public int size() {
        return size;
    }

    @Override
    public void traverse() {
        if(!isEmpty()){
            Node temp=top;
            while(temp!=null){
                System.out.print(temp.getData()+"-->");
                temp=temp.getNext();
            }
            System.out.println("null");
        }else{
            System.out.println("Stack is Empty");
        }
    }

    @Override
    public void search() {

    }
}
```

Experiment-7

Write a program to implement Queue using Array.

```
import java.util.Scanner;

public class QueueUsingArray {

        static int front=-1;
        static int rear=-1;
        static int size;
        static int queue[];
        static Scanner sc=new Scanner(System.in);
        public static void main(String[] args){
            System.out.println("Enter the Size of Queue");
            size=sc.nextInt();
            queue=new int[size];
            int f=1;
            int d=1;
            while(f==1){
```

```
                System.out.println("Enter the data");
                enqueue(sc.nextInt());
                System.out.println(" repeat again press'1' ");
                f=sc.nextInt();
            }

            while(d==1){
                System.out.println("Delete the element:-   "+dequeue());
                System.out.println(" repeat again press'1' ");
                d=sc.nextInt();
            }
            display();
        }
        public static void enqueue(int data){
            if (isFull())
                System.out.println("Queue is Full ");
            else{
                rear++;
                queue[rear]=data;
            }
        }
        public static int dequeue(){
            int x=-1;
            if (isEmpty())
                System.out.println("Queue is Empty");
            else{
                front++;
                x=queue[front];
            }
            return x;
        }
        public static boolean isEmpty(){
            if (front==rear)
                return true;
            return false;
        }
        public static boolean isFull(){
            if (rear==(size-1))
                return true;
            return false;
        }
        public static void display(){
            System.out.println("display the queue");
            for(int i=front+1;i<rear+1;i++)
                System.out.println(queue[i]);
        }
    }
```

Experiment-8

Write program to implement Queue using linked list.

```
public class MyQueue {


        // a field to refer rear end of queue
        private Node rear;
        // a field to refer front end of queue
        private Node front;
        private int size;// total number of elements in the queue

        // constructor
        public MyQueue() {
            front = null;
            rear = null;
            size = 0;
        }

        // a method to check whether queue is empty
        public boolean isEmpty() {
            boolean response = false;
            if (size == 0) {
                response = true;
            }
            return response;
        }


        // a method to add a new element in queue
        public void enqueue(int element) {
            Node node = new Node(element);
            if (front == null) {
                rear = node;
                front = node;
                size++;
            } else {
                rear.setNext(node);
                rear = node;
                size++;
            }
        }

        // a method to remove element of queue from front
        public Node dequeue() {
            Node response = null;
            if (front != null) {
                if (front.getNext() != null) {
                    response = new Node(front.getData());
                    front = front.getNext();
                    size--;
                } else {
                    response = new Node(front.getData());
                    front = null;
                    rear = null;
                    size--;
```

```
                }
            }
            return response;
        }

        // a method to get front element without removing it
        public Node peek() {
            Node response = null;
            if (!isEmpty()) {
                response = new Node(front.getData());
            }
            return response;
        }

        // a method to get size of queue
        public int getSize() {
            return size;
        }
}
```

Experiment-9

Write a program to implement Binary search tree (BST).

```
public class MyBinarySearchTree {

        private TreeNode root;

        public TreeNode getRoot() {
            return root;
        }

        public void insert(int data){
            TreeNode node = new TreeNode(data);

            if(root == null){
                root = node;
            }
            else{
                TreeNode temp = root;
                TreeNode parent = null;
                while (temp != null){
                    parent = temp;
                    if(node.getData() <= temp.getData()){
                        temp = temp.getLeft();
                    }
                    else{
                        temp = temp.getRight();
```

