ADVANCED DATA STRUCTURES AND ALGORITHMS
(R22D5881)

LABORATORY MANUAL

M.TECH
(I YEAR – I SEM)
(2022-23)

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956

(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)

Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956
(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

CERTIFICATE

Department of : .....

Certified that this is the bonafide record of the work done by

Mr./Miss **.....Reg.No. .....**

of M.Tech **(.....) .....**Year **.....**Semester for

Academic year 20.....to 20..... in the **.....**

..... Laboratory

Date : **Staff Incharge** **HOD**

Internal Examiner **External Examiner**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vision

- To acknowledge quality education and instill high patterns of discipline making the students technologically superior and ethically strong which involves the improvement in the quality of life in human race.

Mission

- To achieve and impart holistic technical education using the best of infrastructure, outstanding technical and teaching expertise to establish the students into competent and confident engineers.
- Evolving the center of excellence through creative and innovative teaching learning practices for promoting academic achievement to produce internationally accepted competitive and world class professionals.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO1. To provide an environment that gives hands on experience in Modeling, Designing, Implementing, and evaluating various software development concepts, processes and products.

PEO2. To afford graduates with both fundamental and advanced knowledge which prepares them to posses integrated and ethical behavior as an individual, team member and a leader to handle diverse career paths.

PEO3. To produce high quality graduates to design and implement solutions for rapidly changing computing and information system problems and to encourage lifelong learning to adapt innovation.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1: DEVELOPMENT AND ASSESSMENT SKILLS: Ability to Design, Develop and Analyze software development tools, processes and systems using formal methods in applying problem solving skills and be employable in product or service oriented Industry.

PSO2: RESEARCH & DEVELOPMENT & INNOVATION SKILLS: Ability to take up effectively the challenges in higher Studies, Research & Development, and Entrepreneurship in the modern high speed computing environment.

PROGRAM OUTCOMES (POs)

PO1: RESEARCH SKILLS
An ability to independently carry out research / investigation and development work to solve practical problems.

PO2: SOFT SKILLS
Ability to write and present a substantial technical report / document.

PO3: SCHOLARSHIP OF KNOWLEDE
Students should be able to demonstrate a degree of mastery over the area as per the specialization of the program at a level higher than the relevant bachelor program.

PO4: PROBLEM SOLVING
Apply the knowledge of engineering principles to develop software systems, products and processes thus to solve real world multifaceted problems.

PO5: COLLABORATIVE AND MULTIDISCIPLINARY WORK
Posses knowledge and understand group dynamics, collaborate and contribute in the design, development and conducting experiments, procedures and technical skills necessary for multidisciplinary engineering exploration to solve societal problems and environmental contexts for sustainable development.

PO6: ETHICAL PRACTICES AND SOCIAL RESPONSIBILITY
Recognize the need to engage in self-governing and life-long learning by making use of professional and ethical principles.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
Maisammaguda, Dhulapally Post, Via Hakimpet, Secunderabad – 500100
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.
3. Student should enter into the laboratory with:

a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.
c. Proper Dress code and Identity card.

4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions.Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out ; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Head of the Department **Principal**

Objectives:

- The fundamental design, analysis, and implementation of basic data structures.
- Basic concepts in the specification and analysis of programs.
- Principles for good program design, especially the uses of data abstraction.

SNo	Name of the Experiment	Page no	Date	Signature
1	Write Java programs that use both recursive and non-recursive functions for implementing the following searching methods: a) Linear search b) Binary search			
2	Write Java programs to implement the following using arrays and linked lists			
3	Write Java programs to implement the following using an array. a) Stack ADT b) Queue ADT			
4	Write a Java program that reads an infix expression and converts the expression to postfix form. (Use stack ADT).			
5	Write a Java program that uses both a stack and a queue to test whether the given string is a palindrome or not.			
6	Write Java programs to implement the following using a singly linked list. a) Stack ADT b) Queue ADT			
7	Write a Java program to perform the following operations: a) Construct a binary search tree of elements. b) Search for a key element in the above binary search tree. c) Delete an element from the above binary search tree.			
8	Write a Java program to implement all the functions of a dictionary (ADT) using Hashing.			
9	Write Java programs that use recursive and non-recursive functions to traverse the given binary tree in a)Preorder b) Inorder c) Postorder			
10	Write Java programs for the implementation of bfs and dfs for a given graph.			
11	Write Java programs for implementing the following sorting methods: a) Bubble sort b) Insertion sort c) Quick sort d) Merge sort e) Heap sort f) Radix sort g) Binary tree sort			
12	Write a Java program to perform the following operations: a) Insertion into a B-tree b) Searching in a B-tree			

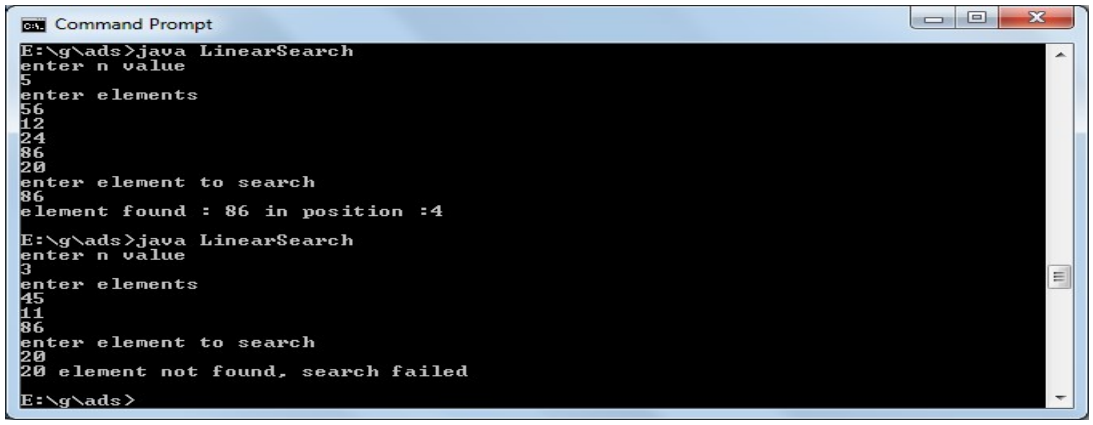
1. Write Java programs that use both recursive and non-recursive functions for implementing the following searching methods:

(a) Linear search (b) Binary search

1(a) Linear Search using non-recursive function

```
import java.io.*;
class LinearSearch
{
	public static void main(String args[]) throws IOException
	{
		int count=0;
		BufferedReader br=new BufferedReader(new
 InputStreamReader(System.in));
		System.out.println("enter n value");
		int n=Integer.parseInt(br.readLine());
		int arr[]=new int[n];
		System.out.println("enter elements");
		for(int i=0;i<n;i++)
		{
			arr[i]=Integer.parseInt(br.readLine());
		}
		System.out.println("enter element to search");
		int key=Integer.parseInt(br.readLine());
		for(int i=0;i<n;i++)
		{
		if(arr[i]==key)
		System.out.println("element found : " + key + " in position :" + (i+1));
			else
				count++;
		}
		if(count==n)
			System.out.println(key + " element not found, search failed");
	}
}
```

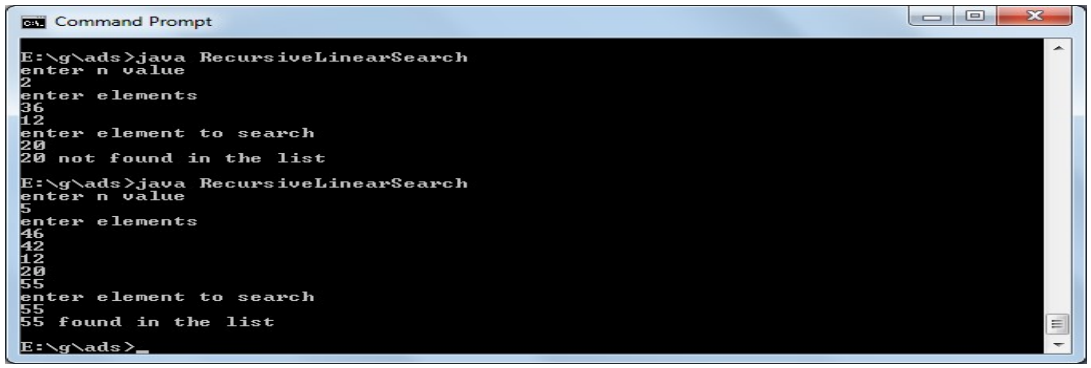
OUTPUT:



//Linear Search using recursive function

```
import java.io.*;
class RecursiveLinearSearch
{
	public static int arr[], key;
	public static void main(String args[]) throws IOException
	{
		BufferedReader br=new BufferedReader(new
					InputStreamReader(System.in));
		System.out.println("enter n value");
		int n=Integer.parseInt(br.readLine());
		arr=new int[n];
		System.out.println("enter elements");
		for(int i=0;i<n;i++)
		{
			arr[i]=Integer.parseInt(br.readLine());
		}
		System.out.println("enter element to search");
		key=Integer.parseInt(br.readLine());

		if( linearSearch(arr.length-1) )
			System.out.println(key + " found in the list" );
		else
			System.out.println(key + " not found in the list");
	}
	static boolean linearSearch(int n)
	{
		if( n < 0 ) return false;
		if(key == arr[n])

			return true;
		else
			return linearSearch(n-1);
	}
}
```

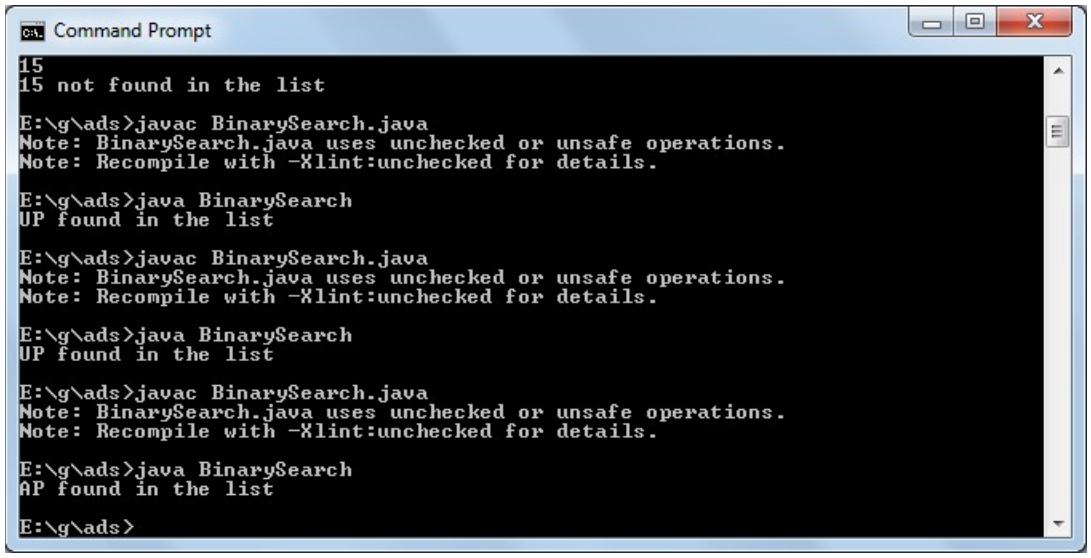
OUTPUT:



_1 (b) Binary Search using non-recursive function

```
class BinarySearch
{
static Object[] a = { "AP", "KA", "MH", "MP", "OR", "TN", "UP", "WB"};
        static Object key = "UP";
        public static void main(String args[])
        {
                if( binarySearch() )
                        System.out.println(key + " found in the list");
                else
                        System.out.println(key + " not found in the list");
        }
        static boolean binarySearch()
        {
                int c, mid, low = 0, high = a.length-1;
                while( low <= high)
                {
                        mid = (low + high)/2;
                        c = ((Comparable)key).compareTo(a[mid]);
                        if( c < 0) high = mid-1;
                        else if( c > 0) low = mid+1;
                        else return true;
                }
                return false;
        }
}
```

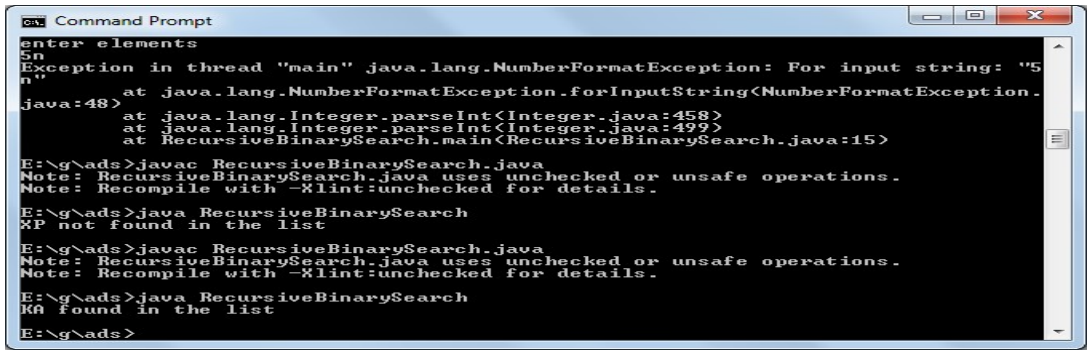
OUTPUT:



//Binary Search using recursive function

```
import java.io.*;
class RecursiveBinarySearch
{
	public static int arr[], key;
	public static void main(String args[]) throws IOException
	{
		BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
		System.out.println("enter n value");
		int n=Integer.parseInt(br.readLine());
		arr=new int[n];
		System.out.println("enter elements");
		for(int i=0;i<n;i++)
		{
			arr[i]=Integer.parseInt(br.readLine());
		}
		System.out.println("enter element to search");
		key=Integer.parseInt(br.readLine());

		if( binarySearch(0, arr.length-1) )
			System.out.println(key + " found in the list");
		else
			System.out.println(key + " not found in the list");
	}
	static boolean binarySearch(int low, int high)
	{
		if( low > high ) return false;
		int mid = (low + high)/2;
		int c = ((Comparable)key).compareTo(arr[mid]);

	if( c < 0) return binarySearch(low, mid-1);
		else if( c > 0) return binarySearch(mid+1, high);
		else return true;
	}
}
```

OUTPUT:



2. Write Java programs to implement the following using arrays and linked lists List ADT

a) List using Arrays

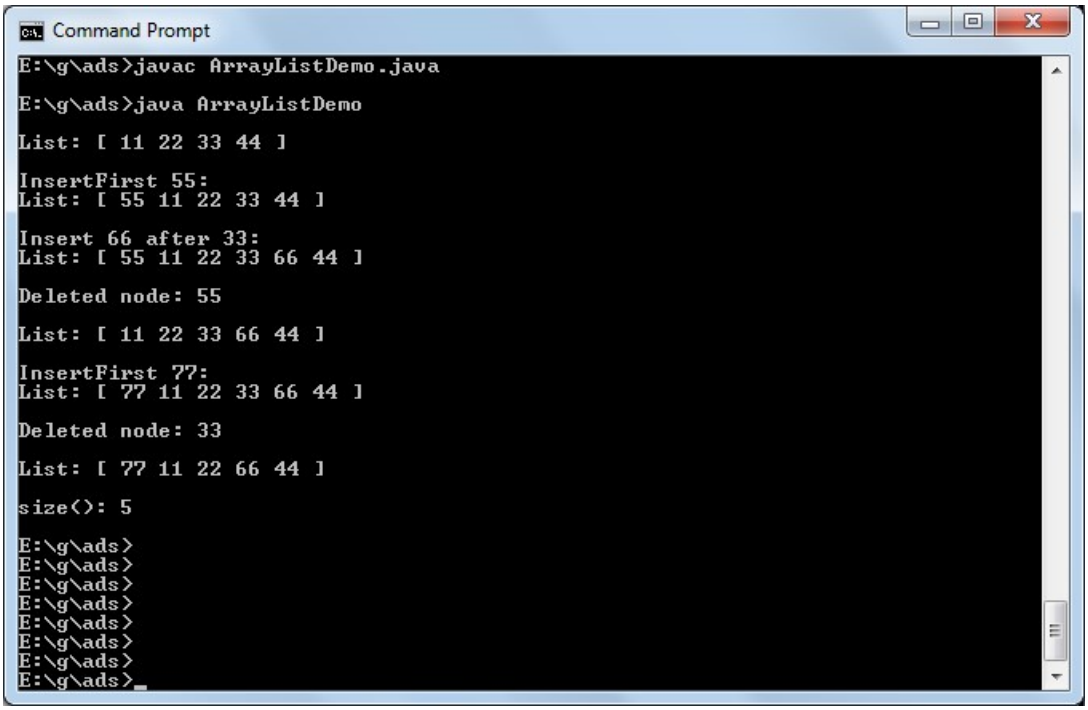
```
//List.java

public interface List
{
public void createList(int n);
public void insertFirst(Object ob);
public void insertAfter(Object ob, Object pos);
public Object deleteFirst();
public Object deleteAfter(Object pos);
public boolean isEmpty();
public int size();
}
// ArrayList.java

class ArrayList implements List
{
class Node
{
Object data;
int next;
Node(Object ob, int i) // constructor
{
data = ob;
next = i;
}
}
int MAXSIZE; // max number of nodes in the list
Node list[]; // create list array
int head, count; // count: current number of nodes in the list
ArrayList( int s) // constructor
{
 MAXSIZE = s;
list = new Node[MAXSIZE];
}

public void initializeList()
{
 for( int p = 0; p < MAXSIZE-1; p++ )
list[p] = new Node(null, p+1);
list[MAXSIZE-1] = new Node(null, -1);
}

public void createList(int n) // create 'n' nodes
{
int p;
for( p = 0; p < n; p++ )
{
list[p] = new Node(11+11*p, p+1);
count++;
}
list[p-1].next = -1; // end of the list
```

```
}

public void insertFirst(Object item)
{
if( count == MAXSIZE )
{
System.out.println("***List is FULL");
return;
}
int p = getNode();
if( p != -1 )
{
list[p].data = item;
if( isEmpty() ) list[p].next = -1;
else list[p].next = head;
head = p;
count++;
}
}
public void insertAfter(Object item, Object x)
{
if( count == MAXSIZE )
{
System.out.println("***List is FULL");
return;
}
int q = getNode(); // get the available position to insert new node
int p = find(x); // get the index (position) of the Object x
if( q != -1 )
{
 list[q].data = item;
list[q].next = list[p].next;
list[p].next = q;
count++;
}
}

        public int getNode() // returns available node index
{
 for( int p = 0; p < MAXSIZE; p++ )
if(list[p].data == null)
 return p;
return -1;
}

public int find(Object ob) // find the index (position) of the Object ob
{
int p = head;
while( p != -1)
{
 if( list[p].data == ob ) return p;
p = list[p].next; // advance to next node
}
return -1;
}
```

```
public Object deleteFirst()
{
if( isEmpty() )
{
System.out.println("List is empty: no deletion");
return null;
}
Object tmp = list[head].data;
if( list[head].next == -1 ) // if the list contains one node,
head = -1; // make list empty.
else
head = list[head].next;
count--; // update count
return tmp;
}

public Object deleteAfter(Object x)
{
 int p = find(x);
if( p == -1 || list[p].next == -1 )
{
System.out.println("No deletion");
return null;
}
int q = list[p].next;
Object tmp = list[q].data;
list[p].next = list[q].next;
count--;
return tmp;
}
public void display()
{
int p = head;
System.out.print("\nList: [ " );
while( p != -1)
{
System.out.print(list[p].data + " "); // print data
p = list[p].next; // advance to next node
}
System.out.println("]\n");//
}
public boolean isEmpty()
{
if(count == 0) return true;
else return false;
}
public int size()
{
 return count;
}
}
// ArrayListDemo.java
class ArrayListDemo
{
```

```
public static void main(String[] args)
{
ArrayList linkedList = new ArrayList(10);
linkedList.initializeList();
linkedList.createList(4); // create 4 nodes
linkedList.display(); // print the list
System.out.print("InsertFirst 55:");
linkedList.insertFirst(55);
linkedList.display();
System.out.print("Insert 66 after 33:");
linkedList.insertAfter(66, 33); // insert 66 after 33
linkedList.display();
Object item = linkedList.deleteFirst();
System.out.println("Deleted node: " + item);
linkedList.display();
System.out.print("InsertFirst 77:");
linkedList.insertFirst(77);
linkedList.display();
item = linkedList.deleteAfter(22); // delete node after node 22
System.out.println("Deleted node: " + item);
linkedList.display();
System.out.println("size(): " + linkedList.size());
}
}
```

OUTPUT:



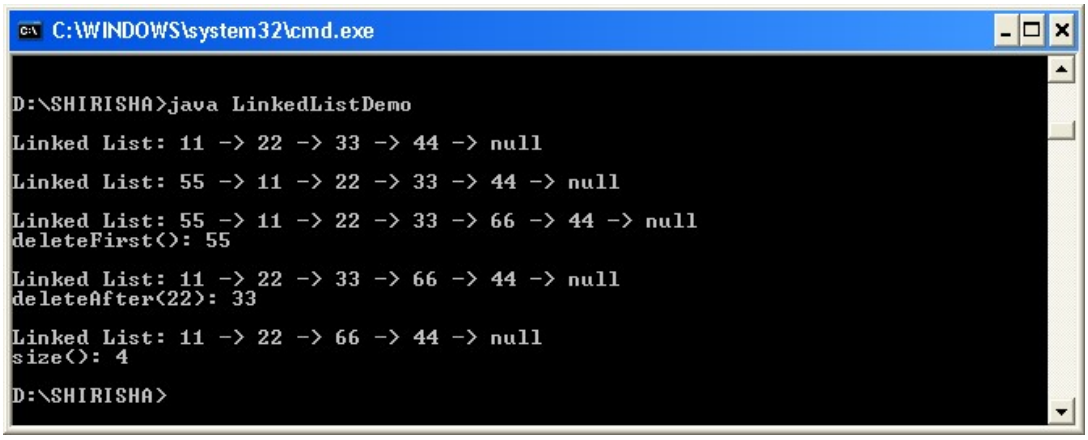
b) List Using LinkedList
//LinkedListDemo.java

```
class LinkedList implements List
{
class Node
{
Object data; // data item
Node next; // refers to next node in the list

Node( Object d ) // constructor
{
data = d;
} // 'next' is automatically set to null
}
Node head; // head refers to first node
Node p; // p refers to current node
int count; // current number of nodes
public void insertFirst(Object item) // insert at the beginning of list
{
p = new Node(item); // create new node
p.next = head; // new node refers to old head
head = p; // new head refers to new node
count++;
}
public void insertAfter(Object item,Object key)
{
p = find(key); // get "location of key item"
if( p == null )
System.out.println(key + " key is not found");
else
{
Node q = new Node(item); // create new node
q.next = p.next; // new node next refers to p.next
p.next = q; // p.next refers to new node
count++;
}
}

public Node find(Object key)
{
p = head;
while( p != null ) // start at beginning of list until end of list
{
if( p.data == key ) return p; // if found, return key address
p = p.next; // move to next node
}
return null; // if key search is unsuccessful, return null
}
public Object deleteFirst() // delete first node
{
```

```
if( isEmpty() )
{
System.out.println("List is empty: no deletion");
return null;
}
Node tmp = head; // tmp saves reference to head
head = tmp.next;
count--;
return tmp.data;
}
public Object deleteAfter(Object key) // delete node after key item
{
p = find(key); // p = “location of key node”
if( p == null )
{
System.out.println(key + " key is not found");
return null;
}
if( p.next == null ) // if(there is no node after key node)
{
System.out.println("No deletion");
return null;
                        }
else
{
 Node tmp = p.next; // save node after key node
p.next = tmp.next; // point to next of node deleted
count--;
return tmp.data; // return deleted node
}
}
public void displayList()
{
p = head; // assign mem. address of 'head' to 'p'
System.out.print("\nLinked List: ");
while( p != null ) // start at beginning of list until end of list
{
System.out.print(p.data + " -> "); // print data
p = p.next; // move to next node
}
System.out.println(p); // prints 'null'
}
public boolean isEmpty() // true if list is empty
{
 return (head == null);
}
public int size()
{
return count;
}
} // end of LinkeList class
```

```
class LinkedListDemo
{
 public static void main(String[] args)
{
 LinkedList list = new LinkedList(); // create list object
list.createList(4); // create 4 nodes
list.displayList();
list.insertFirst(55); // insert 55 as first node
list.displayList();
list.insertAfter(66, 33); // insert 66 after 33
list.displayList();
Object item = list.deleteFirst(); // delete first node
if( item != null )
{
System.out.println("deleteFirst(): " + item);
list.displayList();
}
item = list.deleteAfter(22); // delete a node after node(22)
if( item != null )
{
System.out.println("deleteAfter(22): " + item);
list.displayList();
}


System.out.println("size(): " + list.size());
}
}
```

OUTPUT:



3. Write Java programs to implement the following using an array.
(a) Stack ADT (b) Queue ADT

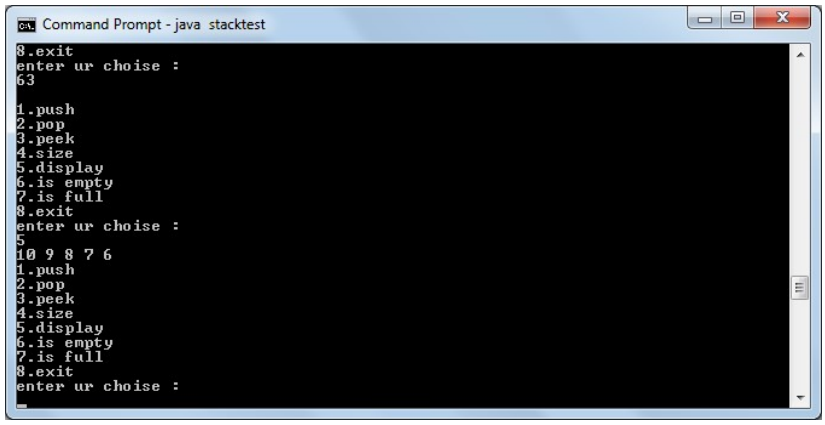
3(a) stack ADT using array

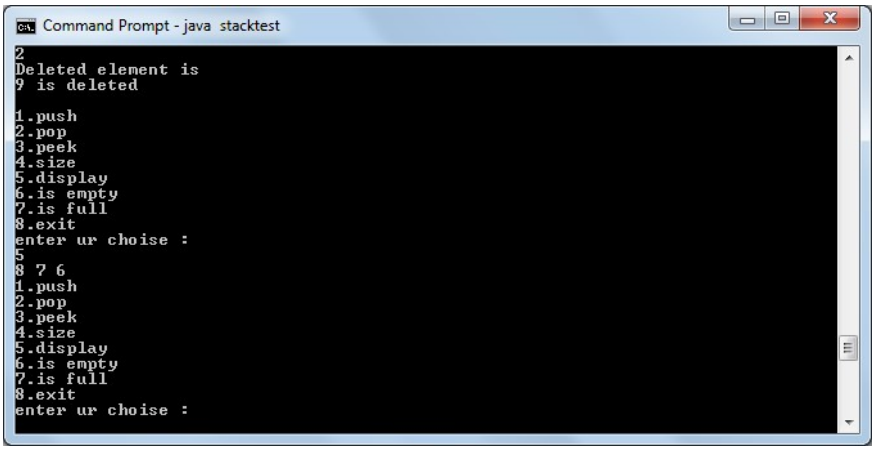
```
import java.io.*;
class stackclass
{
	int top,ele,stack[],size;
	stackclass(int n)
	{
		stack=new int[n];
		size=n;
		top= -1;
	}
	void push(int x)
	{
		ele=x;
		stack[++top]=ele;
	}
	int pop()
	{
		if(!isempty())
		{
			System.out.println("Deleted element is");
			return stack[top--];
		}
		else
		{
			System.out.println("stack is empty");
			return -1;
		}
	}
	boolean isempty()
	{
		if(top==-1)
			return true;
		else
			return false;
	}
	boolean isfull()
	{
		if(size>(top+1))
			return false;
		else
			return true;
	}
	int peek()
	{
		if(!isempty())
```

```
                return stack[top];
            else
            {
                System.out.println("stack is empty");
                return -1;
            }
        }
        void size()
        {
            System.out.println("size of the stack is :"+(top+1));
        }
        void display()
        {
            if(!isempty())
            {
                for(int i=top;i>=0;i--)
                    System.out.print(stack[i]+" ");
            }
            else
                System.out.println("stack is empty");
        }
}

class stacktest
{
        public static void main(String args[])throws Exception
        {
            BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
            System.out.println("enter the size of stack");
            int size=Integer.parseInt(br.readLine());
            stackclass s=new stackclass(size);
            int ch,ele;
            do
            {
                System.out.println();
                System.out.println("1.push");
                System.out.println("2.pop");
                System.out.println("3.peek");
                System.out.println("4.size");
                System.out.println("5.display");
                System.out.println("6.is empty");
                System.out.println("7.is full");
                System.out.println("8.exit");
                System.out.println("enter ur choise :");
                ch=Integer.parseInt(br.readLine());
                switch(ch)
                {
                    case 1:if(!s.isfull())
                            {
                    System.out.println("enter the element to insert: ");
```

```
                                        ele=Integer.parseInt(br.readLine());
                                        s.push(ele);
                                        }
                                        else
                                        {
                                        System.out.print("stack is overflow");
                                        }
                                        break;
                                case 2:int del=s.pop();
                                        if(del!=-1)
                                                System.out.println(del+" is deleted");
                                        break;
                                case 3:int p=s.peek();
                                        if(p!=-1)
                                                System.out.println("peek element is: +p);
                                        break;
                                case 4:s.size();
                                        break;
                                case 5:s.display();
                                        break;
                                case 6:boolean b=s.isempty();
                                        System.out.println(b);
                                        break;
                                case 7:boolean b1=s.isfull();
                                        System.out.println(b1);
                                        break;
                                case 8 :System.exit(1);
                        }
                }while(ch!=0);
        }
}
```

OUTPUT:





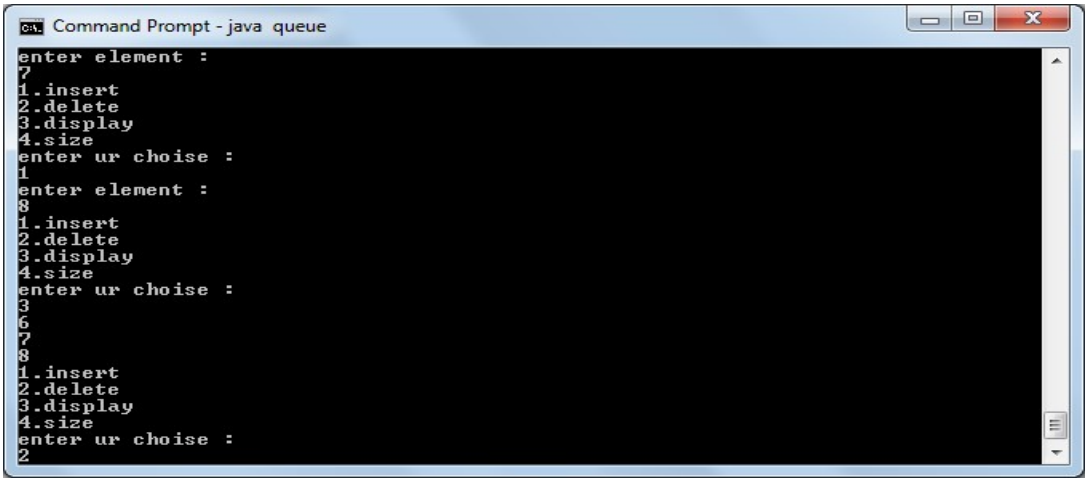
3(b)Queue ADT using array

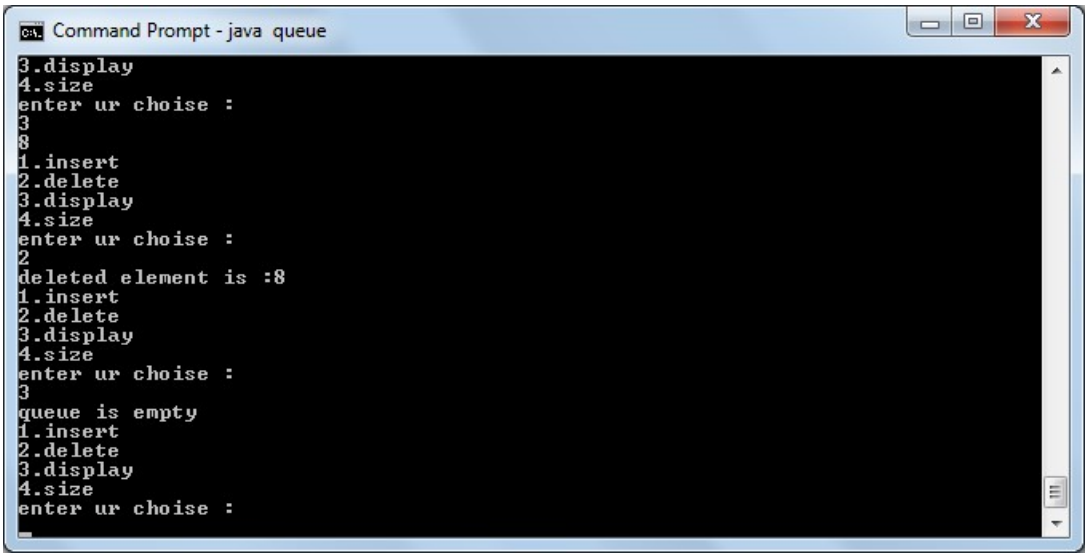
```
import java.util.*;
class queue
{
	int front,rear;
	int que[];
	int max,count=0;
	queue(int n)
	{
		max=n;
		que=new int[max];
		front=rear=-1;
	}
	boolean isfull()
	{
		if(rear==(max-1))
			return true;
		else
			return false;
	}
	boolean isempty()
	{
		if(front==-1)
			return true;
		else
			return false;
	}
	void insert(int n)
	{
		if(isfull())
			System.out.println("list is full");
		else
		{
			rear++;
			que[rear]=n;
			if(front==-1)
				front=0;
```

```
                count++;
        }
}
int delete()
{
        int x;
        if(isempty())
                return -1;
        else
        {
                x=que[front];
                que[front]=0;
                if(front==rear)
                        front=rear=-1;
                else
                        front++;
                count--;
        }
        return x;
}


void display()
{
        if(isempty())
                System.out.println("queue is empty");
        else
                for(int i=front;i<=rear;i++)
                        System.out.println(que[i]);
}
int size()
{
        return count;
}
public static void main(String args[])
{
        int ch;
        Scanner s=new Scanner(System.in);
        System.out.println("enter limit");
        int n=s.nextInt();
        queue q=new queue(n);
        do
        {
                System.out.println("1.insert");
                System.out.println("2.delete");
                System.out.println("3.display");
                System.out.println("4.size");
                System.out.println("enter ur choise :");
                ch=s.nextInt();
                switch(ch)
                {
                        case 1:System.out.println("enter element :");
                                int n1=s.nextInt();
                                q.insert(n1);
                                break;
```

```
                                case 2:int c1=q.delete();
                                        if(c1>0)
                                        System.out.println("deleted element is :"+c1);
                                        else
                                                System.out.println("can't delete");
                                        break;
                                case 3:q.display();
                                        break;
                                case 4:System.out.println("queue size is "+q.size());
                                        break;
                        }
                }
                while(ch!=0);
        }
}
```

OUTPUT:



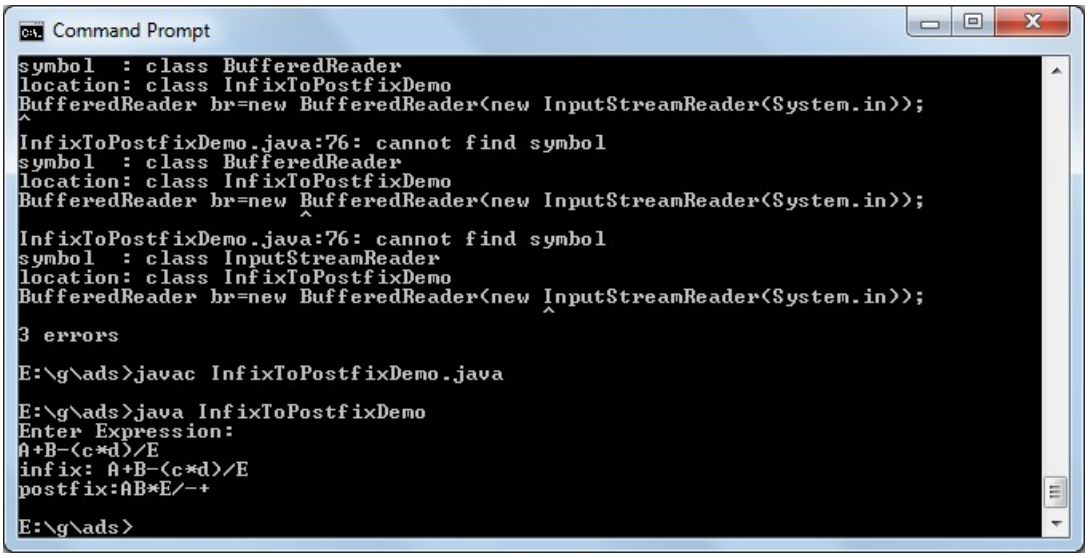


4. Write a java program that reads an infix expression, converts the expression to postfix form and then evaluates the postfix expression (use stack ADT).

```
import java.io.*;

class InfixToPostfix
{
java.util.Stack<Character> stk =new java.util.Stack<Character>();

public String toPostfix(String infix)
{
infix = "(" + infix + ")"; // enclose infix expr within parentheses
String postfix = "";
/* scan the infix char-by-char until end of string is reached */
for( int i=0; i<infix.length(); i++)
{
char ch, item;
ch = infix.charAt(i);
if( isOperand(ch) ) // if(ch is an operand), then
postfix = postfix + ch; // append ch to postfix string
if( ch == '(' ) // if(ch is a left-bracket), then
stk.push(ch); // push onto the stack
if( isOperator(ch) ) // if(ch is an operator), then
{
item = stk.pop(); // pop an item from the stack
/* if(item is an operator), then check the precedence of ch and item*/
if( isOperator(item) )
{
if( precedence(item) >= precedence(ch) )
{
stk.push(item);
stk.push(ch);
}
else
{
postfix = postfix + item;
stk.push(ch);
}
}
else
{
stk.push(item);
stk.push(ch);
}
} // end of if(isOperator(ch))

if( ch == ')' )
{
item = stk.pop();
while( item != '(' )
{
```

```
postfix = postfix + item;
item = stk.pop();
}
}
} // end of for-loop
return postfix;
} // end of toPostfix() method

public boolean isOperand(char c)
{
return(c >= 'A' && c <= 'Z');
}

public boolean isOperator(char c)
{
return( c=='+' || c=='-' || c=='*' || c=='/' );
}

public int precedence(char c)
{
int rank = 1; // rank = 1 for '*' or '/'
if( c == '+' || c == '-' ) rank = 2;
return rank;
}
}

//InfixToPostfixDemo.java
class InfixToPostfixDemo
{
public static void main(String args[]) throws IOException
{
InfixToPostfix obj = new InfixToPostfix();
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter Expression:");
String infix = br.readLine();
//String infix = "A*(B+C/D)-E";
System.out.println("infix: " + infix );
System.out.println("postfix:"+obj.toPostfix(infix) );
}
}
```

OUTPUT:

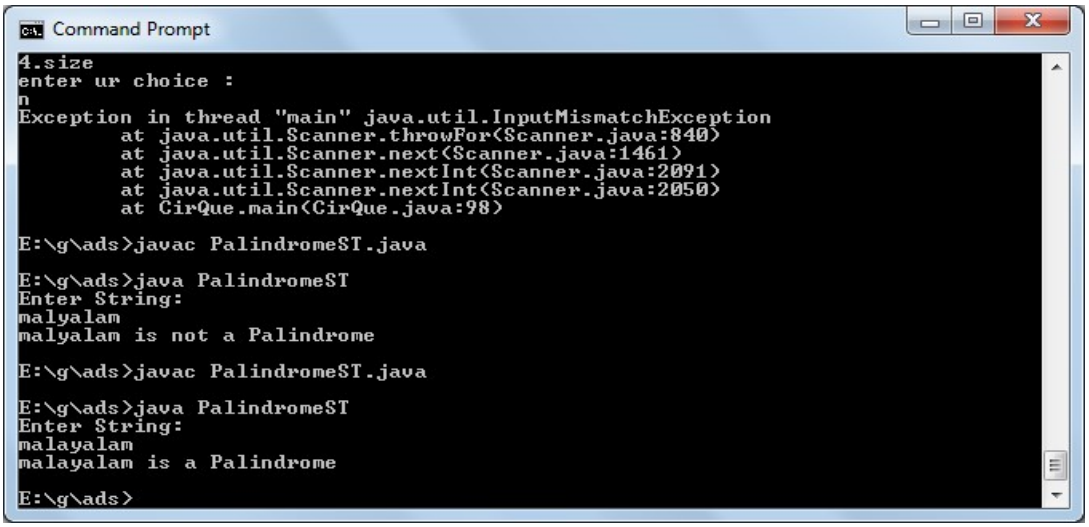

Command Prompt
symbol : class BufferedReader
location: class InfixToPostfixDemo
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
^
InfixToPostfixDemo.java:76: cannot find symbol
symbol : class BufferedReader
location: class InfixToPostfixDemo
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
^
InfixToPostfixDemo.java:76: cannot find symbol
symbol : class InputStreamReader
location: class InfixToPostfixDemo
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
^
3 errors
E:\g\ads>javac InfixToPostfixDemo.java
E:\g\ads>java InfixToPostfixDemo
Enter Expression:
A+B-(c*d)/E
infix: A+B-(c*d)/E
postfix:AB*E/-+
E:\g\ads>

5. Write a Java program that uses both a stack and a queue to test whether the given string is a palindrome or not.

5(a) palindrome using stack

```
import java.util.Stack;
import java.io.*;
class PalindromeST
{
public static void main(String args[]) throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter String:");
String str = br.readLine();
//String str = "MALAYALAM";
if( isPalindrome(str) )
System.out.println( str + " is a Palindrome");
else
System.out.println( str + " is not a Palindrome");
        }
static boolean isPalindrome(String str)
{
Stack<Character> stk = new Stack<Character>();
for( int i=0; i < str.length(); i++ )
stk.push(str.charAt(i));
for( int i=0; i < str.length()/2; i++ )
if( str.charAt(i) != stk.pop() )
return false;
return true;
}
}
```

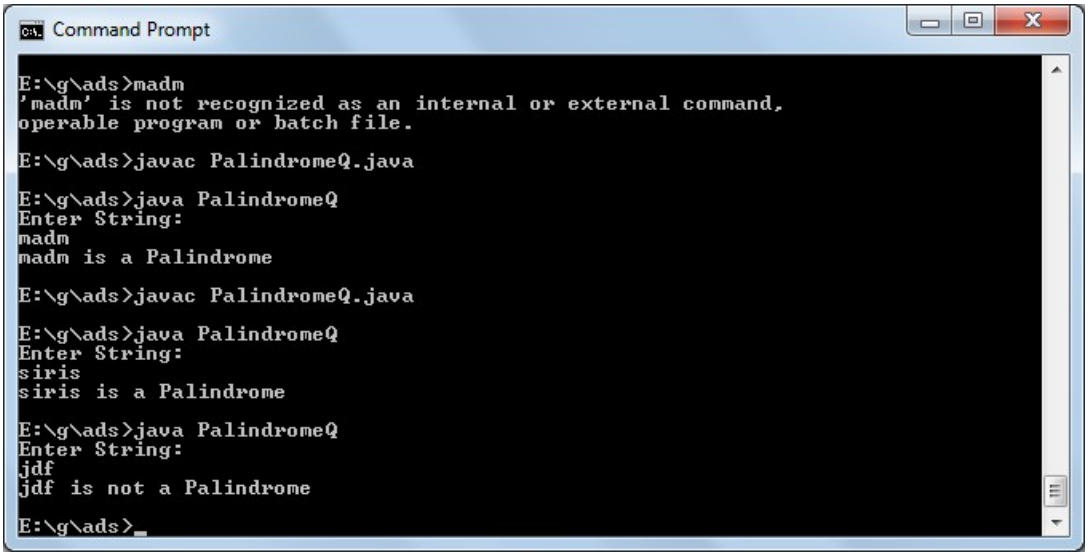
OUTPUT:



5(b) palindrome using queue

```
import java.util.LinkedList;
import java.io.*;
class PalindromeQ
{
public static void main(String args[]) throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter String:");
String str = br.readLine();
//String str = "RADAR";
if( isPalindrome(str) )
System.out.println( str + " is a Palindrome");
else
System.out.println( str + " is not a Palindrome");
}
static boolean isPalindrome(String str)
{
LinkedList<Character> que = new LinkedList<Character>();
int n = str.length();
for( int i=0; i < n; i++ )
que.addLast(str.charAt(i));
for( int i=n-1; i > n/2; i-- )
if( str.charAt(i) != que.removeFirst() )
return false;
return true;
}
}
```

OUTPUT:



6. Write Java programs to implement the following using a singly linked list.
(a) Stack ADT (b) Queue ADT

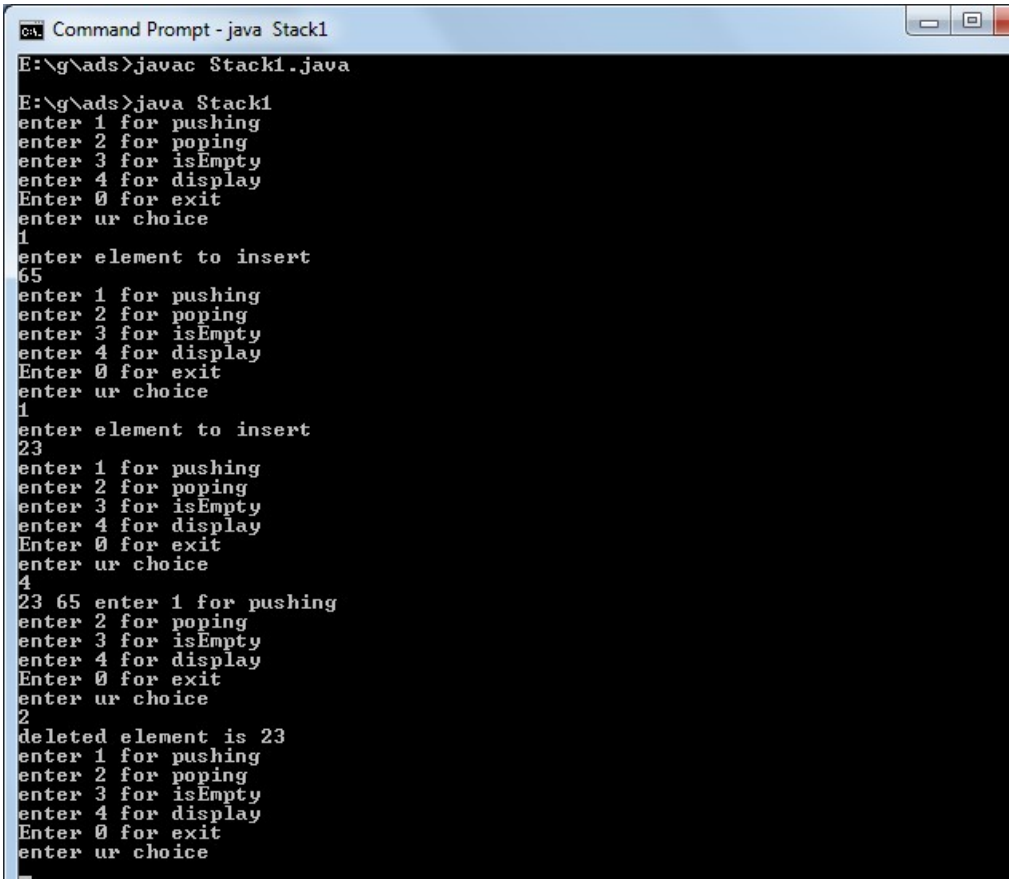
//6 (a) Stack

```
import java.io.*;
class Stack1
{
	Stack1 top,next,prev;
	int data;
	Stack1()
	{
		data=0;
		next=prev=null;
	}
	Stack1(int d)
	{
		data=d;
		next=prev=null;
	}
	void push(int n)
	{
		Stack1 nn;
		nn=new Stack1(n);
		if(top==null)
			top=nn;
		else
		{
			nn.next=top;
			top.prev=nn;
			top=nn;
		}
	}
	int pop()
	{
		int k=top.data;
		if(top.next==null)
		{
			top=null;
			return k;
		}
		else
		{
			top=top.next;
			top.prev=null;
			return k;
		}
	}
	boolean isEmpty()
	{
```

```
        if(top==null)
                return true;
        else
                return false;
}

void display()
{
        Stack1 ptr;
        for(ptr=top;ptr!=null;ptr=ptr.next)
                System.out.print(ptr.data+" ");
}
public static void main(String args[ ])throws Exception
{
        int x;
        int ch;
        BufferedReader b=new BufferedReader(new InputStreamReader(System.in));
        Stack1 a=new Stack1();
        do{
                System.out.println("enter 1 for pushing");
                System.out.println("enter 2 for poping");
                System.out.println("enter 3 for isEmpty");
                System.out.println("enter 4 for display");
                System.out.println("Enter 0 for exit");
                System.out.println("enter ur choice ");
                ch=Integer.parseInt(b.readLine());
                switch(ch)
                {
                        case 1:System.out.println("enter element to insert");
                                int e=Integer.parseInt(b.readLine());
                                a.push(e);
                                break;
                        case 2:if(!a.isEmpty())
                                {
                                int p=a.pop();
                                System.out.println("deleted element is "+p);
                                }
                                else
                                {
                                        System.out.println("stack is empty");
                                }
                                break;
                        case 3:System.out.println(a.isEmpty());
                                break;
                        case 4:if(!a.isEmpty())
                                {
                                        a.display();
                                }
                                else
                                {
```

```
                                        System.out.println("list is empty");
                                    }
                                }
                            }while(ch!=0);
                        }
}
```

OUTPUT:



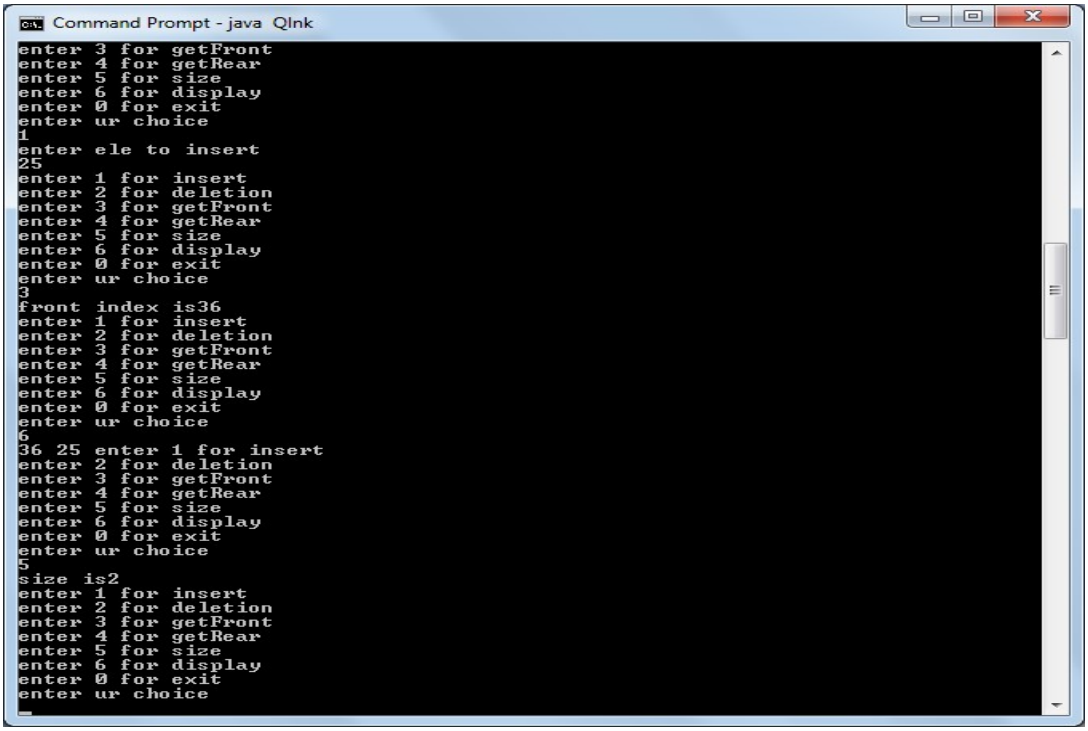
6 (b). Queue

```
import java.io.*;
class Qlnk
{
Qlnk front,rear,next;
int data;
Qlnk()
{
data=0;
next=null;
}
Qlnk(int d)
{
```

```
data=d;
next=null;
}
Qlnk getFront()
{
return front;
}
Qlnk getRear()
{
return rear;
}
void insertelm(int item)
{
Qlnk nn;
nn=new Qlnk(item);
if(isEmpty())
{
front=rear=nn;
}
else
{
rear.next=nn;
rear=nn;
}
}
int delelm()
{
if(isEmpty())
{
System.out.println("deletion failed");
return -1;
}
else
{
int k=front.data;
if(front!=rear)
front=front.next;
else
rear=front=null;
return k;
}
}
boolean isEmpty()
{
if(rear==null)
return true;
else
return false;
}
int size()
```

```
{
Qlnk ptr;
int cnt=0;
for(ptr=front;ptr!=null;ptr=ptr.next)
cnt++;
return cnt;
}
void display()
{
Qlnk ptr;
if(!isEmpty())
{
for(ptr=front;ptr!=null;ptr=ptr.next)
System.out.print(ptr.data+" ");
}
else
System.out.println("q is empty");
}
public static void main(String arr[])throws Exception
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
Qlnk m=new Qlnk();
int ch;
do
{
System.out.println("enter 1 for insert");
System.out.println("enter 2 for deletion");
System.out.println("enter 3 for getFront");
System.out.println("enter 4 for getRear");
System.out.println("enter 5 for size");
System.out.println("enter 6 for display");
System.out.println("enter 0 for exit");
System.out.println("enter ur choice");
ch=Integer.parseInt(br.readLine());
switch(ch)
{
case 1:System.out.println("enter ele to insert");
int item=Integer.parseInt(br.readLine());
m.insertelm(item);break;
case 2:int k=m.delelm();
System.out.println("deleted ele is "+k);break;
case 3:System.out.println("front index is"+(m.getFront()).data);break;
case 4:System.out.println("rear index is"+(m.getRear()).data);break;
case 5:System.out.println("size is"+m.size());break;
case 6:m.display();break;
}
}while(ch!=0);
}
}
```

OUTPUT:


Command Prompt - java QInk
enter 3 for getFront
enter 4 for getRear
enter 5 for size
enter 6 for display
enter 0 for exit
enter ur choice
1
enter ele to insert
25
enter 1 for insert
enter 2 for deletion
enter 3 for getFront
enter 4 for getRear
enter 5 for size
enter 6 for display
enter 0 for exit
enter ur choice
3
front index is36
enter 1 for insert
enter 2 for deletion
enter 3 for getFront
enter 4 for getRear
enter 5 for size
enter 6 for display
enter 0 for exit
enter ur choice
6
36 25 enter 1 for insert
enter 2 for deletion
enter 3 for getFront
enter 4 for getRear
enter 5 for size
enter 6 for display
enter 0 for exit
enter ur choice
5
size is2
enter 1 for insert
enter 2 for deletion
enter 3 for getFront
enter 4 for getRear
enter 5 for size
enter 6 for display
enter 0 for exit
enter ur choice

7. Write a Java program to perform the following operations:
a) Construct a binary search tree of elements.
b) Search for a key element in the above binary search tree.
c) Delete an element from the above binary search tree.

```
import java.util.*;
class Bstnode
{
	Bstnode rc,lc;
	Bstnode root;
	int data;
	Bstnode()
	{
		data=0;
		rc=lc=null;
	}
	Bstnode(int item)
	{
		data=item;
		lc=rc=null;
	}
	Bstnode[] search(int key)
	{
		Bstnode par ,ptr;
		Bstnode b[]=new Bstnode[2];
		ptr=root;
		par=null;
		while(ptr!=null)
		{
			if(ptr.data==key)
			{
				b[0]=par;
				b[1]=ptr;
				return b;
			}
			else if(ptr.data<key)
			{
				par=ptr;
				ptr=ptr.rc;
			}
			else
			{
				par=ptr;
				ptr=ptr.lc;
			}
		}
		b[0]=par;b[1]=ptr;
		return b;
	}
	void insert(int item)
```

```
{
        Bstnode arr[]=new Bstnode[2];
        Bstnode nn=new Bstnode(item);
        arr=search(item);
        if(root!=null)
        {
                Bstnode par=arr[0];
                Bstnode ptr=arr[1];
                if(ptr!=null)
                        System.out.println("key already existed");
                else
                {
                        if(par.data<item)
                                par.rc=nn;
                        else
                                par.lc=nn;
                }
        }
        else
                root=nn;
}
void inorder(Bstnode ptr)
{
        if(ptr!=null)
        {
                inorder(ptr.lc);
                System.out.println(ptr.data);
                inorder(ptr.rc);
        }
}

void preorder(Bstnode ptr)
{
        if(ptr!=null)
        {
                System.out.println(ptr.data);
                inorder(ptr.lc);
                inorder(ptr.rc);
        }
}

void postorder(Bstnode ptr)
{
        if(ptr!=null)
        {
                inorder(ptr.lc);
                inorder(ptr.rc);
                System.out.println(ptr.data);
        }
}
```

