Name: **ZOHAIB HASSAN SOOMRO**

RollNo#: **19SW42**

Subject: **DSA**



Lab#7

**Task#1,2,3,4:**

**ArrayQueue:**

**import java.util.Arrays;**

**public class ArrayQueue implements Queue{**

**int size;**

**Object array[];**

**public ArrayQueue(){**

**this(10);**

**}**

**public ArrayQueue(int capacity){**

**array=new Object[capacity];**

**}**

**@Override**

**public void add(Object object){**

**if(size==array.length) resizeArray();**

**array[size++]= object;**

**}**

**@Override**

**public Object first(){**

**if(size==0) throw new IllegalStateException("Queue is Empty!");**

**return array[0];**

**}**

**@Override**

**public Object remove(){**

**if(size==0) throw new IllegalStateException("Queue is Empty!");**

**Object obj=array[0];**

**System.arraycopy(array,1,array,0,--size);**

**array[size]=null;**

**return obj;**

**}**

**@Override**

**public int size(){**

**return size;**

**}**

**public String toString(){**

**if(size==0) return "[]";**

**String buff= "["+array[0];**

**for (int i=1;i<size;i++)**

**buff+=","+array[i];**

**return buff+"]";**

**}**

**public boolean equals(ArrayQueue queue){**

**if(this.size()!=queue.size()) return false;**

**for (int i=0;i<size();i++) {**

**if(!array[i].equals(queue.array[i]))**

**return false;**

**}**

**return true;**

**}**

**public ArrayQueue clone(){**

**if(size==0) return new ArrayQueue();**

**ArrayQueue queue = new ArrayQueue();**

**for (int i=0;i<size();i++)**

**queue.add(array[i]);**

**return queue;**

**}**

**public Object removeBottomElement(){**

**if(size==0) throw new IllegalStateException("Queue is Empty!");**

**Object obj=array[--size];**

**array[size]=null;**

**return obj;**

**}**

**public Stack toStack(){**

**if(size==0) throw new IllegalStateException("Queue is Empty!");**

**Stack stack= new ArrayStack(size);**

**for (int i=0;i<size();i++)**

**stack.push(array[i]);**

**return stack;**

**}**

**public void reverse(){**

**if(size==0) return;**

**for (int i=0;i<size/2;i++) {**

**Object temp=array[i];**

**array[i]=array[size-i-1];**

**array[size-i-1]=temp;**

**}**

**}**

**public LinkedQueue toLinkedQueue(){**

**if(size==0) return new LinkedQueue();**

**LinkedQueue queue= new LinkedQueue();**

**for (int i=0;i<size();i++)**

**queue.add(array[i]);**

**return queue;**

**}**

**public void resizeArray(){**

**array= Arrays.copyOf(array,array.length\*2);**

**}**

**public static void main(String[] args) {**

**ArrayQueue queue= new ArrayQueue();**

**queue.add(123);**

**queue.add(23);**

**queue.add(56);**

**queue.add("Hello!");**

**queue.add("Hi!");**

**ArrayQueue queue2= new ArrayQueue();**

**queue2.add(45);**

**queue2.add(656);**

**System.out.println(queue.toString());**

**System.out.println(queue.equals(queue2));**

**System.out.println(queue.clone().toString());**

**System.out.println(queue.removeBottomElement());**

**System.out.println(queue.toStack());**

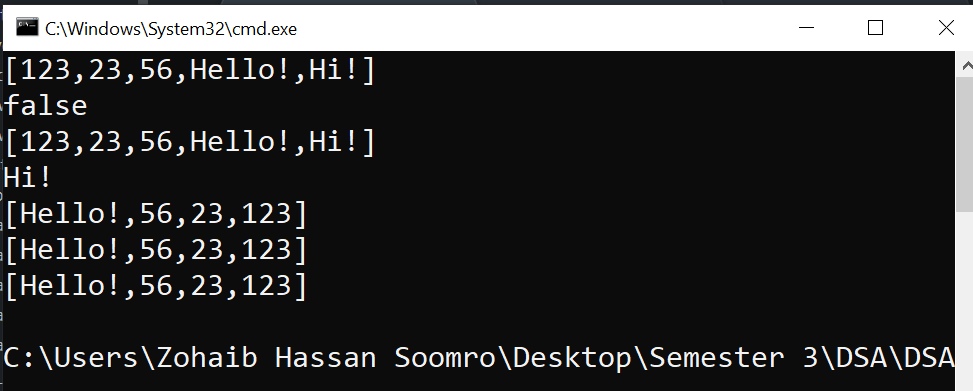
**queue.reverse();**

**System.out.println(queue);**

**System.out.println(queue.toLinkedQueue().toString());**

**}**

**}**



**LinkedQueue:**

**public class LinkedQueue implements Queue{**

**Node start=new Node(null);**

**int size;**

**private class Node{**

**private Object data;**

**private Node prev,next;**

**Node(Object data){**

**this.data=data;**

**prev=next=this;**

**}**

**Node(Object data,Node p, Node n){**

**this.data=data;**

**prev=p;**

**next=n;**

**}**

**}**

**@Override**

**public void add(Object object){**

**start.prev=start.prev.next=new Node(object,start.prev,start);**

**size++;**

**}**

**@Override**

**public Object first(){**

**if(size==0) throw new IllegalStateException("Queue is Empty!");**

**return start.next.data;**

**}**

**@Override**

**public Object remove(){**

**if(size==0) throw new IllegalStateException("Queue is Empty!");**

**Object obj=start.next.data;**

**--size;**

**start.next=start.next.next;**

**start.next.prev=start;**

**return obj;**

**}**

**@Override**

**public int size(){**

**return size;**

**}**

**public String toString(){**

**if(size==0) return "[]";**

**String buff= "[";**

**for (Node p=start.next;p!=start;p=p.next)**

**buff+=p.data+",";**

**return buff+"\b]";**

**}**

**public boolean equals(LinkedQueue queue){**

**if(this.size()!=queue.size()) return false;**

**for (Node p=start.next,q=queue.start.next;p!=start;p=p.next,q=q.next) {**

**if(!p.data.equals(q.data))**

**return false;**

**}**

**return true;**

**}**

**public LinkedQueue clone(){**

**if(size==0) return new LinkedQueue();**

**LinkedQueue queue = new LinkedQueue();**

**for (Node p=start.next;p!=start;p=p.next)**

**queue.add(p.data);**

**return queue;**

**}**

**public Object removeBottomElement(){**

**if(size==0) throw new IllegalStateException("Queue is Empty!");**

**Object obj=start.prev.data;**

**start.prev=start.prev.prev;**

**start.prev.next=start;**

**--size;**

**return obj;**

**}**

**public Stack toStack(){**

**if(size==0) throw new IllegalStateException("Queue is Empty!");**

**Stack stack= new ArrayStack(size);**

**for (Node p=start.next;p!=start;p=p.next)**

**stack.push(p.data);**

**return stack;**

**}**

**public void reverse(){**

**Node p=start,temp;**

**do{**

**temp=p.next;**

**p.next=p.prev;**

**p.prev=temp;**

**p=p.prev;**

**}while(p!=start);**

**}**

**public ArrayQueue toArrayQueue(){**

**if(size==0) return new ArrayQueue(0);**

**ArrayQueue queue= new ArrayQueue();**

**for (Node p=start.next;p!=start;p=p.next)**

**queue.add(p.data);**

**return queue;**

**}**

**public Object[] toArray(){**

**if(size==0) return new Object[0];**

**Object array[]= new Object[size];**

**int count=0;**

**for (Node p=start.next;p!=start;p=p.next)**

**array[count++]=p.data;**

**return array;**

**}**

**public static void main(String[] args) {**

**LinkedQueue queue= new LinkedQueue();**

**queue.add(123);**

**queue.add(23);**

**queue.add(56);**

**queue.add("Hello!");**

**System.out.println(queue.toString());**

**LinkedQueue queue2= new LinkedQueue();**

**queue2.add(45);**

**queue2.add(656);**

**ystem.out.println(queue);**

**System.out.println(queue.toString());**

**System.out.println(queue.equals(queue2));**

**System.out.println(queue.clone().toString());**

**System.out.println(queue.removeBottomElement());**

**System.out.println(queue.toStack());**

**queue.reverse();**

**System.out.println(queue);**

**System.out.println(queue.toArrayQueue().toString());**

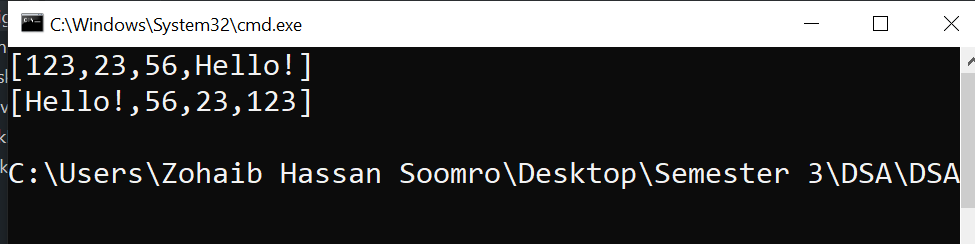
**Object array[]=queue.toArray();**

**for (int i=0;i<array.length;i++)**

**System.out.print(array[i]+" ");**

**}**

**}**



**Task#5:**

**DeQueue:**

**public class DeQueue{**

**Node start=new Node(null);**

**int size;**

**private class Node{**

**private Object data;**

**private Node prev,next;**

**Node(Object data){**

**this.data=data;**

**prev=next=this;**

**}**

**Node(Object data,Node p, Node n){**

**this.data=data;**

**prev=p;**

**next=n;**

**}**

**}**

**public void addAtLast(Object object){**

**start.prev=start.prev.next=new Node(object,start.prev,start);**

**size++;**

**}**

**public void addAtFirst(Object object){**

**start.next=start.next.prev=new Node(object,start,start.next);**

**size++;**

**}**

**public Object first(){**

**if(size==0) throw new IllegalStateException("Queue is Empty!");**

**return start.next.data;**

**}**

**public Object last(){**

**if(size==0) throw new IllegalStateException("Queue is Empty!");**

**return start.prev.data;**

**}**

**public Object removeFirst(){**

**if(size==0) throw new IllegalStateException("Queue is Empty!");**

**Object obj=start.next.data;**

**--size;**

**start.next=start.next.next;**

**start.next.prev=start;**

**return obj;**

**}**

**public Object removeLast(){**

**if(size==0) throw new IllegalStateException("Queue is Empty!");**

**Object obj=start.prev.data;**

**start.prev=start.prev.prev;**

**start.prev.next=start;**

**--size;**

**return obj;**

**}**

**public int size(){**

**return size;**

**}**

**public String toString(){**

**if(size==0) return "[]";**

**String buff= "[";**

**for (Node p=start.next;p!=start;p=p.next)**

**buff+=p.data+",";**

**return buff+"\b]";**

**}**

**public static void main(String[] args) {**

**DeQueue queue= new DeQueue();**

**queue.addAtLast(123);**

**queue.addAtLast(23);**

**queue.addAtFirst(56);**

**System.out.println(queue);**

**queue.addAtLast("Hello!");**

**System.out.println(queue);**

**System.out.println(queue.removeLast());**

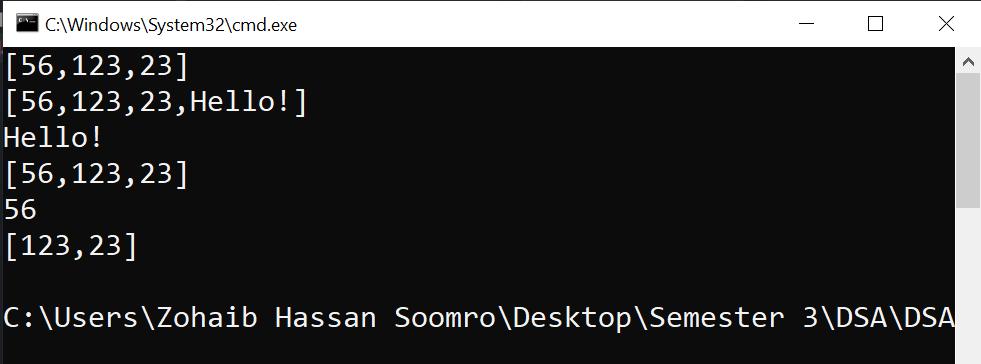
**System.out.println(queue);**

**System.out.println(queue.removeFirst());**

**System.out.println(queue);**

**}**

**}**



Lab#8

Map interface:

public interface Map{

Object putLinearly(Object key,Object value);

Object putQuadraticly(Object key,Object value);

Object get(Object key);

Object remove(Object key);

int size();

}

**Task#1:**

Quadratic Probing:

public class HashTable implements Map{

private Entry entries[];

int size,used;

float loadFactor;

private final Entry NIL= new Entry(null,null);

HashTable(int capacity,float loadFactor){

entries=new Entry[capacity];

this.loadFactor=loadFactor;

}

HashTable(int capacity){

this(capacity,0.75F);

}

HashTable(){

this(101);

}private class Entry{

Object key,value;

Entry(Object key,Object value){

this.key=key;

this.value=value;

}

}

public int hash(Object key){

if(key==null) throw new NullPointerException();

return (key.hashCode() & 0x7FFFFFFF) % entries.length;

}

public Object putLinearly(Object key,Object value){

if(used>loadFactor\*entries.length) reHash();

int h=hash(key);

for (int i=0;i<entries.length;i++) {

int j= nextProbe(h,i);

if (entries[j]==null) {

entries[j]=new Entry(key,value);

System.out.println(i+" Collisions occured for (key,value): ("+key+","+value+")");

size++;

used++;

return null;

}

if(entries[j]==NIL) continue;

if(entries[j].key.equals(key)){

Object oldValue=entries[j].value;

entries[j].value=value;

System.out.println(i+" Collisions occured for (key,value): ("+key+","+value+")");

return oldValue;

}

}

throw new IllegalStateException("Table Overflow!");

}

public Object putQuadraticly(Object key,Object value){

if(used>loadFactor\*entries.length) reHash();

int h=hash(key);

for (int i=0;i<entries.length;i++) {

int j= nextProbe(h,i\*i); //quadratic probing

if (entries[j]==null) {

entries[j]=new Entry(key,value);

System.out.println(i+" Collisions occured for (key,value): ("+key+","+value+")");

size++;

used++;

return null;

}

if(entries[j]==NIL) continue;

if(entries[j].key.equals(key)){

Object oldValue=entries[j].value;

entries[j].value=value;

System.out.println(i+" Collisions occured for (key,value): ("+key+","+value+")+ Old Value("+oldValue+")Updated");

return oldValue;

}

}

throw new IllegalStateException("Table Overflow!");

}

public int nextProbe(int h,int i){

return (h+i)%entries.length;

}

public Object get(Object key){

int h=hash(key);

for (int i=0;i<entries.length;i++) {

int j= nextProbe(h,i);

if (entries[j]==null)break;

if(entries[j]==NIL) continue;

if(entries[j].key.equals(key))

return entries[j].value;

}

throw new IllegalStateException("Value Not Found!");

}

public int size(){

return size;

}

public Object remove(Object key){

int h=hash(key);

for (int i=0;i<entries.length;i++) {

int j= nextProbe(h,i);

if (entries[j]==null)break;

if(entries[j]==NIL) continue;

if(entries[j].key.equals(key)){

Object oldValue=entries[j].value;

entries[j]=NIL;

--size;

return oldValue;

}

}

throw new IllegalStateException("Value Not Found!");

}

public void reHash(){

Entry oldEntries[] = entries;

entries= new Entry[2\*oldEntries.length+1];

for (int k=0;k<oldEntries.length;k++) {

if(oldEntries[k]==null || oldEntries[k]==NIL)

continue;

int h= hash(oldEntries[k].key);

for (int i=0;i<entries.length;i++) {

int j=nextProbe(h,i);

if (entries[j]==null) {

entries[j]=oldEntries[k];

break;

}

}

}

used=size;

}

public int capacity(){

return entries.length;

}

public static void main(String[] args) {

HashTable table=new HashTable(7,0.5F);

table.putQuadraticly("PK","PAKISTAN");

table.putQuadraticly("IN","INDIA");

table.putQuadraticly("SK","SOUTH KOREA");

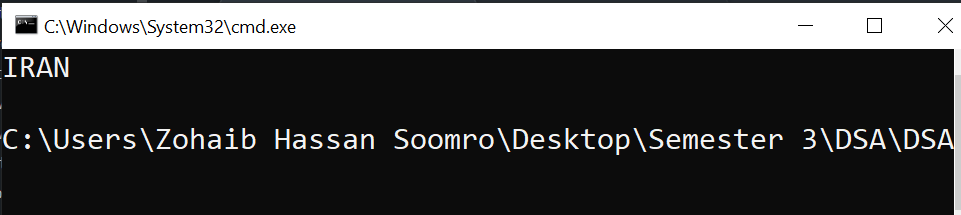
table.putQuadraticly("ID","INDONESIA");

table.putQuadraticly("IN","IRAN"); //updating value of ke "IN"

System.out.println(table.get("IN"));

}

}



**Task#2:**

public class CompareProbing{

public static void main(String[] args) {

HashTable table=new HashTable(7,0.5F); //for linear probing

HashTable table2=new HashTable(7,0.5F); //for quadratic probing

table.putLinearly("PK","PAKISTAN");

table2.putQuadraticly("PK","PAKISTAN");

table.putLinearly("1","INDIA");

table2.putQuadraticly("IN","INDIA");

table.putLinearly("PT","SOUTH KOREA");

table2.putQuadraticly("PT","SOUTH KOREA");

table.putLinearly("JP","IRAN");

table2.putQuadraticly("JP","IRAN");

table.putLinearly("AT","MALAYSIA");

table2.putQuadraticly("AT","MALAYSIA");

table.putLinearly("1","FRANCE");

table2.putQuadraticly("PK","FRANCE");

table.putLinearly("1","USA");

table2.putQuadraticly("PK","USA");

table.putLinearly("PK","UK");

table2.putQuadraticly("PK","UK");

table.putLinearly("2","ENGLAND");

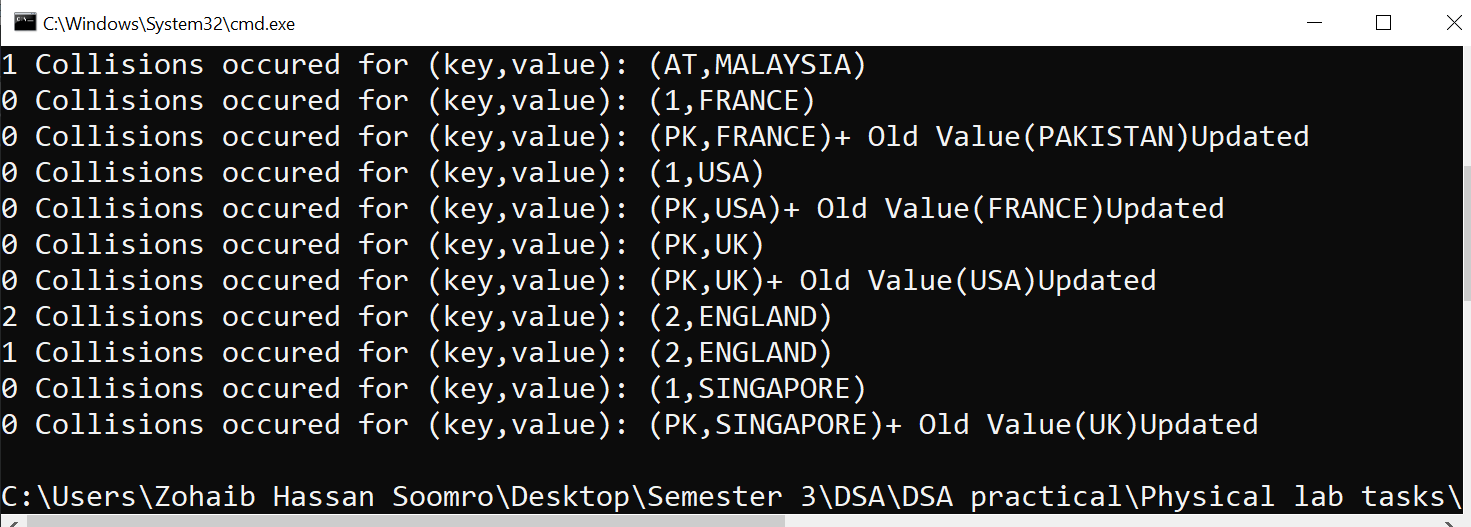
table2.putQuadraticly("2","ENGLAND");

table.putLinearly("1","SINGAPORE");

table2.putQuadraticly("PK","SINGAPORE");

}

}



**Task#3:**

import java.util.\*;

public class JavaUtilHashTable{

public static void main(String[] args) {

Hashtable<Object,Object> table= new Hashtable<Object,Object>();

table.put("PK","Pakistan"); //#1

table.put("2","India");

table.put(3,56);

table.put(4,150);

System.out.println(table.isEmpty()); //#2

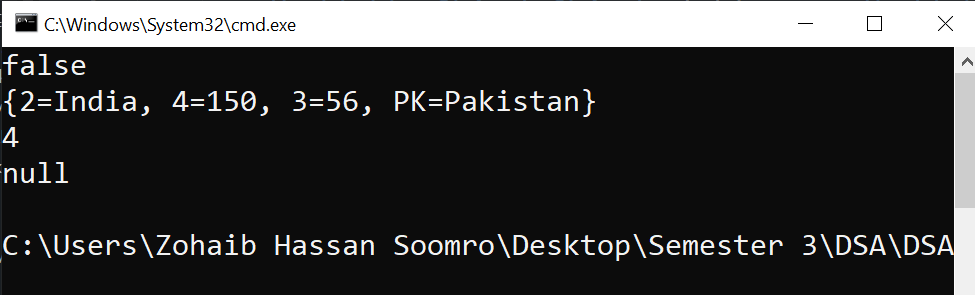
System.out.println(table.toString()); //#3

System.out.println(table.size()); //#4

System.out.println(table.get("4")); //#5

}

}



Lab#9 & 10

Task#1:

**BubbleSort:**

public class BubbleSort{

public static void main(String[] args) {

int array[]={3,2,6,9,32,1,4};

for (int g:array) {

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

}

System.out.println();

bubbleSort(array);

for (int g:array) {

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

}

}

public static void bubbleSort(int array[]){

int size=array.length;

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

for (int j=0;j<size-i-1;j++) {

if(array[j]>array[j+1]){

int temp=array[j];

array[j]=array[j+1];

array[j+1]=temp;

}

}

}

}

}

**InsertionSort:**

public class InsertionSort{

public static void insertionSort(int array[]){

for (int i=1;i<array.length;i++) {

int temp=array[i];

int j=i-1;

while(j>=0){

if(array[j]>temp){

array[j+1]=array[j];

}

else

break;

j=j-1;

}

array[j+1]=temp;

}

}

public static void main(String[] args) {

int array[]={2,5,1,3,7,0,4};

System.out.print("Before Sorting: ");

for (int g:array) {

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

}

insertionSort(array);

System.out.print("\nAfter Sorting: ");

for (int g:array) {

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

}

}

}

Task#2:

Timing in nano seconds:

import java.util.Random;

public class TimingInSort{

public static void main(String[] args) {

int array[]= new int[100000];

Random rand=new Random();

for (int i=0;i<array.length;i++)

array[i]=rand.nextInt(100000);

long t1,t2;

t1=System.nanoTime();

bubbleSort(array);

t2=System.nanoTime();

System.out.println("\n\nTiming in bubble sort: "+(t2-t1));

t1=System.nanoTime();

insertionSort(array);

t2=System.nanoTime();

System.out.println("\n\nTiming in insertion sort: "+(t2-t1));

}

public static void insertionSort(int array[]){

for (int i=1;i<array.length;i++) {

int temp=array[i];

int j=i-1;

while(j>=0){

if(array[j]<temp){

array[j+1]=array[j];

}

else

break;

j=j-1;

}

array[j+1]=temp;

}

}

public static void bubbleSort(int array[]){

int size=array.length;

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

for (int j=0;j<size-i-1;j++) {

if(array[j]<array[j+1]){

int temp=array[j];

array[j]=array[j+1];

array[j+1]=temp;

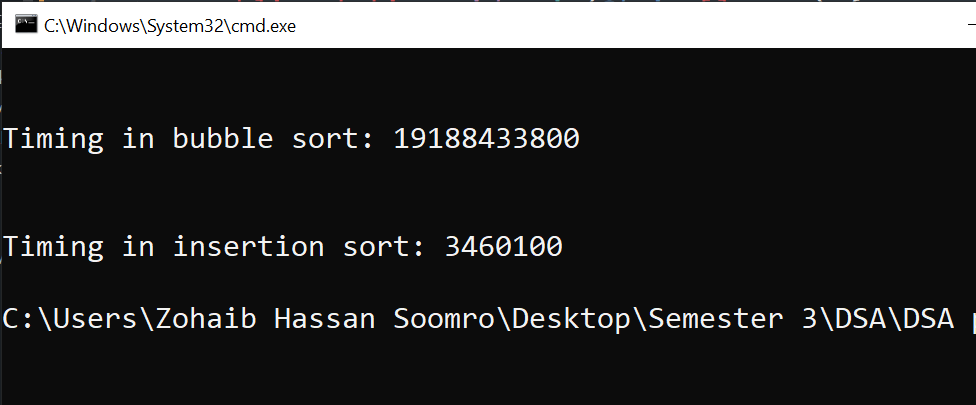
}

}

}

}

}



Task#3:

**BubbleSort:**

public class BubbleSort{

public static void main(String[] args) {

int array[]={3,2,6,9,32,1,4};

for (int g:array) {

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

}

System.out.println();

bubbleSort(array);

for (int g:array) {

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

}

}

public static void bubbleSort(int array[]){

int size=array.length;

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

for (int j=0;j<size-i-1;j++) {

if(array[j]<array[j+1]){

int temp=array[j];

array[j]=array[j+1];

array[j+1]=temp;

}

}

}

}

}

**InsertionSort:**

public class InsertionSort{

public static void insertionSort(int array[]){

for (int i=1;i<array.length;i++) {

int temp=array[i];

int j=i-1;

while(j>=0){

if(array[j]<temp){

array[j+1]=array[j];

}

else

break;

j=j-1;

}

array[j+1]=temp;

}

}

public static void main(String[] args) {

int array[]={2,5,1,3,7,0,4};

System.out.print("Before Sorting: ");

for (int g:array) {

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

}

insertionSort(array);

System.out.print("\nAfter Sorting: ");

for (int g:array) {

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

}

}

}

Lab#11

Task#1:

Fibonacci series:

public class FibonacciSeries{

public static int fibonacciSeries(int n){

if(n==0)

return 0;

if(n<3)

return 1;

return fibonacciSeries(n-1)+fibonacciSeries(n-2);

}

public static void main(String[] args) {

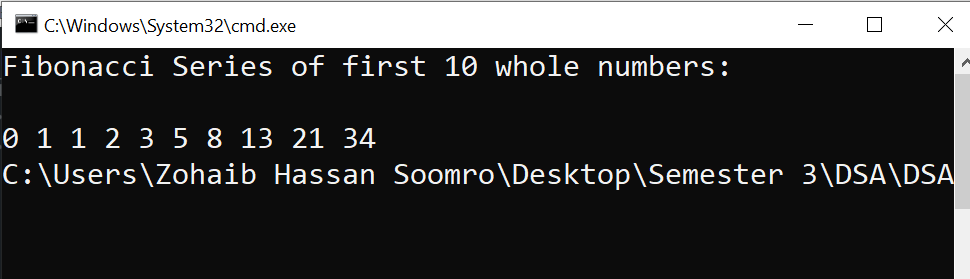
System.out.println("Fibonacci Series of first 10 whole numbers: \n");

for (int i=0;i<10;i++)

System.out.print(fibonacciSeries(i)+" ");

}

}



Task#2:

Tower Of Hanoi:

public class TowerOfHanoi{

public static void towerOfHanoi(int n,char src,char dst,char aux){

if(n==1){

System.out.println("Disk "+n+" moved from "+src+" to "+dst);

return;

}

towerOfHanoi(n-1,src,aux,dst);

System.out.println("Disk "+n+" moved from "+src+" to "+dst);

towerOfHanoi(n-1,aux,dst,src);

}

public static void main(String[] args) {

java.util.Scanner scan= new java.util.Scanner(System.in);

System.out.print("Enter no of disks: ");

int n=scan.nextInt();

towerOfHanoi(n,'A','B','C');

}

}

