```
// Java program for implementation of Selection

Sortclass SelectionSort

{

        void sort(int arr[])

        {

                int n = arr.length;


                // One by one move boundary of unsorted

                subarrayfor (int i = 0; i < n-1; i++)

                {

                        // Find the minimum element in unsorted

                        arrayint min_idx = i;

                        for (int j = i+1; j < n; j++)

                                if (arr[j] < arr[min_idx])

                                      min_idx = j;


                        // Swap the found minimum element with the first

                        // element

                        int temp =

                        arr[min_idx];

                        arr[min_idx] = arr[i];

                        arr[i] = temp;

                }

        }


        // Prints the array
```

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

    // Driver code to test above
    public static void main(String args[])
    {
        SelectionSort ob = new SelectionSort();
        int arr[] = {64,25,12,22,11};
        ob.sort(arr);
        System.out.println("Sorted array");
        ob.printArray(arr);
    }
}
```

Output:

Sorted array

11 12 22 25 64

AIM:Write a java programs for implememntation of Bubblesort

```
public class BubbleSortExample {
    static void bubbleSort(int[] arr) {
        int n = arr.length;
        int temp = 0;
         for(int i=0; i < n; i++){
                 for(int j=1; j < (n-i); j++){
                          if(arr[j-1] > arr[j]){
                                 //swap elements
                                 temp = arr[j-1];
                                 arr[j-1] = arr[j];
                                 arr[j] = temp;
                         }

                 }
         }

    }
    public static void main(String[] args) {
                int arr[] ={3,60,35,2,45,320,5};

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

```java
        System.out.println();

        bubbleSort(arr);//sorting array elements using bubble sort

        System.out.println("Array After Bubble Sort");
        for(int i=0; i < arr.length; i++){
            System.out.print(arr[i] + " ");
        }

    }
}
```

Output:

```
Array Before Bubble Sort
3 60 35 2 45 320 5
Array After Bubble Sort
2 3 5 35 45 60 320
```

1.Linear Search Program :

```
public class LinearSearch
{
  public static int linearSearch(int a[],int n,int val )
   {
     for (int i=0;i<n;i++)
     {
       if (a[i]==val)
       return i;
     }
    return -1;
   }
public static void main (String args[])
  {
    int a[]={55,29,10,40,57,41,20,24,45};
    int val=55;
    int n =a.length;
    int res =linearSearch(a,n,val);
System.out.println();
    System.out.print("The elements of the array are -");
    for(int i=0;i<n;i++)
    System.out.println(" "+a[i]);
     System.out.println();
```

```
      System.out.println(" Element to be searched is -"+val);
   if ( res==-1)
      System.out.println(" Elements is not present in the array");
   else
      System.out.println(" Elements is present at "+res+" position of the array");
 }
}
```

OUTPUT:

```
The elements of the array are - 55
 29
 10
 40
 57
 41
 20
 24
 45

 Element to be searched is -55
 Elements is present at 0 position of the array
```

2. BINARY SEARCH PROGRAM :

```
class BinarySearch
{
 static int binarySearch(int a[],int beg,int end,int val)
  {
    int mid;
    if (end>=beg)
    {
      mid=(beg+end)/2;
      if(a[mid]==val)
      {
        return mid;
      }
        else if (a[mid]<val)
        {
          return binarySearch(a,mid+1,end,val);
        }
          else
           {
             return binarySearch(a,beg,mid-1,val);
            }
    }
     return-1;
  }
```

```
  public static void main(String args[])
  {
   int a[]={8,10,22,27,37,44,49,55,69};
   int val =10;
   int n =a.length;
   int  res=binarySearch(a,0,n-1,val);
   System.out.println(" The elements of the array is :- ");
   for (int i=0;i<n;i++)
     {
       System.out.println(a[i]+"");
     }
     System.out.println();
     System.out.println("Elements to be searched is :- "+val);
     if (res==-1)
       System.out.println("Element is not present in the array ");
     else
      System.out.println("Element is present at "+res+"position of the array");
 }
}
```