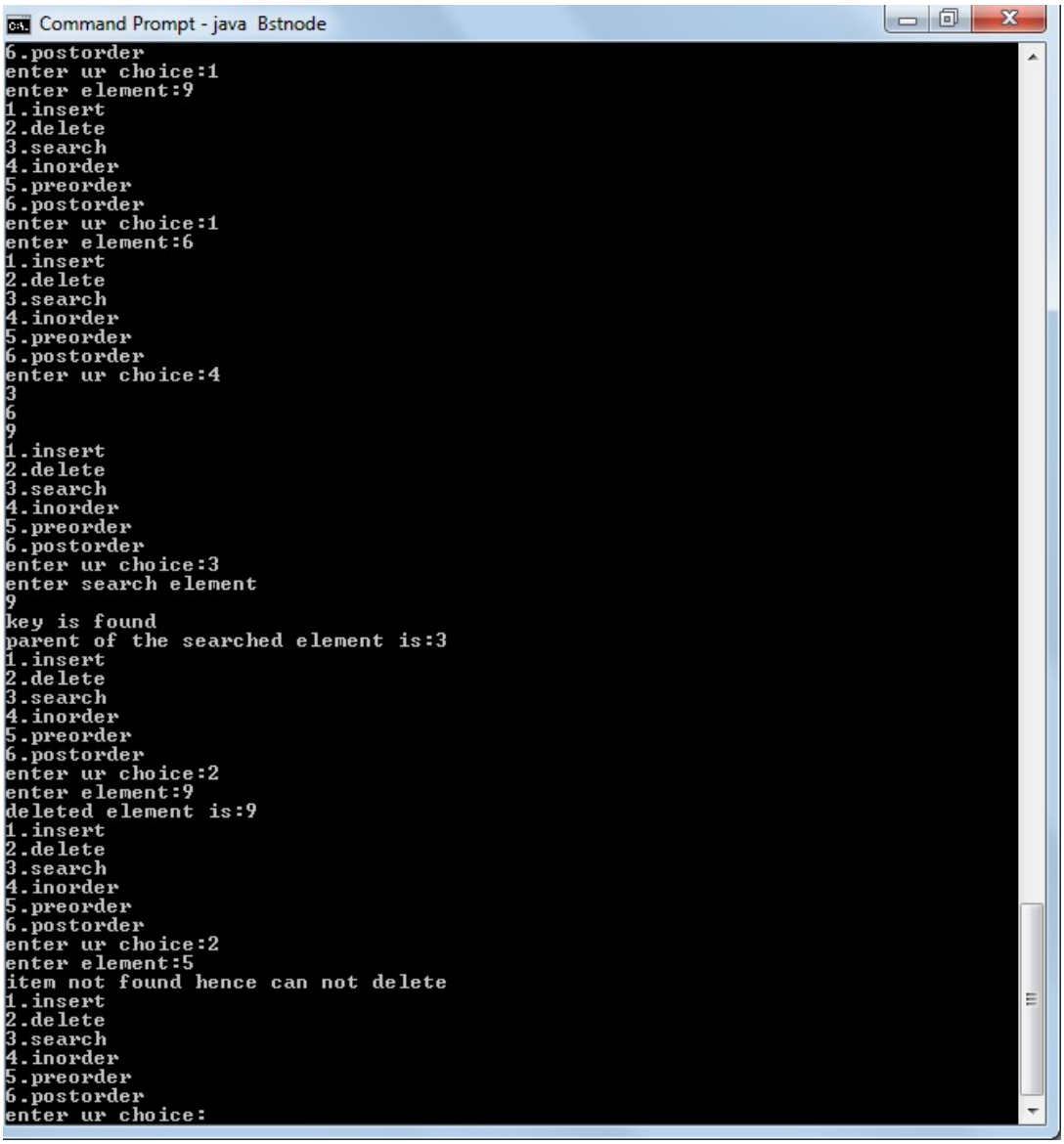
```
int deleteleaf(Bstnode par,Bstnode ptr)
{
        if(par!=null)
        {
                if(par.lc==ptr)
                        par.lc=null;
                else
                        par.rc=null;
        }
        else
                root=null;
        return ptr.data;
}

int delete1childnode(Bstnode par,Bstnode ptr)
{
        if(par!=null)
        {
                if(par.lc==ptr)
                {
                        if(ptr.lc==null)
                                par.lc=ptr.rc;
                        else
                                par.lc=ptr.lc;
                }
                else if(par.rc==ptr)
                {
                        if(ptr.lc==null)
                                par.rc=ptr.rc;
                        else
                                par.rc=ptr.lc;
                }
        }
        else
        {
                if(ptr.rc!=null)
                        root=ptr.rc;
                else
                        root=ptr.lc;
        }
        return ptr.data;
}

int delete2childnode(Bstnode par,Bstnode ptr)
{
        Bstnode ptr1=ptr.rc;
        Bstnode par1=null;
        while(ptr1.lc!=null)
        {
                par1=ptr1;
```

```
                ptr1=ptr1.lc;
        }
        if(par1!=null)
        {
                if(ptr1.rc!=null)
                        par1.lc=ptr1.rc;
                else
                        par1.lc=null;
                ptr1.lc=ptr.lc;
                ptr1.rc=ptr.rc;
        }
        else // if par1=null
                ptr1.lc = ptr.lc;
        if(par!=null)
        {
                if(par.lc==ptr)
                        par.lc=ptr1;
                else
                        par.rc=ptr1;
        }
        else
                root=ptr1;
        return ptr.data;
}

int deletenode(int item)
{
        Bstnode ptr=root,par=null;
        boolean flag=false;
        int k;
        while(ptr!=null&&flag==false)
        {
                if(item<ptr.data)
                {
                        par=ptr;
                        ptr=ptr.lc;
                }
                else if(item>ptr.data)
                {
                        par=ptr;
                        ptr=ptr.rc;
                }
                else
                {
                        ptr.data=item;
                        flag=true;
                }
        }
        if(flag==false)
        {
```

```
                System.out.println("item not found hence can not delete");
                return -1;
        }
        if(ptr.lc==null&&ptr.rc==null)
                k=deleteleaf(par,ptr);
        else if(ptr.lc!=null&&ptr.rc!=null)
                k=delete2childnode(par,ptr);
        else
                k=delete1childnode(par,ptr);
        return k;
}

public static void main(String saichandra[])
{
        Bstnode b=new Bstnode();
        Scanner s=new Scanner (System.in);
        int ch;
        do
        {
                System.out.println("1.insert");
                System.out.println("2.delete");
                System.out.println("3.search");
                System.out.println("4.inorder");
                System.out.println("5.preorder");
                System.out.println("6.postorder");
                System.out.print("enter ur choice:");
                ch=s.nextInt();
                switch(ch)
                {
                        case 1:System.out.print("enter element:");
                                int n=s.nextInt();
                                b.insert(n);
                                break;
                        case 2:if(b.root!=null)
                                {
                                System.out.print("enter element:");
                                int n1=s.nextInt();
                                int res=b.deletenode(n1);
                                if(res!=-1)
                                System.out.println("deleted element is:"+res);
                                }
                                else
                                System.out.println("no elements in tree");
                                break;
                        case 3:if(b.root!=null)
                                {
                                System.out.println("enter search element");
                                int key=s.nextInt();
                                Bstnode search1[]=new Bstnode[2];
                                search1=b.search(key);
```

```
                                        if(search1[1]!=null)
                                                System.out.println("key is found");
                                        else
                                                System.out.println("key not found");
                                        if(search1[0]!=null)
                                        {
                                        if(search1[1]!=null)
        System.out.println("parent of the searched element is:"+search1[0].data);
                                        }
                                        else
                                System.out.println("key is root no parent exist");
                                        }
                                        else
                                        System.out.println("no elements in tree");
                                        break;
                                case 4:if(b.root!=null)
                                                b.inorder(b.root);
                                        else
                                        System.out.println("no elements in tree");
                                        break;
                                case 5:if(b.root!=null)
                                                b.preorder(b.root);
                                        else
                                        System.out.println("no elements in tree");
                                        break;
                                case 6:if(b.root!=null)
                                                b.postorder(b.root);
                                        else
                                        System.out.println("no elements in tree");
                                        break;
                        }
                }while(ch!=0);
        }
}
```

OUTPUT:


Command Prompt - java Bstnode
6.postorder
enter ur choice:1
enter element:9
1.insert
2.delete
3.search
4.inorder
5.preorder
6.postorder
enter ur choice:1
enter element:6
1.insert
2.delete
3.search
4.inorder
5.preorder
6.postorder
enter ur choice:4
3
6
9
1.insert
2.delete
3.search
4.inorder
5.preorder
6.postorder
enter ur choice:3
enter search element
9
key is found
parent of the searched element is:3
1.insert
2.delete
3.search
4.inorder
5.preorder
6.postorder
enter ur choice:2
enter element:9
deleted element is:9
1.insert
2.delete
3.search
4.inorder
5.preorder
6.postorder
enter ur choice:2
enter element:5
item not found hence can not delete
1.insert
2.delete
3.search
4.inorder
5.preorder
6.postorder
enter ur choice:

8. Write a Java program to implement all the functions of a dictionary (ADT) using Hashing.

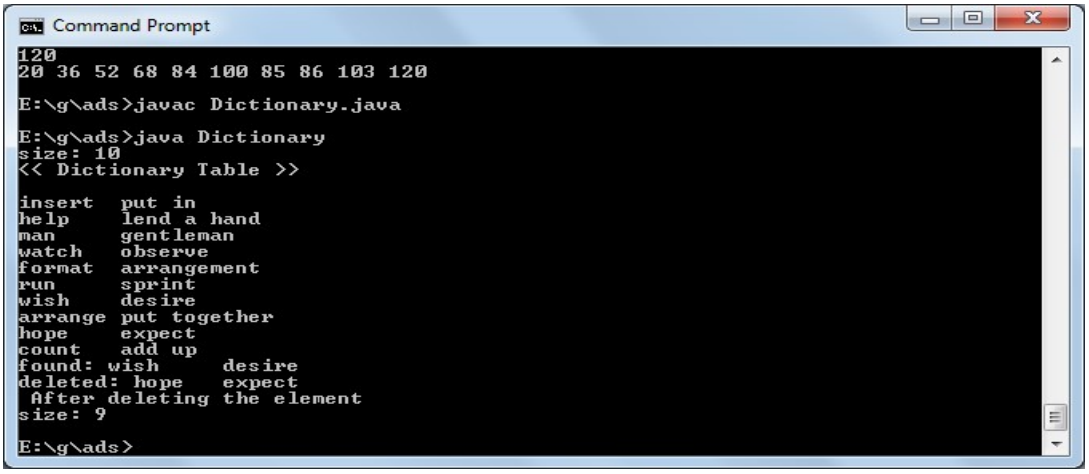
8(a): Dictionary operations using Hash tables

```
// Dictionary.java

class Entry
{
	public String key; // word
	public String element; // word meaning
	public Entry(String k, String e) // constructor
	{
		key = k;
		element = e;
	}
}
class HashTable
{
	Entry[] hashArray; // array holds hash table
	int size; // table size
	int count; // current number of items in the table
	public HashTable(int s) // constructor
	{
		size = s;
		count = 0;
		hashArray = new Entry[size];
	}
	int hashFunc( String theKey ) // convert the string into a numeric key
	{
		int hashVal=0;
		// convert the string into a numeric key
		for(int i = 0; i < theKey.length(); i++)
		hashVal = 37*hashVal + (int)theKey.charAt(i);
		hashVal = hashVal % size;
		if(hashVal < 0 )
		hashVal = hashVal + size;
		return hashVal;
	}
	public void insert(String theKey, String str) // insert a record
	{
		if( !isFull() )
		{
			int hashVal = hashFunc(theKey); // hash the key
			// until empty cell or null,
			while(hashArray[hashVal] != null )
			{
				++hashVal; // go to next cell
				hashVal %= size; // wraparound if necessary
			}
			hashArray[hashVal] = new Entry(theKey, str);
```

```
                count++; // update count
            }
            else
                System.out.println("Table is full");
        }
        public Entry delete(String theKey) // delete a record with the key
        {
            if( !isEmpty() )
            {
                int hashVal = hashFunc(theKey); // hash the key
                while(hashArray[hashVal] != null) // until empty cell,
                {
                    if(hashArray[hashVal].key == theKey) // found the key?
                    {
                        Entry tmp = hashArray[hashVal]; // save item
                        hashArray[hashVal] = null; // delete item
                        count--;
                        return tmp; // return item
                    }
                    ++hashVal; // go to next cell
                    hashVal %= size; // wraparound if necessary
                }
                return null; // cannot find item
            }
            else
                System.out.println("Table is empty");
            return null;
        }
        public Entry search(String theKey) // find item with key
        {
            int hashVal = hashFunc(theKey); // hash the key
            while(hashArray[hashVal] != null) // until empty cell,
            {
                if(hashArray[hashVal].key == theKey) // found the key?
                    return hashArray[hashVal]; // yes, return item
                ++hashVal; // go to next cell
                hashVal %= size; // wraparound if necessary
            }
            return null; // cannot find item
        }
        public void displayTable()
        {
            System.out.println("<< Dictionary Table >>\n");
            for(int i=0; i<size; i++)
            {
                if(hashArray[i] != null )
                    System.out.println( hashArray[i].key + "\t" +
hashArray[i].element );
            }
        }
```

```
	public boolean isEmpty() // returns true, if table is empty
	{
		return count == 0;
	}
	public boolean isFull() // returns true, if table is full
	{
		return count == size;
	}
	public int currentSize()
	{
		return count;
	}
}
////////////////////////// Dictionary.java ////////////////////////////
class Dictionary
{
	public static void main(String[] args)
	{
		HashTable ht = new HashTable(19); // create hash table of size, 19
		// Insert the following items into hash table
		ht.insert("man", "gentleman");
		ht.insert("watch", "observe");
		ht.insert("hope", "expect");
		ht.insert("arrange", "put together");
		ht.insert("run", "sprint");
		ht.insert("wish", "desire");
		ht.insert("help", "lend a hand");
		ht.insert("insert", "put in");
		ht.insert("count", "add up");
		ht.insert("format", "arrangement");
		System.out.println("size: " + ht.currentSize());
		ht.displayTable(); // Display the table items
		// Search an item
		String word = "wish";
		Entry item = ht.search(word);
		if( item != null )
			System.out.println("found: " + item.key + "\t" + item.element);
		else
			System.out.println(word + " not found");
		// Delete an item
		word = "hope";
		item = ht.delete(word);
		if( item != null )
			System.out.println("deleted: " + item.key + "\t" + item.element);
		else
			System.out.println(word + " not found - no deletion");
		// Current number of items in the table
		System.out.println(" After deleting the element");
		System.out.println("size: " + ht.currentSize());
	} }
```

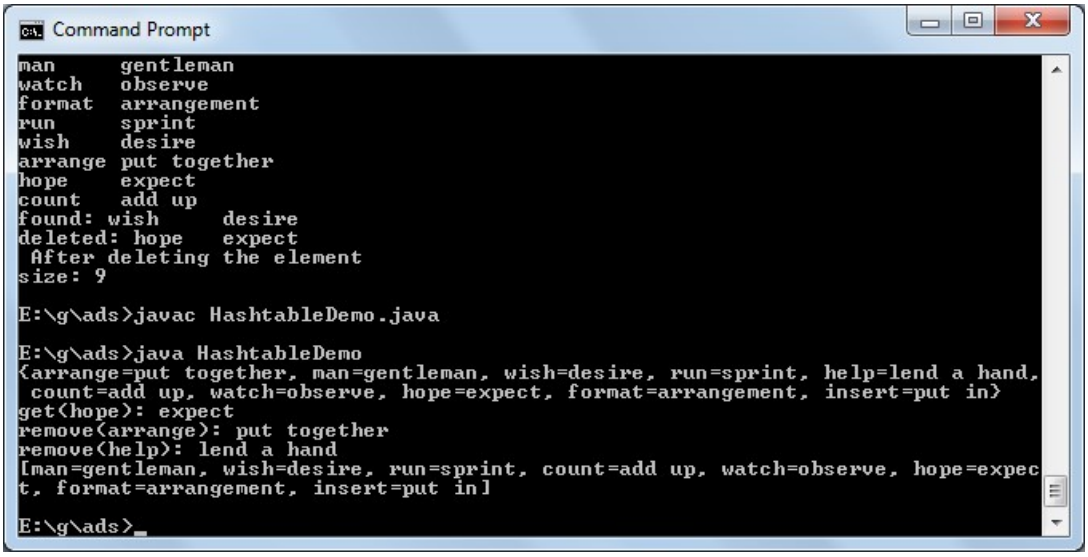
OUTPUT:



8 (b): Dictionary operations using java.util.Hashtable

```
import java.util.*;
class HashtableDemo
{
	public static void main(String[] args)
	{
		Hashtable<String, String> htab = new Hashtable<String, String>();
		// Insert the following items
		htab.put("man", "gentleman");
		htab.put("watch", "observe");
		htab.put("hope", "expect");
		htab.put("arrange", "put together");
		htab.put("run", "sprint");
		htab.put("wish", "desire");
		htab.put("help", "lend a hand");
		htab.put("insert", "put in");
		htab.put("count", "add up");
		htab.put("format", "arrangement");
		System.out.println(htab); // Display the table items
		System.out.println("get(hope): " + htab.get("hope"));
		System.out.println("remove(arrange): " + htab.remove("arrange"));
		System.out.println("remove(help): " + htab.remove("help"));
		// returns a set containing all the pairs (key, value).
		System.out.println(htab.entrySet());
	}
}
```

OUTPUT:


Command Prompt
man gentleman
watch observe
format arrangement
run sprint
wish desire
arrange put together
hope expect
count add up
found: wish desire
deleted: hope expect
After deleting the element
size: 9
E:\g\ads>javac HashtableDemo.java
E:\g\ads>java HashtableDemo
{arrange=put together, man=gentleman, wish=desire, run=sprint, help=lend a hand,
count=add up, watch=observe, hope=expect, format=arrangement, insert=put in}
get(hope): expect
remove(arrange): put together
remove(help): lend a hand
[man=gentleman, wish=desire, run=sprint, count=add up, watch=observe, hope=expec
t, format=arrangement, insert=put in]
E:\g\ads>

9. Write Java programs that use recursive and non-recursive functions to traverse the given binary tree in Preorder b) Inorder c) Postorder.