```
                }
            }
            if(node.getData() <= parent.getData()){
                parent.setLeft(node);
            }
            else {
                parent.setRight(node);
            }
        }
    }
    public boolean search(int data){
        boolean response = false;
        TreeNode temp = root;
        while (temp !=null){
            if(temp.getData() == data) {
                response = true;
                break;
            }
            if(data <= temp.getData()) {
                temp = temp.getLeft();
            }
            else{
                temp = temp.getRight();
            }

        }


        return response;

    }
    public void traversePreOrder(TreeNode node){
        if(node == null) {

        }
        else{
            System.out.print(node.getData() + ",");
            traversePreOrder(node.getLeft());
            traversePreOrder(node.getRight());
        }
    }

    public void traverseInOrder(TreeNode node){
        if(node == null) {
        }
        else{
            traverseInOrder(node.getLeft());
            System.out.print(node.getData() + ",");
            traverseInOrder(node.getRight());
        }

    }

    public void traversePostOrder(TreeNode node){
```

```
            if(node == null) {
            }
            else{
                traversePostOrder(node.getLeft());
                traversePostOrder(node.getRight());
                System.out.print(node.getData() + ",");

            }
        }
        public TreeNode delete(int data){
            TreeNode response = null;
            TreeNode temp = root;
            TreeNode parent = null;
            //searching the node with given data
            while (temp != null && temp.getData() != data ){
                parent = temp;
                if(data < temp.getData()){
                    temp = temp.getLeft();
                }
                else{
                    temp = temp.getRight();
                }
            }
            if(temp != null){
                response = temp;
                if(isLeaf(temp)){

                    // that means given node is leaf node
                    // to delete it, remove its reference from parent
                    if(parent != null) {
                        if (data < parent.getData()) {
                            parent.setLeft(null);
                        } else {
                            parent.setRight(null);
                        }
                    }
                    else{
                        root = null;
                    }
                }


                else if(temp.getLeft() != null && temp.getRight() == null){
                    //response = temp;
                    if(parent != null) {
                        if (data < parent.getData()) {
                            parent.setLeft(temp.getLeft());
                        } else {
                            parent.setRight(temp.getLeft());
                        }
                    }
                    else{
                        root = temp.getLeft();
                    }
```

```
                }
                else if(temp.getRight() != null && temp.getLeft() == null){
                    //response = temp;
                    if(parent != null) {
                        if (data < parent.getData()) {
                            parent.setLeft(temp.getRight());
                        } else {
                            parent.setRight(temp.getRight());
                        }
                    }
                    else{
                        root = temp.getRight();
                    }
                }
                else{
                    response = temp;
                    TreeNode successor = getSuccessor(temp);
                    System.out.println("successor is ==>" + successor.getData());
                    successor.setRight(temp.getRight());
                    successor.setLeft(temp.getLeft());
                    if(parent != null) {
                        if (data < parent.getData()) {
                            parent.setLeft(successor);
                        } else {
                            parent.setRight(successor);
                        }
                    }
                    else{
                        root = successor;

                    }
                }

            }


            return response;
        }

        private TreeNode getSuccessor(TreeNode temp) {
            TreeNode response = null;
            temp = temp.getRight();
            while(temp!= null){
                response = temp;
                temp = temp.getLeft();
            }
            response = delete(response.getData());
            return response;
        }

        private boolean isLeaf(TreeNode node){
            return (node.getLeft() == null && node.getRight() == null);
        }
```