OUTPUT:

The elements of the array is :-

8 10 22 37 44 49 55 69

Elements to be searched is :- 10

Element is present at 1position of the array

3.INSERTION SORT :

```
class InsertionSort
{
 public static void sortInsertion(int [] sort_arr)
  {
   for(int i=0;i<sort_arr.length;i++)
    {
     int j=i;
     while (j>0 && sort_arr[j-1]>sort_arr[j])
     {
      int key = sort_arr[j];
      sort_arr[j]= sort_arr[j-1];
      sort_arr[j-1]=key;
      j=j-1;
     }
    }
   }
public static void main (String args[])
 {
  int [] arr= {9,7,8,4,2,1};
  sortInsertion(arr);
  for(int i=0;i<arr.length;++i)
  {
   System.out.println(arr[i]+ " ");
```

```
  }
 }
}
```

OUTPUT:

1 2 4 7 8 9

```
/ JAVA program for implementation of KMP pattern
// searching algorithm

class KMP_String_Matching {
	void KMPSearch(String pat, String txt)
	{
		int M = pat.length();
		int N = txt.length();

		// create lps[] that will hold the longest
		// prefix suffix values for pattern
		int lps[] = new int[M];
		int j = 0; // index for pat[]

		// Preprocess the pattern (calculate lps[]
		// array)
		computeLPSArray(pat, M, lps);

		int i = 0; // index for txt[]
		while (i < N) {
			if (pat.charAt(j) == txt.charAt(i)) {
				j++;
				i++;
			}
			if (j == M) {
```

```
                System.out.println("Found pattern "
                                    + "at index " + (i - j));
                j = lps[j - 1];
            }

            // mismatch after j matches
            else if (i < N && pat.charAt(j) != txt.charAt(i)) {
                // Do not match lps[0..lps[j-1]] characters,
                // they will match anyway
                if (j != 0)
                    j = lps[j - 1];
                else
                    i = i + 1;
            }
        }
    }

    void computeLPSArray(String pat, int M, int lps[])
    {
        // length of the previous longest prefix suffix
        int len = 0;
        int i = 1;
        lps[0] = 0; // lps[0] is always 0

        // the loop calculates lps[i] for i = 1 to M-1
```

```
        while (i < M) {
            if (pat.charAt(i) == pat.charAt(len)) {
                len++;
                lps[i] = len;
                i++;
            }
            else // (pat[i] != pat[len])
            {
                // This is tricky. Consider the example.
                // AAACAAAA and i = 7. The idea is similar
                // to search step.
                if (len != 0) {
                    len = lps[len - 1];

                    // Also, note that we do not increment
                    // i here
                }
                else // if (len == 0)
                {
                    lps[i] = len;
                    i++;
                }
            }
        }
    }
```

```
	// Driver program to test above function
	public static void main(String args[])
	{
		String txt = "ABABDABACDABABCABAB";
		String pat = "ABABCABAB";
		new KMP_String_Matching().KMPSearch(pat, txt);
	}
}
```

OUTPUT:

Found pattern at index 10

9.

```
// A class for creation of nodes of the binary Tree
// nodes of the binary tree contain
// a left and a right reference
// and a value of the node
class TreeNode
{
// for holding value of the node
int val;

// for referring to the other nodes
TreeNode left, right;
```

```
// constructor of the class TreeNode

// the construct initializes the class fields

public TreeNode(int i)

{

val = i;

right = left = null;

}

}


public class BTreeLevelOrder

{

// top node i.e. root of the Binary Tree

TreeNode r;


// constructor of the class BTree

public BTreeLevelOrder() { r = null; }


// method for displaying the level order traversal of the binary tree

void displayLevelOrder()

{

int ht = treeHeight(r);

int j;


for (j = 1; j <= ht; j++)
```

```
{
displayCurrentLevel(r, j);
}
}

// finding the "height" of the binary tree
// Note that the total number of nodes
// present in the longest path from the topmost node (root node_
// to the leaf node, which is farthest from the root node, gives the
// height of the  tree
int treeHeight(TreeNode r)
{
if (r == null)
{
return 0;
}
else
{
// finding the height of the left and right subtrees
int lh = treeHeight(r.left);
int rh = treeHeight(r.right);

// picking up the larger one
if (lh > rh)
{
```

```
return (lh + 1);

}

else

{

return (rh + 1);

}

}

}


// Printing nodes present in the current level

void displayCurrentLevel(TreeNode r, int l)

{

// null means nothing is there to print

if (r == null)

{

return;

}


// l == 1 means only one node

// is present in the binary tree

if (l == 1)

{

System.out.print(r.val + " ");

}
```

```
// l > 1 means either there are nodes present in
// the left side of the current node or in the
// right side of the current node or in both sides
// therefore, we have to look in the left as well as in
// the right side of the current node
else if (l > 1)
{
displayCurrentLevel(r.left, l - 1);
displayCurrentLevel(r.right, l - 1);
}
}

// main method
public static void main(String argvs[])
{
// creating an object of the class BTreeLevelOrder
BTreeLevelOrder  tree = new BTreeLevelOrder ();

// root node
tree.r = new TreeNode(18);

// remaining nodes of the tree
tree.r.left = new TreeNode(20);
tree.r.right = new TreeNode(30);
tree.r.left.left = new TreeNode(60);
```

```
tree.r.left.right = new TreeNode(34);
tree.r.right.left = new TreeNode(45);
tree.r.right.right = new TreeNode(65);
tree.r.left.left.left = new TreeNode(12);
tree.r.left.left.right = new TreeNode(50);
tree.r.left.right.left = new TreeNode(98);
tree.r.left.right.right = new TreeNode(82);
tree.r.right.left.left = new TreeNode(31);
tree.r.right.left.right = new TreeNode(59);
tree.r.right.right.left = new TreeNode(71);
tree.r.right.right.right = new TreeNode(41);


System.out.println("Level order traversal of binary tree is ");
tree.displayLevelOrder();
}
}
```

```
// Array-based list implementation
class AList implements List {
  private Object listArray[];              // Array holding list
elements
  private static final int DEFAULT_SIZE = 10; // Default size
  private int maxSize;                     // Maximum size of list
  private int listSize;                    // Current # of list items
  private int curr;                        // Position of current
element

  // Constructors
  // Create a new list object with maximum size "size"
  AList(int size) {
    maxSize = size;
    listSize = curr = 0;
    listArray = new Object[size];          // Create listArray
  }
  // Create a list with the default capacity
  AList() { this(DEFAULT_SIZE); }           // Just call the other
constructor

  public void clear()                      // Reinitialize the list
    { listSize = curr = 0; }               // Simply reinitialize
values

  // Insert "it" at current position
  public boolean insert(Object it) {
    if (listSize >= maxSize) return false;
    for (int i=listSize; i>curr; i--)  // Shift elements up
      listArray[i] = listArray[i-1];   //   to make room
```

```java
    listArray[curr] = it;
    listSize++;                             // Increment list size
    return true;
  }

  // Append "it" to list
  public boolean append(Object it) {
    if (listSize >= maxSize) return false;
    listArray[listSize++] = it;
    return true;
  }

  // Remove and return the current element
  public Object remove() throws NoSuchElementException {
    if ((curr<0) || (curr>=listSize))  // No current element
      throw new NoSuchElementException("remove() in AList has current
of " + curr + " and size of "
        + listSize + " that is not a a valid element");
    Object it = listArray[curr];        // Copy the element
    for(int i=curr; i<listSize-1; i++) // Shift them down
      listArray[i] = listArray[i+1];
    listSize--;                         // Decrement size
    return it;
  }

  public void moveToStart() { curr = 0; }        // Set to front
  public void moveToEnd() { curr = listSize; } // Set at end
  public void prev() { if (curr != 0) curr--; } // Move left
  public void next() { if (curr < listSize) curr++; } // Move right
```

```java
  public int length() { return listSize; }      // Return list size

  public int currPos() { return curr; }         // Return current
position


  // Set current list position to "pos"

  public boolean moveToPos(int pos) {

    if ((pos < 0) || (pos > listSize)) return false;

    curr = pos;

    return true;

  }


  // Return true if current position is at end of the list

  public boolean isAtEnd() { return curr == listSize; }


  // Return the current element

  public Object getValue() throws NoSuchElementException {

    if ((curr < 0) || (curr >= listSize)) // No current element

      throw new NoSuchElementException("getvalue() in AList has
current of " + curr + " and size of "

        + listSize + " that is not a a valid element");

    return listArray[curr];

  }


  // Check if the list is empty

  public boolean isEmpty() { return listSize == 0; }

}
```