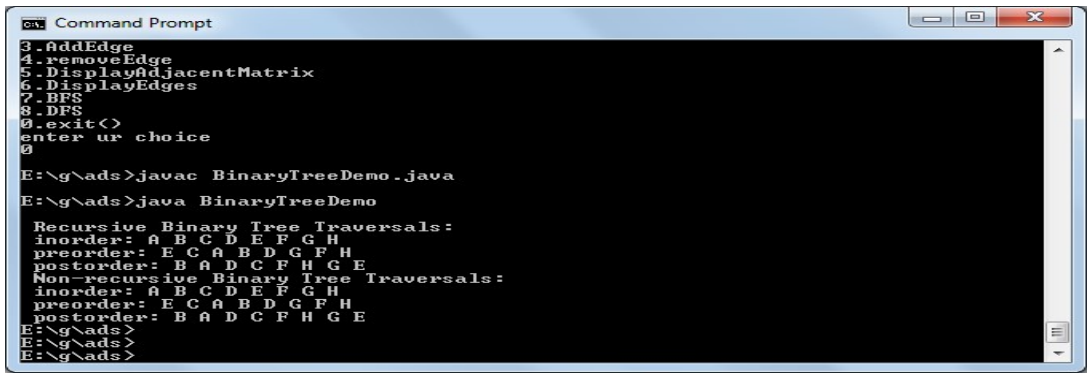
```
class Node
{
	Object data;
	Node left;
	Node right;
	Node( Object d ) // constructor
	{
		data = d;
	}
}
class BinaryTree
{
	Object tree[];
	int maxSize;
	java.util.Stack<Node> stk = new java.util.Stack<Node>();
	BinaryTree( Object a[], int n ) // constructor
	{
		maxSize = n;
		tree = new Object[maxSize];
		for( int i=0; i<maxSize; i++ )
			tree[i] = a[i];
	}
	public Node buildTree( int index )
	{
		Node p = null;
		if( tree[index] != null )
		{
			p = new Node(tree[index]);
			p.left = buildTree(2*index+1);
			p.right = buildTree(2*index+2);
		}
		return p;
	}


	/* Recursive methods - Binary tree traversals */
	public void inorder(Node p)
	{
		if( p != null )
		{
			inorder(p.left);
			System.out.print(p.data + " ");
			inorder(p.right);
		}
	}
	public void preorder(Node p)
	{
		if( p != null )
```

```
        {
                System.out.print(p.data + " ");
                preorder(p.left);
                preorder(p.right);
        }
}
public void postorder(Node p)
{
        if( p != null )
        {
                postorder(p.left);
                postorder(p.right);
                System.out.print(p.data + " ");
        }
}
/* Non-recursive methods - Binary tree traversals */
public void preorderIterative(Node p)
{
        if(p == null )
        {
                System.out.println("Tree is empty");
                return;
        }
        stk.push(p);
        while( !stk.isEmpty() )
        {
                p = stk.pop();
                if( p != null )
                {
                        System.out.print(p.data + " ");
                        stk.push(p.right);
                        stk.push(p.left);
                }
        }
}
public void inorderIterative(Node p)
{
        if(p == null )
        {
                System.out.println("Tree is empty");
                return;
        }
        while( !stk.isEmpty() || p != null )
        {
                if( p != null )
                {
                        stk.push(p); // push left-most path onto stack
                        p = p.left;
                }
                else
                {
```

```
                                p = stk.pop(); // assign popped node to p
                                System.out.print(p.data + " "); // print node data
                                p = p.right; // move p to right subtree
                        }
                }
        }
        public void postorderIterative(Node p)
        {
                if(p == null )
                {
                        System.out.println("Tree is empty");
                        return;
                }
                Node tmp = p;
                while( p != null )
                {
                        while( p.left != null )
                        {
                                stk.push(p);
                                p = p.left;
                        }
                        while( p != null && (p.right == null || p.right == tmp ))
                        {
                                System.out.print(p.data + " "); // print node data
                                tmp = p;
                                if( stk.isEmpty() )
                                        return;
                                p = stk.pop();
                        }
                        stk.push(p);
                        p = p.right;
                }
        }
} // end of BinaryTree class

class BinaryTreeDemo
{
        public static void main(String args[])
        {
                Object arr[] = {'E', 'C', 'G', 'A', 'D', 'F', 'H', null,'B',
                                null, null, null, null, null, null, null, null, null, null };
                BinaryTree t = new BinaryTree( arr, arr.length );
                Node root = t.buildTree(0); // buildTree() returns reference to root
                System.out.print("\n Recursive Binary Tree Traversals:");
                System.out.print("\n inorder: ");
                t.inorder(root);
                System.out.print("\n preorder: ");
                t.preorder(root);
                System.out.print("\n postorder: ");
                t.postorder(root);
                System.out.print("\n Non-recursive Binary Tree Traversals:");
```

```
                System.out.print("\n inorder: ");
                t.inorderIterative(root);
                System.out.print("\n preorder: ");
                t.preorderIterative(root);
                System.out.print("\n postorder: ");
                t.postorderIterative(root);
        }
}
```

OUTPUT:



10.Write Java programs for the implementation of bfs and dfs for a given graph.

//bfs

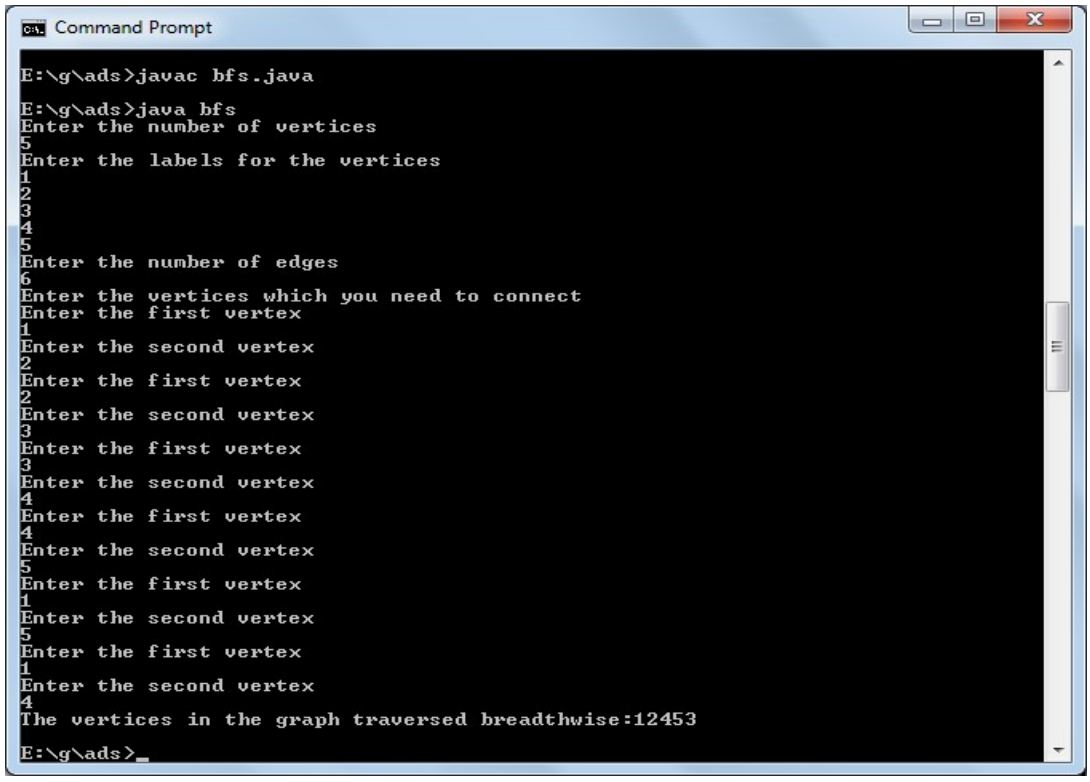
```
import java.io.*;
class quelist
{
  public int front;
  public int rear;
  public int maxsize;
  public int[] que;

  public quelist(int size)
  {
    maxsize = size;
    que = new int[size];
    front = rear = -1;
  }

  public void display()
  {
     for(int i = front;i<=rear;i++)
         System.out.print(que[i]+"     ");
  }

  public void enque(int x)
  {
     if(front==-1)
     front = 0;
     que[++rear]=x;
  }

  public int deque()
  {
    int temp = que[front];
    front = front +1;
    return temp;
  }

  public boolean isempty()
  {
    return((front>rear)||(front==-1));
  }
}

class vertex
{
 public char label;
```

```
 public boolean wasvisited;

 public vertex(char lab)
 {
  label = lab;
  wasvisited = false;
 }
}

class graph
{
 public final int MAX = 20;
 public int nverts;
 public int adj[][];
 public vertex vlist[];
 quelist qu;

 public graph()
 {
  nverts = 0;
  vlist = new vertex[MAX];
  adj = new int[MAX][MAX];
  qu = new quelist(MAX);
  for(int i=0;i<MAX;i++)
   for(int j=0;j<MAX;j++)
    adj[i][j] = 0;
 }
 public void addver(char lab)
 {
  vlist[nverts++] = new vertex(lab);
 }

 public void addedge(int start,int end)
 {
  adj[start][end] = 1;
  adj[end][start] = 1;
 }

 public int getadjunvis(int i)
 {
  for(int j=0;j<nverts;j++)
    if((adj[i][j]==1)&&(vlist[j].wasvisited==false))
    return j;
  return (MAX+1);
 }

 public void display(int i)
 {
  System.out.print(vlist[i].label);
 }
```

```
  public int getind(char l)
  {
    for(int i=0;i<nverts;i++)
      if(vlist[i].label==l)
      return i;
    return (MAX+1);
  }

  public void brfs()
  {
    vlist[0].wasvisited = true;
    display(0);
    qu.enque(0);
    int v2;
    while(!(qu.isempty()))
    {
     int v1 = qu.deque();
     while((v2=getadjunvis(v1))!=(MAX+1))
      {
    vlist[v2].wasvisited = true;
       display(v2);
       qu.enque(v2);
      }
    }
    System.out.print("\n");
  }
}
class bfs
{
  public static void main(String args[])throws IOException
  {
    graph gr = new graph();
    InputStreamReader isr = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(isr);
    System.out.println("Enter the number of vertices");
    int n = Integer.parseInt(br.readLine());
    System.out.println("Enter the labels for the vertices");
    for(int i=0;i<n;i++)
    {
      String temp = br.readLine();
      char ch = temp.charAt(0);
      gr.addver(ch);
    }
    System.out.println("Enter the number of edges");
    int edg = Integer.parseInt(br.readLine());
    System.out.println("Enter the vertices which you need to connect");
    for(int j=0;j<edg;j++)
    {
      System.out.println("Enter the first vertex");
```

```
    String t = br.readLine();
    char c = t.charAt(0);
    int start = gr.getind(c);

    System.out.println("Enter the second vertex");
    t = br.readLine();
    c = t.charAt(0);
    int end = gr.getind(c);

    gr.addedge(start,end);
   }
   System.out.print("The vertices in the graph traversed breadthwise:");
   gr.brfs();
  }
}
```

OUTPUT:



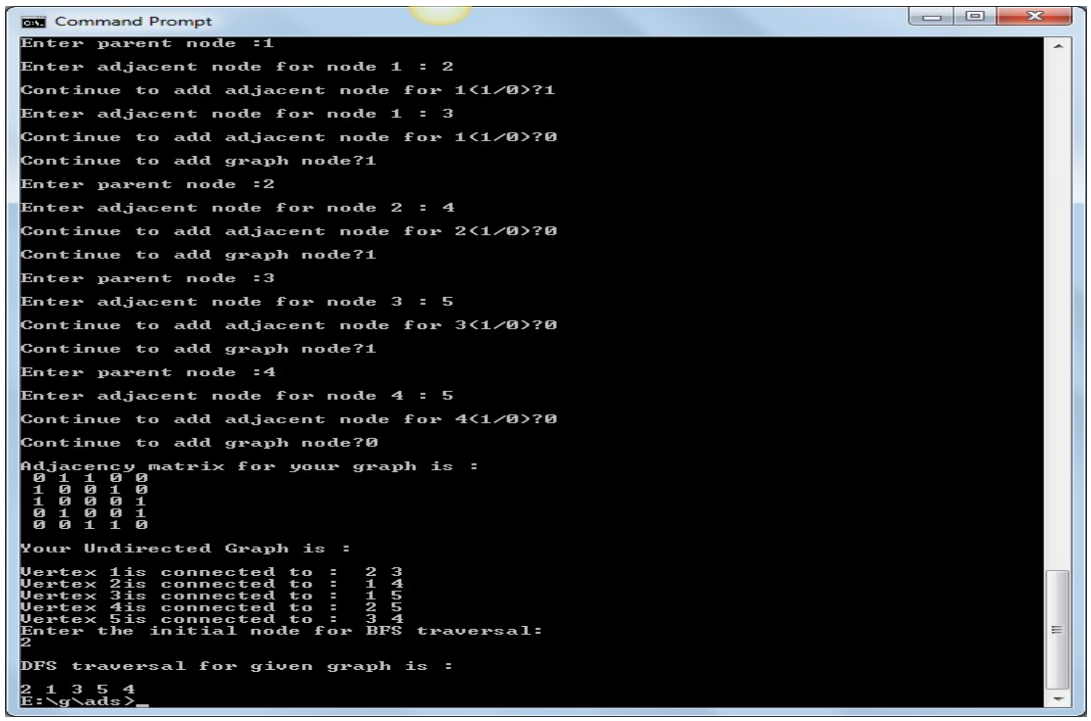
```
//dfs
import java.io.*;
import java.util.*;

class Stack
{
   int stk[]=new int[10];
   int top;
   Stack()
   {
      top=-1;
   }
   void push (int item)
   {
      if (top==9)
         System.out.println("Stack overflow");
      else
         stk[++top]=item;
   }/*end push*/

   boolean isempty()
   {
      if (top<0)
         return true;
      else
         return false;
   }/*end isempty*/

   int pop()
   {
      if (isempty())
      {
         System.out.println("Stack underflow");
         return 0;
      }
      else
         return (stk[top--]);
   }/*end pop*/

   void stacktop()
   {
      if(isempty())
         System.out.println("Stack underflow ");
      else
         System.out.println("Stack top is "+(stk[top]));
   }/*end stacktop*/

   void display()
   {
      System.out.println("Stack-->");
```

```
        for(int i=0;i<=top;i++)
            System.out.println(stk[i]);
    }/*end display*/
}

class Graph
{
    int MAXSIZE=51;
    int adj[][]=new int[MAXSIZE][MAXSIZE];
    int visited[]=new int [MAXSIZE];
    Stack s=new Stack();
    /*Function  for Depth-First-Search    */

    void createGraph()
    {
        int n,i,j,parent,adj_parent,initial_node;
        int  ans=0,ans1=0;

        System.out.print("\nEnter total number elements in a Undirected Graph :");
        n=getNumber();
        for ( i=1;i<=n;i++)
            for( j=1;j<=n;j++)
                adj[i][j]=0;


        /*All graph nodes are unvisited, hence assigned zero to visited field of each node */
        for (int c=1;c<=50;c++)
            visited[c]=0;
        System.out.println("\nEnter graph structure for BFS  ");

        do
        {
            System.out.print("\nEnter parent node :");
            parent=getNumber();
            do
            {
                System.out.print("\nEnter adjacent node for node "+parent+ " : ");
                adj_parent=getNumber();
                adj[parent][adj_parent]=1;
                adj[adj_parent][parent]=1;
                System.out.print("\nContinue to add adjacent node for "+parent+"(1/0)?");
                ans1= getNumber();
            } while (ans1==1);
            System.out.print("\nContinue to add graph node?");
            ans= getNumber();
        }while (ans ==1);


        System.out.print("\nAdjacency matrix for your graph is :\n");
        for (i=1;i<=n;i++)
```

```
    {
      for (j=1;j<=n;j++)
      System.out.print(" "+adj[i][j]);
      System.out.print("\n");
    }

    System.out.println("\nYour Undirected Graph is :");
    for (i=1;i<=n;i++)
    {
      System.out.print("\nVertex "+i+"is connected to : ");
      for (j=1;j<=n;j++)
      {
      if (adj[i][j]==1)
      System.out.print(" "+j);
      }
    }
    System.out.println("\nEnter the initial node for BFS traversal:");
    initial_node=getNumber();
    DFS (initial_node, n);
  }

  void DFS (int initial_node,int n)
  {
    int u,i;
    s.top = -1;
    s.push(initial_node);
    System.out.println("\nDFS traversal for given graph is : ");
    while(!s.isempty())
    {
      u=s.pop();
      if(visited[u]==0)
      {
        System.out.print("\n"+u);
        visited[u]=1;
      }
      for (i=1;i<=n;i++)
      {
        if((adj[u][i]==1) && (visited[i]==0))
        {
          s.push(u);
          visited[i]=1;
          System.out.print(" "+i);
          u = i;
        }
      }
    }
  }/*  end of DFS function  */


  int getNumber()
```

```
    {
        String str;
        int ne=0;
        InputStreamReader input=new InputStreamReader(System.in);
        BufferedReader in=new BufferedReader(input);
        try
        {
           str=in.readLine();
           ne=Integer.parseInt(str);
        }
        catch(Exception e)
        {
          System.out.println("I/O Error");
        }
        return ne;
    }
}

class Graph_DFS
{
   public static void main(String args[])
   {
      Graph g=new Graph();
      g.createGraph();
   }        /*  end of program */
}
```