```
public void traverseLevelOrder(TreeNode node){
    if(node != null) {
        Queue<TreeNode> q = new LinkedList<>();
        q.add(node);
        while (!q.isEmpty()) {
            TreeNode current = q.remove();
            System.out.print(current.getData() + ",");
            if (current.getLeft() != null) {
                q.add(current.getLeft());
            }
            if (current.getRight() != null) {
                q.add(current.getRight());
            }
        }
    }
    else{
        System.out.println("empty tree");
    }
}

public void traversePreOrderWithoutRecursion(TreeNode node){
    if(node != null){
        Stack<TreeNode> s = new Stack<>();
        s.push(node);
        while(!s.empty()){
            TreeNode current = s.pop();
            System.out.print(current.getData() + ",");
            if(current.getRight() != null){
                s.push(current.getRight());
            }
            if(current.getLeft() != null){
                s.push(current.getLeft());
            }
        }
    }
    else{
        System.out.println("empty tree");
    }
}
public void traverseInOrderWithoutRecursion(TreeNode node){
    if(node != null){
        Stack<TreeNode> s = new Stack<>();
        s.push(null);

        while(s.peek() !=null){
            s.push(node);

        }
    }
    else{
        System.out.println("empty tree");
    }
}
```

```
        public int height(TreeNode node){
            if(node == null){
                return -1;
            }
            else{
                return 1 + Math.max(height(node.getLeft()),
height(node.getRight()));
            }
        }
        public int heightNonRecursive(TreeNode node){
            if(node == null){
                return -1;
            }
            Queue<TreeNode> q = new LinkedList<>();
            q.add(node);
            int height = -1;

            while (!q.isEmpty()){
                int size = q.size();
                height++;
                while(size > 0 ){
                    TreeNode curr = q.remove();
                    if(curr.getLeft() != null){
                        q.add(curr.getLeft());
                    }
                    if(curr.getRight() != null){
                        q.add(curr.getRight());
                    }
                    size--;
                }

            }
            return height;
        }
```

Experiment-10

Write a program to search an element with in linked list using linear search.

```
public class LinearSearchLinkedList {


    //Node class
    class Node
    {
        int data;
```

```
        Node next;
        Node(int d)
        {
            data = d;
            next = null;
        }
    }


    Node head; //Head of list

    public void push(int new_data)
    {
        Node new_node = new Node(new_data);

        new_node.next = head;

        head = new_node;
    }

    public boolean search(Node head, int x)
    {
        Node current = head;
        while (current != null)
        {
            if (current.data == x)
                return true;
            current = current.next;
        }
        return false;
    }


    public static void main(String args[])
    {

        //Start with the empty list
        LinearSearchLinkedList llist = new LinearSearchLinkedList();

        llist.push(10);
        llist.push(30);
        llist.push(11);
        llist.push(21);
        llist.push(14);

        if (llist.search(llist.head, 21))
            System.out.println("Yes");
        else
            System.out.println("No");
    }
}
```

Experiment-11

Write a program to search an element with in array using binary search.

```
public class BinarySearchLinkedList {

    class Node
    {
        int data;
        Node next;

        Node(int d)
        {
            data = d;
            next = null;
        }
    }



        static Node push(Node head, int data)
        {
            Node newNode = new Node(data);
            newNode.next = head;
            head = newNode;
            return head;
        }


        static Node middleNode(Node start, Node last)
        {
            if (start == null)
                return null;

            Node slow = start;
            Node fast = start.next;

            while (fast != last)
            {
                fast = fast.next;
                if (fast != last)
                {
                    slow = slow.next;
                    fast = fast.next;
                }
            }
            return slow;
        }
```

```
        // function to insert a node at the beginning
        // of the Singly Linked List
        static Node binarySearch(Node head, int value)
        {
            Node start = head;
            Node last = null;

            do
            {
                // Find Middle
                Node mid = middleNode(start, last);

                // If middle is empty
                if (mid == null)
                    return null;

                // If value is present at middle
                if (mid.data == value)
                    return mid;

                    // If value is less than mid
                else if (mid.data > value)
                {
                    start = mid.next;
                }

                // If the value is more than mid.
                else
                    last = mid;
            } while (last == null || last != start);

            // value not present
            return null;
        }

        // Driver Code
        public static void main(String[] args)
        {
            Node head = null;


            head = push(head, 1);
            head = push(head, 4);
            head = push(head, 7);
            head = push(head, 8);
            head = push(head, 9);
            head = push(head, 10);
            int value = 7;

            if (binarySearch(head, value) == null)
            {
                System.out.println("Value not present");
            }
            else
```

```
            {
                System.out.println("Present");
            }
        }
    }
```

Experiment-12

Write a program to Sort the given array using Selection sort.