List ADT.

```
// defining node of the list
class node{

    public;

    int data; // to store the data
    node* next; // to store the address of the next List node

    node(int val) // a constructor to initialize the node parameters
    {
        data=val;
        next=NULL;
    }
}

class list{

    int count=0; // to count the number of nodes in the list

    public:

    int front(); // returns value of the node present at the front of the list
    int back(); // returns value of the node present at the back of the list
    void push_front(int val); // creates a pointer with value = val and keeps this pointer to the front of the linked list
```

void push_back(int val); // creates a pointer with value = val and keeps this pointer to the back of the linked list

void pop_front(); // removes the front node from the list

void pop_back(); // removes the last node from the list

bool empty(); // returns true if list is empty, otherwise returns false

int size(); // returns the number of nodes that are present in the list

};

Stack ADT

class node{

public:

int data; *// to store data in a stack node*

node* next; *// to store the address of the next node in the stack*

node(int val) *// a constructor to initialize stack parameters*

{

data=val;

next=NULL;

```
    }
};

class stack(){
    int count=0; // to count number of nodes in the stack

    public:

    int top(); // returns value of the node present at the top of the stack
    void push(int val); // creates a node with value = val and put it at the stack top
    void pop(); // removes node from the top of the stack
    bool empty(); // returns true if stack is empty, otherwise returns false
    int size(); // returns the number of nodes that are present in the stack
};
```

Queue ADT

```
class node{

   public:

   int data; // to store data in a stack node
   node* next; // to store the address of the next node in the stack

   node(int val) // a constructor to initialize stack parameters
   {
      data=val;
      next=NULL;
   }
};

class queue{

   int count=0; // to count number of nodes in the stack

   public:

   int front(); // returns value of the node present at the front of the queue
   int back(); // returns value of the node present at the back of the queue
   void push(int val); // creates a node with value = val and put it at the front of the queue
   void pop(); // removes node from the rear of the queue
```

```
  bool empty(); // returns true if queue is empty, otherwise returns false
  int size(); // returns the number of nodes that are present in the queue};
//singly linked list
class Node{
  int data;
 Node next;
}
void insertAtStart(Node newNode, Node head){
 newNode.data = 10;
 newNode.next = head;
 head.next = newNode;
}
void insertAfterTargetNode(Node newNode, Node head, int target){
 newNode.data = 10;
 Node temp = head;
 while(temp.data != target){
 temp = temp.next;
 }
 newNode.next = temp.next;
 temp.next = newNode;
}
void insertAtEnd(Node newNode, Node head){
 newNode.data = 10;
 Node temp = head;
 while(temp.next != null){
```

```
    temp = temp.next;
  }
  temp.next = newNode;
  newNode.next = null;
}
void deleteAtFirst(Node head){
  head = head.next;
}void deleteAfterTarget(Node head, int target){
  Node temp = head;
  while(temp.data != target){
    temp = temp.next;
 }
  temp.next= temp.next.next;
}
void deleteLast(Node head){
  Node temp = head;
  while(temp.next.next != null){
  temp = temp.next;
  }
  temp.next = null;
}
void display(Node head){
  Node temp = head;
  while(temp != null){
    System.out.println(temp.data);
```

```
    temp = temp.next;
  }
 }
Node search(Node head, int target){
  Node temp = head;
  while(temp != null && temp.data != target){
    temp = temp.next;
  }
  return temp;
}
```

```
//INFIX TO POST FIX  EXPRESSION
import java.util.Stack;

public class InfixToPostFix {

    static int precedence(char c){
        switch (c){
            case '+':
            case '-':
                return 1;
            case '*':
            case '/':
                return 2;
            case '^':
                return 3;
        }
        return -1;
    }

    static String infixToPostFix(String expression){

        String result = "";
        Stack<Character> stack = new Stack<>();
        for (int i = 0; i <expression.length() ; i++) {
            char c = expression.charAt(i);
```

```
        //check if char is operator
        if(precedence(c)>0){
            while(stack.isEmpty()==false && precedence(stack.peek())>=precedence(c)){
                result += stack.pop();
            }
            stack.push(c);
        }else if(c==')'){
            char x = stack.pop();
            while(x!='('){
                result += x;
                x = stack.pop();
            }
        }else if(c=='('){
            stack.push(c);
        }else{
            //character is neither operator nor (
            result += c;
        }
    }
    for (int i = 0; i <=stack.size() ; i++) {
        result += stack.pop();
    }
    return result;
}
```

```java
    public static void main(String[] args) {
        String exp = "A+B*(C^D-E)";
        System.out.println("Infix Expression: " + exp);
        System.out.println("Postfix Expression: " + infixToPostFix(exp));
    }
}
```

Output :-

Infix Expression: A+B*(C^D-E)

Postfix Expression: ABCD^E-*+

```
ackage com.java2novice.ds.queue;

import java.util.ArrayList;
import java.util.List;

public class DoubleEndedQueueImpl {

    private List<Integer> deque = new ArrayList<Integer>();

    public void insertFront(int item){
        //add element at the beginning of the queue
        System.out.println("adding at front: "+item);
        deque.add(0,item);
        System.out.println(deque);
    }

    public void insertRear(int item){
        //add element at the end of the queue
        System.out.println("adding at rear: "+item);
        deque.add(item);
        System.out.println(deque);
    }

    public void removeFront(){
        if(deque.isEmpty()){
```

```
        System.out.println("Deque underflow!! unable to remove.");
        return;
    }
    //remove an item from the beginning of the queue
    int rem = deque.remove(0);
    System.out.println("removed from front: "+rem);
    System.out.println(deque);
}

public void removeRear(){
    if(deque.isEmpty()){
        System.out.println("Deque underflow!! unable to remove.");
        return;
    }
    //remove an item from the beginning of the queue
    int rem = deque.remove(deque.size()-1);
    System.out.println("removed from front: "+rem);
    System.out.println(deque);
}

public int peakFront(){
    //gets the element from the front without removing it
    int item = deque.get(0);
    System.out.println("Element at first: "+item);
    return item;
```

```
        }

        public int peakRear(){
                //gets the element from the rear without removing it
                int item = deque.get(deque.size()-1);
                System.out.println("Element at rear: "+item);
                return item;
        }

        public static void main(String a[]){

                DoubleEndedQueueImpl deq = new DoubleEndedQueueImpl();
                deq.insertFront(34);
                deq.insertRear(45);
                deq.removeFront();
                deq.removeFront();
                deq.removeFront();
                deq.insertFront(21);
                deq.insertFront(98);
                deq.insertRear(5);
                deq.insertFront(43);
                deq.removeRear();
        }
}
```

Output:

adding at front: 34

[34]

adding at rear: 45

[34, 45]

removed from front: 34

[45]

removed from front: 45

[]

Deque underflow!! unable to remove.

adding at front: 21

[21]

adding at front: 98

[98, 21]

adding at rear: 5

[98, 21, 5]

adding at front: 43

[43, 98, 21, 5]

removed from front: 5

[43, 98, 21]

Java program for Deque using doubly linked list

```
class Node{
 //data
 int i;
 // next node in the list
 Node next;
 // previous node in the list
 Node prev;
}
public class LinkedListDeque {
 private Node head;
 private Node tail;

 static class Node{
  //data
  int i;
  // next node in the list
  Node next;
  // previous node in the list
  Node prev;
  Node(int i){
   this.i = i;
  }
  public void displayData(){
   System.out.print(i + " ");
```

```
 }
}
// constructor
public LinkedListDeque(){
 this.head = null;
 this.tail = null;
}

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

public void insertFirst(int i){
 //Create a new node
 Node newNode = new Node(i);
 // if first insertion tail should
 // also point to this node
 if(isEmpty()){
  tail = newNode;
 }else{
  head.prev = newNode;
 }
 newNode.next = head;
 head = newNode;
}
```

```
public void insertLast(int i){
 Node newNode = new Node(i);
 // if first insertion head should
 // also point to this node
 if(isEmpty()){
  head = newNode;
 }else{
  tail.next = newNode;
  newNode.prev = tail;
 }
 tail = newNode;
}

public Node removeFirst(){
 if(head == null){
  throw new RuntimeException("Deque is empty");
 }
 Node first = head;
 if(head.next == null){
 tail = null;
 }else{
  // previous of next node (new first) becomes null
  head.next.prev = null;
```

```
 }
 head = head.next;
 return first;
}

public Node removeLast(){
 if(tail == null){
  throw new RuntimeException("Deque is empty");
 }
 Node last = tail;
 if(head.next == null){
 head = null;
 }else{
  // next of previous node (new last) becomes null
  tail.prev.next = null;
 }
 tail = tail.prev;
 return last;
}

public int getFirst(){
 if(isEmpty()){
  throw new RuntimeException("Deque is empty");
 }
 return head.i;
```

```
}

public int getLast(){
 if(isEmpty()){
  throw new RuntimeException("Deque is empty");
 }
 return tail.i;
}

// Method for forward traversal
public void displayForward(){
Node current = head;
while(current != null){
current.displayData();
  current = current.next;
 }
 System.out.println("");
}

// Method to traverse and display all nodes
public void displayBackward(){
 Node current = tail;
 while(current != null){
 current.displayData();
 current = current.prev;
```

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

  public static void main(String[] args) {
    LinkedListDeque deque = new LinkedListDeque();
    //deque.getLast();
    deque.insertFirst(2);
    deque.insertFirst(1);
    deque.insertLast(3);
    deque.insertLast(4);
    deque.displayForward();
    Node node = deque.removeFirst();
    System.out.println("Node with value "+ node.i + " deleted");
    deque.displayForward();
    System.out.println("First element in the deque " + deque.getFirst());
    System.out.println("Last element in the deque " + deque.getLast());
  }
}
```

Output

1 2 3 4

Node with value 1 deleted

2 3 4

First element in the deque 2

Last element in the deque 4

9. // Recursive Java program for level

```
// order traversal of Binary Tree


/* Class containing left and right child of current
   node and key value*/
class Node {
    int data;
    Node left, right;
    public Node(int item)
    {
        data = item;
        left = right = null;
    }
}


class BinaryTree {
    // Root of the Binary Tree
    Node root;


    public BinaryTree() { root = null; }


    /* function to print level order traversal of tree*/
    void printLevelOrder()
    {
        int h = height(root);
```

```
    int i;
    for (i = 1; i <= h; i++)
        printCurrentLevel(root, i);
}

/* Compute the "height" of a tree -- the number of
nodes along the longest path from the root node
down to the farthest leaf node.*/
int height(Node root)
{
    if (root == null)
        return 0;
    else {
        /* compute  height of each subtree */
        int lheight = height(root.left);
        int rheight = height(root.right);

        /* use the larger one */
        if (lheight > rheight)
            return (lheight + 1);
        else
            return (rheight + 1);
    }
}
```

```
/* Print nodes at the current level */
void printCurrentLevel(Node root, int level)
{
    if (root == null)
        return;
    if (level == 1)
        System.out.print(root.data + " ");
    else if (level > 1) {
        printCurrentLevel(root.left, level - 1);
        printCurrentLevel(root.right, level - 1);
    }
}

/* Driver program to test above functions */
public static void main(String args[])
{
    BinaryTree tree = new BinaryTree();
    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);
    tree.root.left.left = new Node(4);
    tree.root.left.right = new Node(5);

   System.out.println("Level order traversal of
                       binary tree is ");
```

```
        tree.printLevelOrder();
    }
}
```

output

Level Order traversal of binary tree is

1 2 3 4 5

```
public class Demo{
  int rec_bin_search(int my_arr[], int left, int right, int x){
    if (right >= left){
      int mid = left + (right - left) / 2;
      if (my_arr[mid] == x)
      return mid;
      if (my_arr[mid] > x)
      return rec_bin_search(my_arr, left, mid - 1, x);
      return rec_bin_search(my_arr, mid + 1, right, x);
    }
    return -1;
  }
  public static void main(String args[]){
    Demo my_object = new Demo();
    int my_arr[] = { 56, 78, 90, 32, 45, 99, 104};
    int len = my_arr.length;
    int x = 104;
    int result = my_object.rec_bin_search(my_arr, 0, len - 1, x);
```

```
    if (result == -1)
      System.out.println("The element is not present in the array");
    else
      System.out.println("The element has been found at index " + result);
  }
}
```

Output

The element has been found at index 6

```
// Java program to print BFS traversal from a given source vertex.
// BFS(int s) traverses vertices reachable from s.
import java.io.*;
import java.util.*;

// This class represents a directed graph using adjacency list
// representation
class Graph
{
    private int V;  // No. of vertices
    private LinkedList<Integer> adj[]; //Adjacency Lists

    // Constructor
    Graph(int v)
    {
        V = v;
```

```
    adj = new LinkedList[v];
    for (int i=0; i<v; ++i)
        adj[i] = new LinkedList();
}

// Function to add an edge into the graph
void addEdge(int v,int w)
{
    adj[v].add(w);
}

// prints BFS traversal from a given source s
void BFS(int s)
{
    // Mark all the vertices as not visited(By default
    // set as false)
    boolean visited[] = new boolean[V];

    // Create a queue for BFS
    LinkedList<Integer> queue = new LinkedList<Integer>();

    // Mark the current node as visited and enqueue it
    visited[s]=true;
    queue.add(s);
```

```
    while (queue.size() != 0)
    {
        // Dequeue a vertex from queue and print it
        s = queue.poll();
        System.out.print(s+" ");

        // Get all adjacent vertices of the dequeued vertex s
        // If a adjacent has not been visited, then mark it
        // visited and enqueue it
        Iterator<Integer> i = adj[s].listIterator();
        while (i.hasNext())
        {
            int n = i.next();
            if (!visited[n])
            {
                visited[n] = true;
                queue.add(n);
            }
        }
    }
}

// Driver method to
public static void main(String args[])
{
```

```
        Graph g = new Graph(4);

        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3);

        System.out.println("Following is Breadth First Traversal "+
                           "(starting from vertex 2)");

        g.BFS(2);
    }
}
// This code is contributed by Aakash Hasija
```

Output

Following is Breadth First Traversal (starting from vertex 2)

2 0 3 1