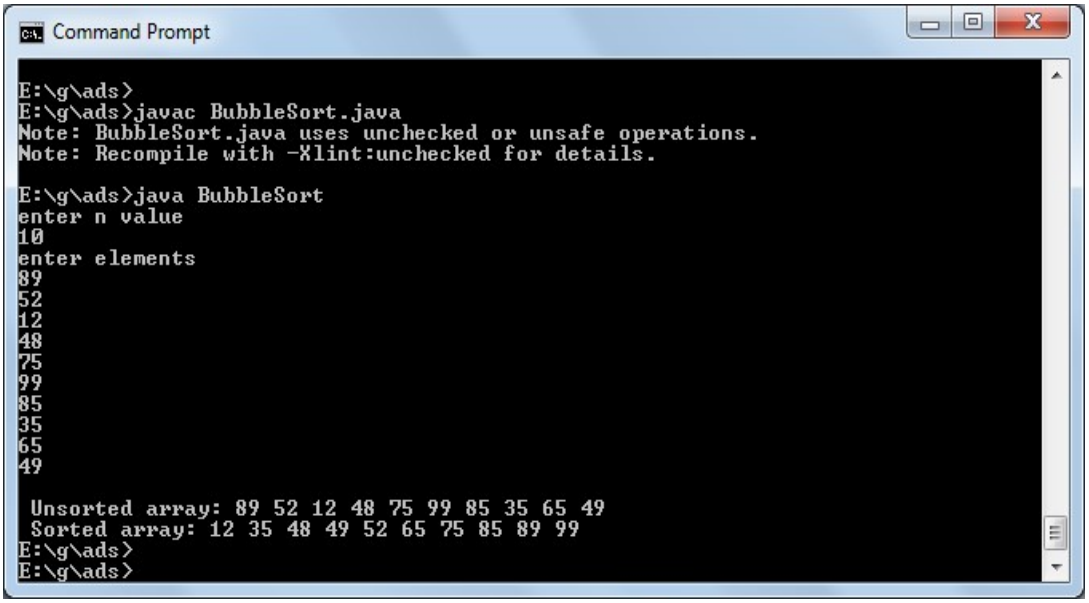
Output:



11. Write Java programs for implementing the following sorting methods:
a) Bubble sort d) Merge sort g) Binary tree sort
b) Insertion sort e) Heap sort
c) Quick sort f) Radix sort

```
//Bubble Sort
import java.io.*;
class BubbleSort
{
	public static void main(String[] args) throws IOException
	{
		BufferedReader br=new BufferedReader(new
					InputStreamReader(System.in));
		System.out.println("enter n value");
		int n=Integer.parseInt(br.readLine());
		int arr[]=new int[n];
		System.out.println("enter elements");
		for(int i=0;i<n;i++)
		{
			arr[i]=Integer.parseInt(br.readLine());
		}

		System.out.print("\n Unsorted array: ");
		display( arr );

		bubbleSort( arr );
		System.out.print("\n Sorted array: ");
		display( arr );
	}
	static void bubbleSort(int[] a)
	{
		int i, pass, exch, n = a.length;
		int tmp;
		for( pass = 0; pass < n; pass++ )
		{
			exch = 0;
			for( i = 0; i < n-pass-1; i++ )
				if( ((Comparable)a[i]).compareTo(a[i+1]) > 0)
				{
					tmp = a[i];
					a[i] = a[i+1];
					a[i+1] = tmp;
					exch++;
				}
			if( exch == 0 ) return;
		}
	}

	static void display( int a[] )
	{
		for( int i = 0; i < a.length; i++ )
		System.out.print( a[i] + " " );
	}
}
```

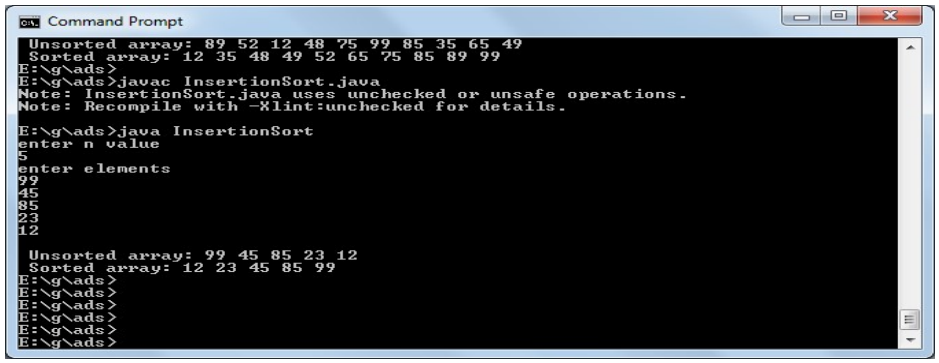
OUTPUT:




```
//Insertion Sort
import java.io.*;
class InsertionSort
{
	public static void main(String[] args) throws IOException
	{
		BufferedReader br=new BufferedReader(new
							InputStreamReader(System.in));
		System.out.println("enter n value");
		int n=Integer.parseInt(br.readLine());
		int arr[]=new int[n];
		System.out.println("enter elements");
		for(int i=0;i<n;i++)
		{
			arr[i]=Integer.parseInt(br.readLine());
		}

		System.out.print("\n Unsorted array: ");
		display( arr );

		insertionSort( arr );
		System.out.print("\n Sorted array: ");
		display( arr );
	}
```

```
        static void insertionSort(int a[])
        {
                int i, j, n = a.length;
                int item;
                for( j = 1; j < n; j++ )
                {
                        item = a[j];
                        i = j-1;
                        while( i >= 0 && ((Comparable)item).compareTo(a[i]) < 0)
                        {
                                a[i+1] = a[i];
                                i = i-1;
                        }
                        a[i+1] = item;
                }
        }
        static void display( int a[] )
        {
                for( int i = 0; i < a.length; i++ )
                System.out.print( a[i] + " " );
        }
}
```

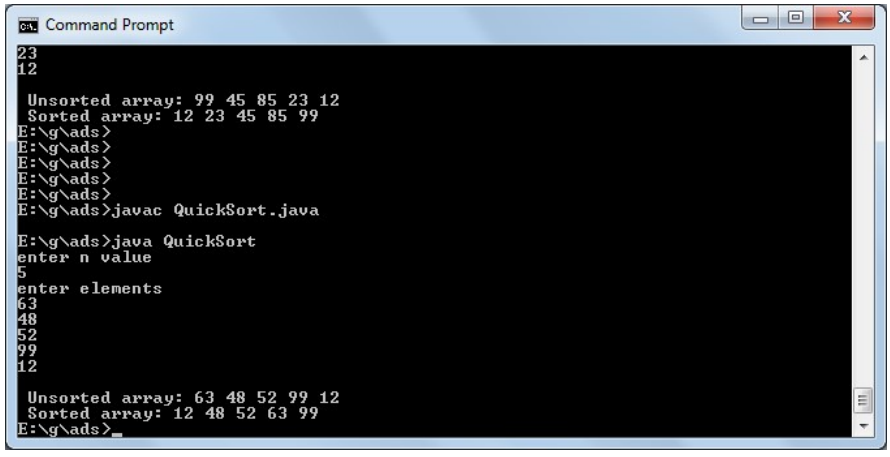
OUTPUT:




```
//Quick Sort
import java.io.*;
class QuickSort
{
        public static void main(String[] args) throws IOException
        {
                BufferedReader br=new BufferedReader(new
                                                InputStreamReader(System.in));
                System.out.println("enter n value");
                int n=Integer.parseInt(br.readLine());
                int arr[]=new int[n];
                System.out.println("enter elements");
                for(int i=0;i<n;i++)
                {
                        arr[i]=Integer.parseInt(br.readLine());
                }
        System.out.print("\n Unsorted array: ");
                display( arr );
```

```
            quickSort( arr, 0, arr.length-1 );
            System.out.print("\n Sorted array: ");
            display( arr );
        }
        static void quickSort(int a[], int left, int right)
        {
            int newleft = left, newright = right;
            int amid, tmp;
            amid = a[(left + right)/2];
            do
            {
                while( (a[newleft] < amid) && (newleft < right))
                    newleft++;
                while( (amid < a[newright]) && (newright > left))
                    newright--;
                if(newleft <= newright)
                {
                    tmp = a[newleft];
                    a[newleft] = a[newright];
                    a[newright] = tmp;
                    newleft++; newright--;
                }
            } while(newleft <= newright);
            if(left < newright)
                quickSort(a, left, newright);
            if(newleft < right)
                quickSort(a, newleft, right);
        }

        static void display( int a[] )
        {
            for( int i = 0; i < a.length; i++ )
            System.out.print( a[i] + " " );
        }
}
```

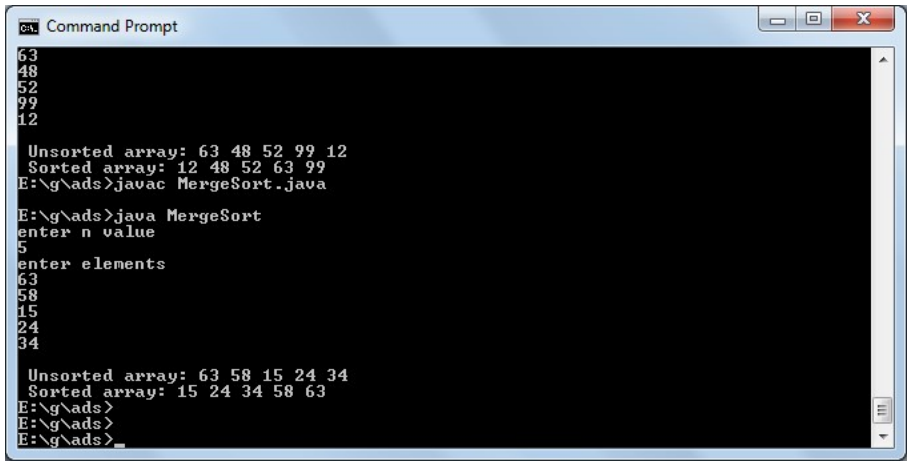
OUTPUT:



```
//Merge Sort

import java.io.*;
class MergeSort
{
	int[] a;
	int[] tmp;
	MergeSort(int[] arr)
	{
		a = arr;
		tmp = new int[a.length];
	}
	public static void main(String[] args) throws IOException
	{

		BufferedReader br=new BufferedReader(new
					InputStreamReader(System.in));
		System.out.println("enter n value");
		int n=Integer.parseInt(br.readLine());
		int arr[]=new int[n];
		System.out.println("enter elements");
		for(int i=0;i<n;i++)
		{
			arr[i]=Integer.parseInt(br.readLine());
		}

		System.out.print("\n Unsorted array: ");
		display( arr );

		MergeSort ms=new MergeSort(arr);
		ms.msort();

		System.out.print("\n Sorted array: ");
		display( arr );
	}
```

```
        void msort()
        {
                sort(0, a.length-1);
        }
        void sort(int left, int right)
        {
                if(left < right)
                {
                        int mid = (left+right)/2;
                        sort(left, mid);
                        sort(mid+1, right);
                        merge(left, mid, right);
                }
        }
        void merge(int left, int mid, int right)
        {
                int i = left;
                int j = left;
                int k = mid+1;
                while( j <= mid && k <= right )
                {       if(a[j] < a[k])
                                tmp[i++] = a[j++];
                        else
                                tmp[i++] = a[k++];
                }
                while( j <= mid )
                        tmp[i++] = a[j++];
                for(i=left; i < k; i++)
                        a[i] = tmp[i];
        }
        static void display( int a[] )
        {               for( int i = 0; i < a.length; i++ )
                System.out.print( a[i] + " " );
        }
}
```

OUTPUT:

