DataStructures and algorithms

Lab 6

Name:Pranjal bajaj

Cms:023-21-0022

Task 1

public class recurssion {

    public static int binarySearch(int[] arr, int low, int high, int target) {

        if (low > high) {

            return -1;

        }

        int mid = low + high / 2;

        if (arr[mid] == target) {

            return mid;

        }

        else if (arr[mid] > target) {

            return binarySearch(arr, low, mid - 1, target);

        }

        else {

            return binarySearch(arr, mid + 1, high, target);

        }

    }

    public static void main(String[] args) {

        int[] arr = {20,30,39,40,49,50,59,60,69,70};

        int target = 70;

        int result = binarySearch(arr, 0, arr.length - 1, target);

        if (result != -1) {

            System.out.println("Element " + target + " is present at index " + result + ".");

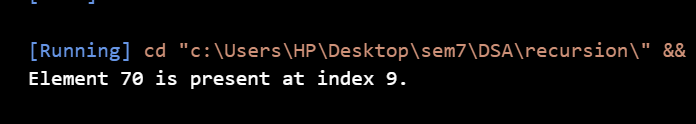
        } else {

            System.out.println("Element " + target + " is not found in the array.");

        }

    }

}



Task 2

public class recurssion {

    public static int factorial(int num) {

        if(num == 1 || num ==0){

            return 1;

        }

        else{

            return num \* factorial(num-1);

        }

    }

    public static void main(String[] args) {

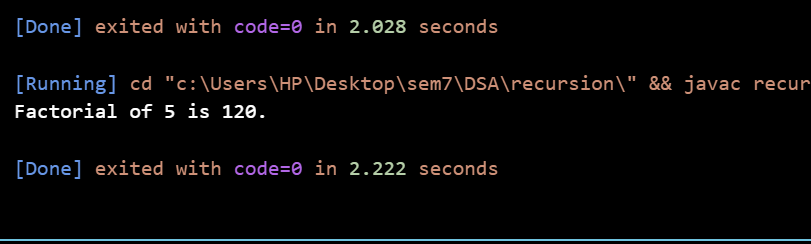
        int num = 5;

        int result = factorial(num);

            System.out.println("Factorial of "+num +" is " + result + ".");

    }

}



Task 3

public class recurssion {

    public static int fabonacci(int num) {

        if(num == 1| num ==0){

            return 1;

        }

        else if( num ==0){

            return 0;

        }

        else{

            return fabonacci(num-1)+fabonacci(num-2);

        }

    }

     void printFibonacci(int num) {

        System.out.print("Fibonacci Series: ");

        for (int i = 0; i < num; i++) {

            int f =fabonacci(i);

            System.out.print(f + " ");

        }

        System.out.println();

    }

    public static void main(String[] args) {

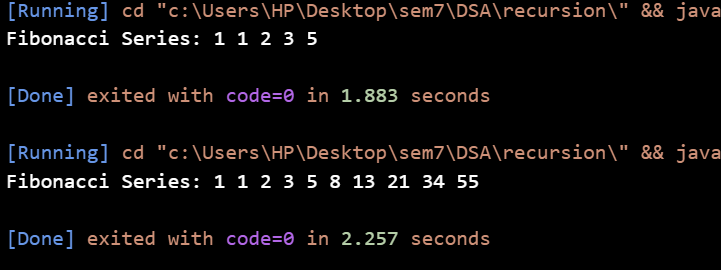
        recurssion rec = new recurssion();

        int num = 10;

        rec.printFibonacci( num);

    }

}



Task 4

import java.util.Stack;

public class recurssion {

    public static void insertAtBottom(Stack<Integer> stack, int element) {

        if (stack.isEmpty()) {

            stack.push(element);

            return;

        }

        int top = stack.pop();

        insertAtBottom(stack, element);

        stack.push(top);

    }

    public static void reverseStack(Stack<Integer> stack) {

        if (stack.isEmpty()) {

            return;

        }

        int top = stack.pop();

        reverseStack(stack);

        insertAtBottom(stack, top);

    }

    public static void printStack(Stack<Integer> stack) {

        for (int element : stack) {

            System.out.print(element + " ");

        }

        System.out.println();

    }

    public static void main(String[] args) {

        Stack<Integer> stack = new Stack<>();

        stack.push(1);

        stack.push(2);

        stack.push(3);

        stack.push(4);

        stack.push(5);

        System.out.println("Original Stack:");

        printStack(stack);

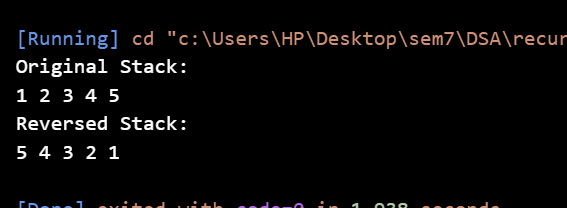
        reverseStack(stack);

        System.out.println("Reversed Stack:");

        printStack(stack);

    }

}



Task 5

public class recurssion {

    class Node{

        Node next;

        int data;

        Node(int data){

            this.next = null;

            this.data= data;

        }

    }

    Node head;

    public static void LinkedList(Node node ) {

        if (node == null) {

            return;

        }

        System.out.print(node.data + " ");

        LinkedList(node.next);

    }

     void create(int data) {

        Node newNode = new Node(data);

        if (head == null) {

            head = newNode;

        } else {

            Node temp = head;

            while (temp.next != null) {

                temp = temp.next;

            }

            temp.next = newNode;

        }

     }

    public static void main(String[] args) {

        recurssion rec = new recurssion();

        rec.create(20);

        rec.create(30);

        rec.create(40);

        rec.create(50);

        rec.create(60);

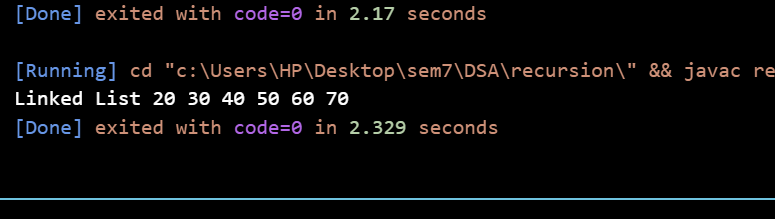
        rec.create(70);

        System.out.print("Linked List " );

         rec.LinkedList(rec.head);

    }

}



Task 6

class Node {

    int data;

    Node next;

    Node(int data) {

        this.data = data;

        this.next = null;

    }

}

class SinglyLinkedList {

    Node head;

    public void printReverse(Node node) {

        if (node == null) {

            return;

        }

        printReverse(node.next);

        System.out.println(node.data);

    }

    public void add(int data) {

        Node newNode = new Node(data);

        if (head == null) {

            head = newNode;

        } else {

            Node current = head;

            while (current.next != null) {

                current = current.next;

            }

            current.next = newNode;

        }

    }

}

public class recurssion {

    public static void main(String[] args) {

        SinglyLinkedList list = new SinglyLinkedList();

        list.add(1);

        list.add(2);

        list.add(3);

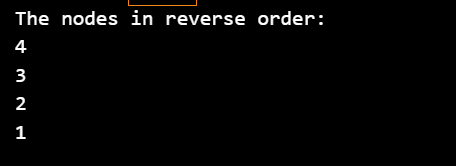
        list.add(4);

        System.out.println("The nodes in reverse order:");

        list.printReverse(list.head);

    }

}



Task 7

public class recurssion {

    public static boolean isPalindrome(String s, int left, int right) {

        if (left >= right) {

            return true;

        }

        if (Character.toLowerCase(s.charAt(left)) == Character.toLowerCase(s.charAt(right))) {

            return isPalindrome(s, left + 1, right - 1);

        }

        return false;

    }

    public static void main(String[] args) {

        String s1 = "racecar";

        String s2 = "cake";

        String s3 = "deified";

        if (isPalindrome(s1, 0, s1.length() - 1)) {

            System.out.println(s1 + " is a palindrome.");

        } else {

            System.out.println(s1 + " is not a palindrome.");

        }

        if (isPalindrome(s2, 0, s2.length() - 1)) {

            System.out.println(s2 + " is a palindrome.");

        } else {

            System.out.println(s2 + " is not a palindrome.");

        }

        if (isPalindrome(s3, 0, s1.length() - 1)) {

            System.out.println(s3 + " is a palindrome.");

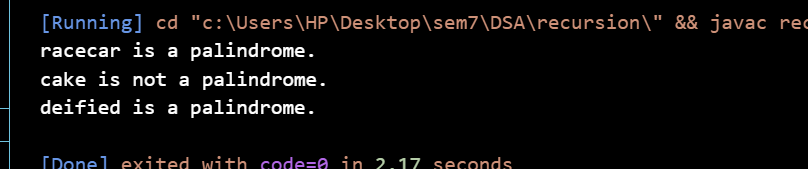
        } else {

            System.out.println(s3 + " is not a palindrome.");

        }

    }

}



Task 8

public class recurssion {

    public static void solveHanoi(int n, char source, char destination, char auxiliary) {

        if (n == 1) {

            System.out.println("Move disk 1 from " + source + " to " + destination);

            return;

        }

        solveHanoi(n - 1, source, auxiliary, destination);

        System.out.println("Move disk " + n + " from " + source + " to " + destination);

               solveHanoi(n - 1, auxiliary, destination, source);

    }

    public static void main(String[] args) {

        int n = 3

        solveHanoi(n, 'A', 'C', 'B');

    }

}