```
public class SelectionSort {


        void sort(int arr[])
        {
            int n = arr.length;

            // One by one move boundary of unsorted subarray
            for (int i = 0; i < n-1; i++)
            {
                // Find the minimum element in unsorted array
                int min_idx = i;
                for (int j = i+1; j < n; j++)
                    if (arr[j] < arr[min_idx])
                        min_idx = j;

                // Swap the found minimum element with the first
                // element
                int temp = arr[min_idx];
                arr[min_idx] = arr[i];
                arr[i] = temp;
            }
        }

        // Prints the array
        void printArray(int arr[])
        {
            int n = arr.length;
            for (int i=0; i<n; ++i)
                System.out.print(arr[i]+" ");
            System.out.println();
        }

        // Driver code to test above
        public static void main(String args[])
        {
            SelectionSort ob = new SelectionSort();
            int arr[] = {64,25,12,22,11};
```

```
            ob.sort(arr);
            System.out.println("Sorted array");
            ob.printArray(arr);
        }
    }
```

Experiment-13

Write a program to Sort the given array using Insertion sort.

```
public class InsertionSort {
    public static void insertionSort(int array[]) {
        int n = array.length;
        for (int j = 1; j < n; j++) {
            int key = array[j];
            int i = j-1;
            while ( (i > -1) && ( array [i] > key ) ) {
                array [i+1] = array [i];
                i--;
            }
            array[i+1] = key;
        }
    }

    public static void main(String a[]){
        int[] arr1 = {9,14,3,2,43,11,58,22};
        System.out.println("Before Insertion Sort");
        for(int i:arr1){
            System.out.print(i+" ");
        }
        System.out.println();

        insertionSort(arr1);//sorting array using insertion sort

        System.out.println("After Insertion Sort");
        for(int i:arr1){
            System.out.print(i+" ");
        }
    }
}
```

Experiment-14

Write a program to Sort the given array using Quick sort.

```
class QuickSort
{

    int partition(int arr[], int low, int high)
    {
        int pivot = arr[high];
        int i = (low-1); // index of smaller element
        for (int j=low; j<high; j++)
        {
            // If current element is smaller than or
            // equal to pivot
            if (arr[j] <= pivot)
            {
                i++;

                // swap arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // swap arr[i+1] and arr[high] (or pivot)
        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;

        return i+1;
    }


    /* The main function that implements QuickSort()
    arr[] --> Array to be sorted,
    low --> Starting index,
    high --> Ending index */
    void sort(int arr[], int low, int high)
    {
        if (low < high)
        {
            /* pi is partitioning index, arr[pi] is
            now at right place */
            int pi = partition(arr, low, high);

            // Recursively sort elements before
            // partition and after partition
            sort(arr, low, pi-1);
            sort(arr, pi+1, high);
        }
    }

    /* A utility function to print array of size n */
    static void printArray(int arr[])
```

```
    {
        int n = arr.length;
        for (int i=0; i<n; ++i)
            System.out.print(arr[i]+" ");
        System.out.println();
    }

    // Driver program
    public static void main(String args[])
    {
        int arr[] = {10, 7, 8, 9, 1, 5};
        int n = arr.length;

        QuickSort ob = new QuickSort();
        ob.sort(arr, 0, n-1);

        System.out.println("sorted array");
        printArray(arr);
    }
}
```

Experiment-15

Write a program to Sort the given array using Merge sort.

```
import java.util.Arrays;


class Main {

    void merge(int array[], int p, int q, int r) {

        int n1 = q - p + 1;
        int n2 = r - q;

        int L[] = new int[n1];
        int M[] = new int[n2];

        // fill the left and right array
        for (int i = 0; i < n1; i++)
            L[i] = array[p + i];
        for (int j = 0; j < n2; j++)
            M[j] = array[q + 1 + j];


        int i, j, k;
        i = 0;
        j = 0;
```

```
        k = p;

        // use if(L[i] >= <[j])
        while (i < n1 && j < n2) {
            if (L[i] <= M[j]) {
                array[k] = L[i];
                i++;
            } else {
                array[k] = M[j];
                j++;
            }
            k++;
        }

        // When we run out of elements in either L or M,
        // pick up the remaining elements and put in A[p..r]
        while (i < n1) {
            array[k] = L[i];
            i++;
            k++;
        }

        while (j < n2) {
            array[k] = M[j];
            j++;
            k++;
        }
    }

    void mergeSort(int array[], int left, int right) {
        if (left < right) {

            int mid = (left + right) / 2;

            mergeSort(array, left, mid);
            mergeSort(array, mid + 1, right);

            // Merge the sorted sub arrays
            merge(array, left, mid, right);
        }
    }

    public static void main(String args[]) {

        // created an unsorted array
        int[] array = { 6, 5, 12, 10, 9, 1 };

        Main ob = new Main();


        ob.mergeSort(array, 0, array.length - 1);

        System.out.println("Sorted Array:");
        System.out.println(Arrays.toString(array));
```

```
    }
}
```

Experiment-16

Write a program to Sort the given singly linked list in O(n) time. The given link list contains integer values 0, 1, & 2 only.

```
public class LinkedList {


        Node head; // head of list

        /* Linked list Node*/
        class Node
        {
            int data;
            Node next;
            Node(int d) {data = d; next = null; }
        }

        void sortList()
        {
            // initialise count of 0 1 and 2 as 0
            int count[] = {0, 0, 0};

            Node ptr = head;

            /* count total number of '0', '1' and '2'
             * count[0] will store total number of '0's
             * count[1] will store total number of '1's
             * count[2] will store total number of '2's */
            while (ptr != null)
            {
                count[ptr.data]++;
                ptr = ptr.next;
            }

            int i = 0;
            ptr = head;

            /* Let say count[0] = n1, count[1] = n2 and count[2] = n3
             * now start traversing list from head node,
             * 1) fill the list with 0, till n1 > 0
             * 2) fill the list with 1, till n2 > 0
             * 3) fill the list with 2, till n3 > 0 */
            while (ptr != null)
            {
```

```
                if (count[i] == 0)
                    i++;
                else
                {
                    ptr.data= i;
                    --count[i];
                    ptr = ptr.next;
                }
            }
        }


        /* Utility functions */

        /* Inserts a new Node at front of the list. */
        public void push(int new_data)
        {
        /* 1 & 2: Allocate the Node &
                  Put in the data*/
            Node new_node = new Node(new_data);

            /* 3. Make next of new Node as head */
            new_node.next = head;

            /* 4. Move the head to point to new Node */
            head = new_node;
        }

        /* Function to print linked list */
        void printList()
        {
            Node temp = head;
            while (temp != null)
            {
                System.out.print(temp.data+" ");
                temp = temp.next;
            }
            System.out.println();
        }

        /* Driver program to test above functions */
        public static void main(String args[])
        {
            LinkedList llist = new LinkedList();

        /* Constructed Linked List is 1->2->3->4->5->6->7->
        8->8->9->null */
            llist.push(0);
            llist.push(1);
            llist.push(0);
            llist.push(2);
            llist.push(1);
            llist.push(1);
            llist.push(2);
```

```
            llist.push(1);
            llist.push(2);

            System.out.println("Linked List before sorting");
            llist.printList();

            llist.sortList();

            System.out.println("Linked List after sorting");
            llist.printList();
        }
    }
```