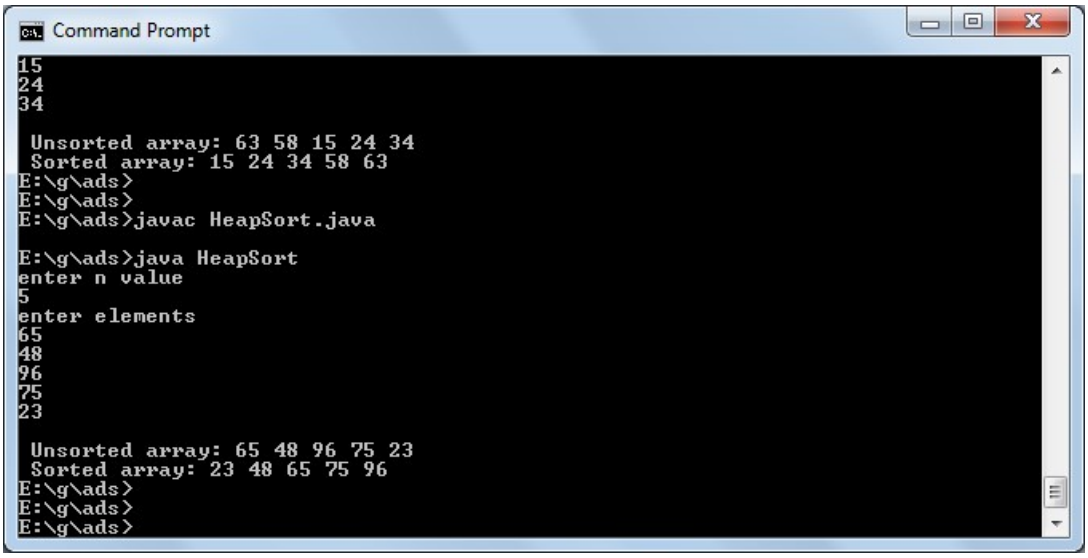


//Heap Sort

```
import java.io.*;

class HeapSort
{
	int[] a;
	int maxSize;
	int currentSize;
	public HeapSort(int m)
	{
		maxSize = m;
		currentSize = 0;
		a = new int[maxSize];
	}
	public static void main(String[] args) throws IOException
	{

		BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
		System.out.println("enter n value");
		int n=Integer.parseInt(br.readLine());
		int arr[]=new int[n];
		System.out.println("enter elements");
		for(int i=0;i<n;i++)
		{
			arr[i]=Integer.parseInt(br.readLine());
		}

		System.out.print("\n Unsorted array: ");
		display( arr );

		HeapSort hs=new HeapSort(n);
		hs.heapsort(arr);

		System.out.print("\n Sorted array: ");
		display( arr );
	}

	public boolean insert(int key)
	{
		if(currentSize == maxSize)
			return false;
		a[currentSize] = key;
		moveUp(currentSize++);
		return true;
	}
	public void moveUp(int index)
	{
		int parent = (index-1)/2;
		int bottom = a[index];
		while(index > 0 && a[parent] < bottom)
		{
			a[index] = a[parent];
			index = parent;
			parent = (parent-1)/2;
```

```
                }
                a[index] = bottom;
        }
        public int remove()
        {
                if( isEmpty() )
                {
                        System.out.println("Heap is empty");
                        return -1;
                }
                int root = a[0];
                a[0] = a[--currentSize];
                moveDown(0);
                return root;
        }
        public void moveDown(int index)
        {
                int largerChild;
                int top = a[index];
                while(index < currentSize/2)
                {
                        int leftChild = 2*index+1;
                        int rightChild = 2*index+2;
                        if(rightChild<currentSize && a[leftChild]<a[rightChild] )
                                largerChild = rightChild;
                        else
                                largerChild = leftChild;
                        if(top >= a[largerChild]) break;
                        a[index] = a[largerChild];
                        index = largerChild;
                }
                a[index] = top;
        }
        public boolean isEmpty()
        {
                return currentSize==0;
        }
        void heapsort(int []arr)
        {
                HeapSort h = new HeapSort(arr.length);
                for(int i = 0; i < arr.length; i++)
                        h.insert(arr[i]);
                for( int i = arr.length-1; i >= 0; i-- )
                        arr[i] = h.remove();
        }
        static void display( int a[] )
        {
                for( int i = 0; i < a.length; i++ )
                System.out.print( a[i] + " " );
        }
}
```

OUTPUT:

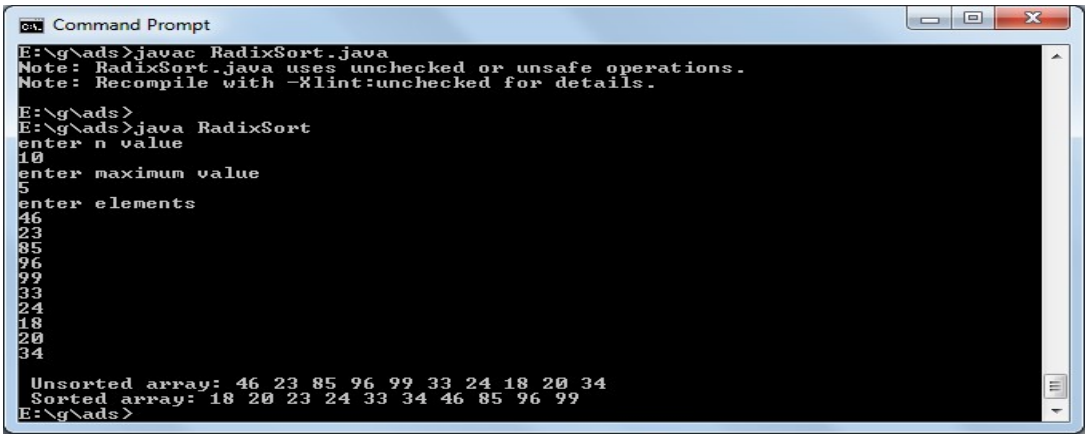


//Radix Sort

```
import java.io.*;
class RadixSort
{
	public static void main(String[] args) throws IOException
	{

		BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
		System.out.println("enter n value");
		int n=Integer.parseInt(br.readLine());
		BufferedReader br1=new BufferedReader(new InputStreamReader(System.in));
		System.out.println("enter maximum value");
		int n1=Integer.parseInt(br1.readLine());
		int arr[]=new int[n];
		System.out.println("enter elements");
		for(int i=0;i<n;i++)
		{
			arr[i]=Integer.parseInt(br.readLine());
		}

		System.out.print("\n Unsorted array: ");
		display( arr );

		radixSort(arr,n,n1);

		System.out.print("\n Sorted array: ");
		display( arr );
	}
	static void radixSort(int[] arr, int radix, int maxDigits)
```

```
        {
                int d, j, k, m, divisor;
                java.util.LinkedList[] queue = new java.util.LinkedList[radix];
                for( d = 0; d < radix; d++ )
                        queue[d] = new java.util.LinkedList();
                divisor = 1;
                for(d = 1; d <= maxDigits; d++)
                {
                        for(j = 0; j < arr.length; j++)
                        {
                                m = (arr[j]/divisor) % radix;
                                queue[m].addLast(new Integer(arr[j]));
                        }
                        divisor = divisor*radix;
                        for(j = k = 0; j < radix; j++)
                        {
                                while( !queue[j].isEmpty())
                                        arr[k++] = (Integer)queue[j].removeFirst();
                        }
                }
        }
        static void display( int a[] )
        {
                for( int i = 0; i < a.length; i++ )
                System.out.print( a[i] + " " );
        }
}
```

OUTPUT:



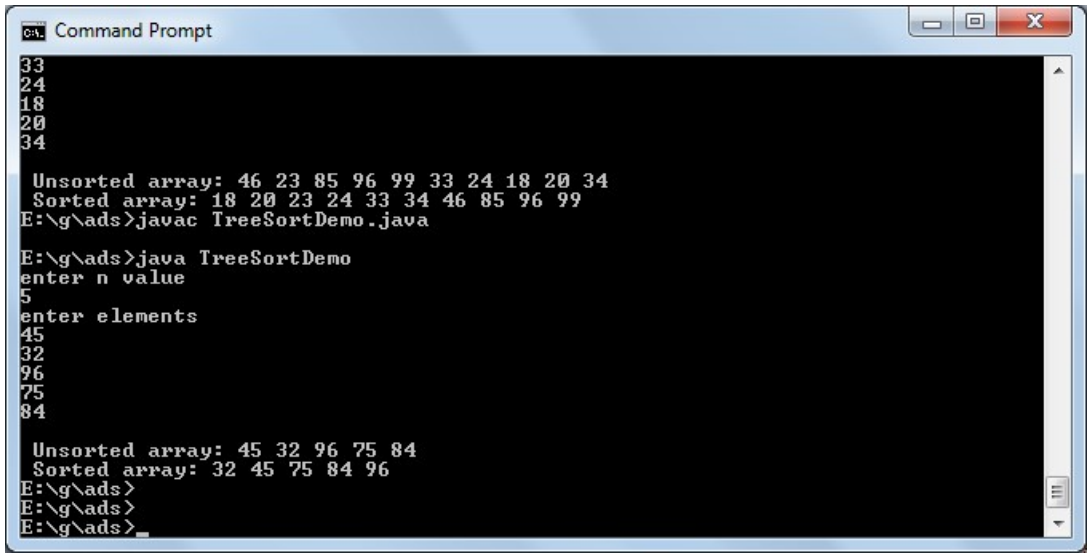
//Binary Tree Sort

```
import java.io.*;
class BSTNode
{
        int data;
        BSTNode left;
        BSTNode right;
```

```
        BSTNode( int d ) // constructor
        {
                data = d;
        }
}

class BinarySearchTree
{
        int i;
        int[] a;
        BSTNode root;
        BinarySearchTree(int[] arr) // constructor
        {
                a = new int[arr.length];
                a = arr;
        }
        private void buildTree()
        {
                for( i = 0; i < a.length; i++ )
                        root = insertTree( root, a[i] );
        }

       private BSTNode insertTree(BSTNode p, int key)
        {
                if( p == null )
                        p = new BSTNode(key);
                else if( key < p.data)
                        p.left = insertTree( p.left, key);
                else p.right = insertTree( p.right, key);
                        return p;
        }

      public void treeSort()
        {
                buildTree();
                i = 0;
                inorder(root);
        }
        private void inorder(BSTNode p) // 'p' starts with root
        {
                if( p != null )
                {
                        inorder(p.left);
                        a[i++] = p.data;
                        inorder(p.right);
                }
        }
        public void display(int a[])
        {
                for( i = 0; i < a.length; i++ )
```

```
                System.out.print(a[i] + " " );
        }

}

class TreeSortDemo
{
        public static void main(String args[]) throws IOException
        {
                //int arr[] = { 55, 22, 99, 77, 11, 88, 44, 66, 33 };
                BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                System.out.println("enter n value");
                int n=Integer.parseInt(br.readLine());
                int arr[]=new int[n];
                System.out.println("enter elements");
                for(int i=0;i<n;i++)
                {
                        arr[i]=Integer.parseInt(br.readLine());
                }

                BinarySearchTree bst = new BinarySearchTree(arr);
                System.out.print("\n Unsorted array: ");
                bst.display( arr );

                bst.treeSort();

                System.out.print("\n Sorted array: ");
                bst.display( arr );
        }
}
```

OUTPUT:



12. Write a Java program to perform the following operations:
a) Insertion into a B-tree b) Searching in a B-tree

```
class BTree
{
	final int MAX = 4;
	final int MIN = 2;
	class BTNode // B-Tree node
	{
		int count;
		int key[] = new int[MAX+1];
		BTNode child[] = new BTNode[MAX+1];
	}
	BTNode root = new BTNode();
	class Ref // This class creates an object reference
	{
		int m;
	} // and is used to retain/save index values
	// of current node between method calls.
	/*
	* New key is inserted into an appropriate node.
	* No node has key equal to new key (duplicate keys are not allowed.
	*/
	void insertTree( int val )
	{
		Ref i = new Ref();
		BTNode c = new BTNode();
		BTNode node = new BTNode();
		boolean pushup;
		pushup = pushDown( val, root, i, c );
		if ( pushup )
		{
			node.count = 1;
			node.key[1] = i.m;
			node.child[0] = root;
			node.child[1] = c;
			root = node;
		}
	}
	/*
	* New key is inserted into subtree to which current node points.
	* If pushup becomes true, then height of the tree grows.
	*/
	boolean pushDown( int val, BTNode node, Ref p, BTNode c )
	{
		Ref k = new Ref();
		if ( node == null )
		{
			p.m = val;
			c = null;
			return true;
		}
		else
		{
```

```
                if ( searchNode( val, node, k ) )
                        System.out.println("Key already exists.");
                if ( pushDown( val, node.child[k.m], p, c ) )
                {
                        if ( node.count < MAX )
                        {
                                pushIn( p.m, c, node, k.m );
                                return false;
                        }
                        else
                        {
                                split( p.m, c, node, k.m, p, c );
                                return true;
                        }
                }
                return false;
        }
}
/*
* Search through a B-Tree for a target key in the node: val
* Outputs target node and its position (pos) in the node
*/
BTNode searchTree( int val, BTNode root, Ref pos )
{
        if ( root == null )
                return null ;
        else
        {
                if ( searchNode( val, root, pos ) )
                        return root;
                else
                        return searchTree( val, root.child[pos.m], pos );
        }
}
/*
* This method determines if the target key is present in
* the current node, or not. Seraches keys in the current node;
* returns position of the target, or child on which to continue search.
*/
boolean searchNode( int val, BTNode node, Ref pos )
{
        if ( val < node.key[1] )
        {
                pos.m = 0 ;
                return false ;
        }
        else
        {
                pos.m = node.count ;
                while ( ( val < node.key[pos.m] ) && pos.m > 1 )
                        (pos.m)--;
                if ( val == node.key[pos.m] )
                        return true;
                else
                        return false;
```

```
            }
        }
        /*
        * Inserts the key into a node, if there is room
        * for the insertion
        */
        void pushIn( int val, BTNode c, BTNode node, int k )
        {
            int i ;
            for ( i = node.count; i > k ; i-- )
            {
                node.key[i + 1] = node.key[i];
                node.child[i + 1] = node.child[i];
            }
            node.key[k + 1] = val ;
            node.child[k + 1] = c ;
            node.count++ ;
        }
        /*
        * Splits a full node into current node and new right child
        * with median.
        */
        void split( int val, BTNode c, BTNode node,int k, Ref y, BTNode newnode )
        {
            int i, mid; // mid is median
            if ( k <= MIN )
                mid = MIN;
            else
                mid = MIN + 1;
            newnode = new BTNode();
            for ( i = mid+1; i <= MAX; i++ )
            {
                newnode.key[i-mid] = node.key[i];
                newnode.child[i-mid] = node.child[i];
            }
            newnode.count = MAX - mid;
            node.count = mid;
            if ( k <= MIN )
                pushIn ( val, c, node, k );
            else
                pushIn ( val, c, newnode, k-mid ) ;
            y.m = node.key[node.count];
            newnode.child[0] = node.child[node.count] ;
            node.count-- ;
        }
        // calls display( )
        void displayTree()
        {
            display( root );
        }
        // displays the B-Tree
        void display( BTNode root )
        {
            int i;
            if ( root != null )
```

```
                {
                        for ( i = 0; i < root.count; i++ )
                        {
                                display( root.child[i] );
                                System.out.print( root.key[i+1] + " " );
                        }
                        display( root.child[i] );
                }
        }
} // end of BTree class
////////////////////////// BTreeDemo.java ////////////////////////////
class BTreeDemo
{
        public static void main( String[] args )
        {
                BTree bt = new BTree();

                int[] arr = { 11, 23, 21, 12, 31, 18, 25, 35, 29, 20, 45,
                27, 42, 55, 15, 33, 36, 47, 50, 39 };
                for ( int i = 0; i < arr.length; i++ )
                        bt.insertTree( arr[i] );
                System.out.println("B-Tree of order 5:");
                bt.displayTree();
        }
}
```

OUTPUT: