
Contents

1. Array Queue
 2. Array Stack
 3. Binary Tree
 4. Binary Search Tree
 5. Circular Linked List
 6. Circular Queue
 7. DE' Queue
 8. Double Linked List
 9. Fixed Top Stack
 10. Linked List
 11. Linked List (recursive)
 12. Linked Queue
 13. Linked Stack
 14. Merge Sort
 15. Quick Sort
-

//Array Queue

```
class ArrayQueue
```

```
{ int a[];
  int front, rear;
```

```
    ArrayQueue(int size)
```

```
{ a=new int[size];
  front=0;
  rear=-1;
}
```

public void insert(int value)

```
{ if(rear==a.length-1)
  { System.out.println("Overflow");
  }
  else
  { rear++;
    a[rear]=value;
  }
} //insert()
```

public int delete()

```
{ if(rear==-1)
  { System.out.println("Underflow");
    return 0; //error code
  }
}
```

```
    else
    {   int temp=a[front];
        front++;
        if(front>rear)
        {   reset();
            }
        return temp;
    }
} //delete()
```

```
private void reset()
{   front=0;
    rear=-1;
}
```

```
public void fifo()//traverse
{   for(int i=front; i<=rear; i++)
    {   System.out.print(a[i]+" ");
        }
    System.out.println();
}
```

```
public static void main(String args[])
{   ArrayQueue queue=new ArrayQueue(5);
    queue.insert(1);
    queue.insert(2);
    queue.insert(3);
    queue.insert(4);
    queue.insert(5);
    queue.insert(6);
    queue.fifo();
    queue.delete();
    queue.delete();
    queue.delete();
    queue.fifo();
} //main
} //class

/*
OUTPUT
Overflow
1 2 3 4 5
4 5
*/
```

```
//Array Stack
```

```
class ArrayStack
```

```
{ int a[];
```

```
int top;
```

```
ArrayStack(int size)
```

```
{ a=new int[size];
```

```
top=-1;
```

```
}
```

```
public void push(int value)
```

```
{ if(top==a.length-1)
```

```
{ System.out.println("Overflow");
```

```
}
```

```
else
```

```
{ top++;
```

```
a[top]=value;
```

```
}
```

```
}
```

```
public int pop()
```

```
{ if(top== -1)
```

```
{ System.out.println("Underflow");
```

```
return 0; //or return an error code like -999
```

```
}
```

```
else
```

```
{ int temp=a[top];
```

```
top--;
```

```
return temp;
```

```
}
```

```
}
```

```
public void lifo() //for traversal
```

```
{ for(int i=top; i>=0; i--)
```

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

```
}
```

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

```
}
```

```
public static void main(String args[])
```

```
{
```

```
    ArrayStack stack=new ArrayStack(5);
    stack.push(1);
    stack.push(2);
    stack.push(3);
    stack.push(4);
    stack.push(5);
    stack.push(6);
    stack.lifo();
    stack.pop();
    stack.pop();
    stack.push(7);
    stack.lifo();
}
}
```

```
/*Output
Overflow
5 4 3 2 1
7 3 2 1
*/
```

```
//Binary tree
```

```
class Node
{ Node left, right;
  char data;
} //class node
```

```
public class BinaryTree
```

```
{ void displayPreorder(Node root)
{ if(root!=null)
{ System.out.println(root.data);
  displayPreorder(root.left);
  displayPreorder(root.right);
}
} //display Preorder
```

```
void displayInorder(Node root)
```

```
{ if(root!=null)
{ displayInorder(root.left);
  System.out.println(root.data);
  displayInorder(root.right);
}
} //display Inorder
```

```
void displayPostorder(Node root)
```

```
{ if(root!=null)
  { displayPostorder(root.left);
    displayPostorder(root.right);
    System.out.println(root.data);
  }
} //display Postorder
```

```
Node createRoot(char item)
```

```
{ Node root=new Node();
  root.data=item;
  root.left=root.right=null;
  return root;
} //createRoot
```

```
void create(Node root, char item, char after, char pos)
```

```
{ if(root!=null)
  { if(after==root.data)
    { Node ptr=new Node();
      ptr.data=item;
      ptr.left=ptr.right=null;
      if(pos=='l')
        root.left=ptr;
      else
        root.right=ptr;
    }
    create(root.left, item,after,pos);
    create(root.right,item,after,pos);
  } //if(root!=null)
} //create
```

```
int countNodes(Node root)
```

```
{ if(root!=null)
  { return 1+countNodes(root.left)+countNodes(root.right);
  }
  else
  { return 0;
  }
} //countNodes
```

```
public static void main(String args[])
{ BinaryTree bt=new BinaryTree();
  Node root;
```

```
    root=bt.createRoot('a');
    bt.create(root, 'b', 'a', 'l');
    bt.create(root, 'c', 'a', 'r');
    bt.create(root, 'd', 'b', 'l');
    bt.create(root, 'e', 'b', 'r');
    bt.create(root, 'f', 'c', 'l');
    bt.create(root, 'g', 'c', 'r');
    bt.create(root, 'h', 'f', 'l');
    bt.create(root, 'i', 'f', 'r');
    System.out.println("Preorder traversal");
    bt.displayPreorder(root);
    System.out.println("Number of nodes");
    System.out.println(bt.countNodes(root));
} //main
} //class Binary Tree
/*      a
      b  c
     d e f g
        h i
*/
```

//BST

```
class Node
{ Node left, right;
  int data;
} //class node

public class BST
{ void displayPreorder(Node root)
  { if(root!=null)
    { System.out.print(root.data+" ");
      displayPreorder(root.left);
      displayPreorder(root.right);
    }
  } //display Preorder
```

Node createRoot(int item)

```
{ Node root=new Node();
  root.data=item;
  root.left=root.right=null;
  return root;
} //createRoot
```



```
        { root=null;
        }
        else if(root.right==null)
        { root=root.left;
        }
        else if(root.left==null)
        { root=root.right;
        }
        else
        { root.data=root.right.data;
          delete(root, root.data); //will check the other occurrence
        }
    } //deleteRoot

public static void main(String args[])
{   BST bst=new BST();
    Node root;
    root=bst.createRoot(5);
    bst.insert(root,3);
    bst.insert(root,4);
    bst.insert(root,8);
    bst.insert(root,9);
    bst.insert(root,7);
    System.out.println("Preorder traversal");
    bst.displayPreorder(root);
    System.out.println();
    System.out.println(bst.search(root, 9));
    System.out.println(bst.search(root, 1));
    bst.delete(root, 8);
    bst.displayPreorder(root);
    bst.deleteRoot(root);
    System.out.println();
    bst.displayPreorder(root);
} //main
} //class BST
/*
Preorder traversal
5 3 4 8 7 9
true
false
5 3 4 9 7
9 3 4 7
*/
```

```
// CircularLinkedList
```

```
class Node
{   public int info;
    public Node next;
}
```

```
public class CircularLinkedList
{   Node start;
    CircularLinkedList()
    {   start=null;
    }
}
```

```
void create()
```

```
{   Node ptr=null;
    for(int i=1; i<=5; i++)//5 nodes
    {   if(i==1) {start=ptr=new Node(); }
        else { ptr=ptr.next; }
        ptr.info=i;//input if reqd
        if(i==5) ptr.next=start;
        else ptr.next=new Node();
    }
} //create()
```

```
void display()
```

```
{   Node ptr=start;
    do
    {   System.out.print(ptr.info+" ");
        ptr=ptr.next;
    }while(ptr!=start);
} //display
```

```
public static void main(String args[])
{   CircularLinkedList obj=new CircularLinkedList();
    obj.create();
    obj.display();
}
} //class
```

```
/* Output
1 2 3 4 5
*/
```

//Circular Queue

```
class CircularQueue
```

```
{ int a[];
  int front, rear;
```

```
    CircularQueue(int size)
```

```
{ a=new int[size];
  front=-1;
  rear=-1;
}
```

public void insert(int value)

```
{ if(front==0 && rear==a.length-1 || front==rear+1)
  { System.out.println("Overflow");
    return;
  }
  else if(front==-1)
  { front=rear=0;
  }
  else if(rear==a.length-1)
  { rear=0;
  }
  else
  { rear++;
  }
  a[rear]=value;
} //insert()
```

public int delete()

```
{ if(front==-1)
  { System.out.println("Underflow");
    return 0; //or return an error code like -999
  }
  else
  { int temp=a[front];
    if(front==rear)
    { reset();
    }
    else if(front==a.length-1)
    { front=0;
    }
    else
    { front++;
    }
  }
}
```

```
        return temp;
    }
} //delete()
```

```
private void reset()
{   front=-1;
    rear=-1;
}
```

```
public void traverse()
{   for(int i=front; ; i++)
    {   if(i==a.length)
        {   i=0;
            }
        System.out.print(a[i]+" ");
        if(i==rear)
            {   break;
                }
            }
        System.out.println();
    }
```

```
public static void main(String args[])
{   CircularQueue cq=new CircularQueue(5);
    cq.insert(1);
    cq.insert(2);
    cq.insert(3);
    cq.insert(4);
    cq.insert(5);
    cq.insert(6);
    cq.traverse();
    cq.delete();
    cq.delete();
    cq.delete();
    cq.insert(6);
    cq.insert(7);
    cq.traverse();
} //main
} //class
```

/*

OUTPUT
Overflow
1 2 3 4 5

4 5
*/

```
public class DQueue
```

```
{ int a[], elements, front, rear;  
  DQueue()  
  { a=new int[5];  
    elements=0;  
    front=-1;  
    rear=-1;  
  }  
}
```

```
void pushFront(int value)
```

```
{ if(elements==a.length)  
  { System.out.println("Overflow");  
  }  
  else  
  { if(front==0)  
    { System.out.println("Front full");  
    }  
    else  
    { elements++;  
      front--;  
      a[front]=value;  
    }  
  }  
}  
} //pushFront
```

```
int popFront()
```

```
{ int temp=0;  
  if(elements==0)  
  { System.out.println("Underflow");  
  }  
  else  
  { temp=a[front];  
    front++;  
    elements--;  
    if(elements==0)//reset  
    { front=-1;  
      rear=-1;  
    }  
  }  
  return temp;  
} //popFront
```

```
void pushRear(int value)
```

```
{ if(elements==a.length)
  { System.out.println("Overflow");
  }
  else
  { elements++;
    rear++;
    if(front==-1)
    { front=0;
    }
    a[rear]=value;
  }
} //pushRear
```

```
int popRear()
```

```
{ int temp=0;
  if(elements==0)
  { System.out.println("Underflow");
  }
  else
  { temp=a[rear];
    rear--;
    elements--;
    if(elements==0)//reset
    { front=-1;
      rear=-1;
    }
  }
  return temp;
} //popRear
```

```
void display()
```

```
{ for(int i=front; i<=rear; i++)
  { System.out.print(a[i]+" ");
  }
  System.out.println();
}
```

```
public static void main(String args[])
```

```
{ DQueue dq=new DQueue();
  dq.pushRear(1);
  dq.pushRear(2);
  dq.pushRear(3);
```

```
    dq.display();
    dq.popFront();
    dq.display();
    dq.pushFront(4);
    dq.display();
    dq.popRear();
    dq.display();
    dq.pushRear(5);
    dq.display();
} //main
} //class
```

```
/* OUTPUT
```

```
1 2 3
2 3
4 2 3
4 2
4 2 5
*/
```

```
//Doubly Linked List
```

```
import java.io.*;
class Node
{ int info;
  Node prev, next;
}
public class DoublyLinkedList
{ Node start;
  DoublyLinkedList()
  { start=null;
  }
}
```

```
void create()
```

```
{ Node ptr=null;
  for(int i=1; i<=10; i++)//creation of 10 nodes
  { if(i==1)
    { start=ptr=new Node();
      start.prev=null;
    }
    else
    { Node save=ptr;
      ptr=ptr.next;
      ptr.prev=save;
    }
  }
}
```

```
    }
    ptr.info=(int)(Math.random()*100);
    if(i==10) ptr.next=null;
    else ptr.next=new Node();
  }
} //create
```

```
void display()
```

```
{ for(Node temp=start; temp!=null; temp=temp.next)
  { System.out.print(temp.prev+"\t");
    System.out.print(temp+"\t");
    System.out.println(temp.next+" ");
  }
  System.out.println();
} //display
```

```
public static void main()
{ DoublyLinkedList obj=new DoublyLinkedList();
  obj.create();
  obj.display();
}
} //class
```

```
/* output
```

```
null Node@17ba38f Node@2f0d54
Node@17ba38f Node@2f0d54Node@1142196
Node@2f0d54Node@1142196 Node@a9255c
Node@1142196 Node@a9255cNode@d3c6a3
Node@a9255cNode@d3c6a3Node@1961581
Node@d3c6a3Node@1961581 Node@5ddb6e
Node@1961581 Node@5ddb6e Node@1f1235b
Node@5ddb6e Node@1f1235b Node@4865ce
Node@1f1235b Node@4865ceNode@113beb5
Node@4865ceNode@113beb5 null
```

```
*/
```

```
//Fixed Top Stack Example with shifting values
```

```
class FixedTopStack
{ int a[];
  int elements;

  FixedTopStack(int size)
```

```
{ a=new int[size];
  elements=0;
}
```

```
public void push(int value)
```

```
{ if(elements==a.length)
  { System.out.println("Overflow");
  }
  else
  { elements++;
    for(int i=elements-1; i>0; i--)
    { a[i]=a[i-1];
    }
    a[0]=value;
  }
}
```

```
public int pop()
```

```
{ if(elements==0)
  { System.out.println("Underflow");
    return 0;
  }//id
  else
  { elements--;
    int temp=a[0];
    for(int i=0; i<elements; i++)
    { a[i]=a[i+1];
    }//for
    return temp;
  }//else
}
```

```
public void lifo() //for traversal
```

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

```
public static void main(String args[])
```

```
{
  FixedTopStack stack=new FixedTopStack(5);
  stack.push(1);
  stack.push(2);
}
```

```
    stack.push(3);
    stack.push(4);
    stack.push(5);
    stack.push(6);
    stack.lifo();
    stack.pop();
    stack.pop();
    stack.push(7);
    stack.lifo();
} //main
} //class
```

```
/*Output
Overflow
5 4 3 2 1
7 3 2 1
*/
```

//Linked List

```
class Node
{   public int info;
    public Node next;
}
public class LinkedList
{   Node start;
    LinkedList()
    {   start=null;
    }
}
```

void create()

```
{   Node ptr=null;
    for(int i=1; i<=5; i++)//creation
    {   if(i==1) start=ptr=new Node();
        else ptr=ptr.next;
        ptr.info=i;
        if(i==5) ptr.next=null;
        else ptr.next=new Node();
    }
} //create
```

void display()

```
{   Node ptr=start;
    while(ptr!=null)
```

```
    { System.out.print(ptr.info+" ");
      ptr=ptr.next;
    }
    System.out.println();
} //display
```

void insert()

```
{ Node ptr=null;
  int value=99, pos=3;//input
  int ctr=0;
  //Insert as first node
  if(pos==1)
  { Node temp=new Node();
    temp.info=value;
    temp.next=start;
    start=temp;
  }
  //Insert somewhere in between the list
  for(ptr=start;ptr!=null;ptr=ptr.next)//traversal
  { ctr++;
    if(ctr==pos-1)
    { Node temp=new Node();
      temp.info=value;
      temp.next=ptr.next;
      ptr.next=temp;
    }
  }
} //insert
```

void delete()

```
{ Node ptr=null;
  int pos=3; //input
  int ctr=0;
  //Delete the first node
  if(pos==1)
  { start=start.next;
  }
  //delete somewhere in between the list
  for(ptr=start;ptr!=null;ptr=ptr.next)//traversal
  { ctr++;
    if(ctr==pos-1)
    { ptr.next=ptr.next.next;
    }
  }
}
```

```
 }//delete
```

```
void concat(LinkedList List2)
```

```
{ Node ptr=null;
  for(ptr=start;ptr!=null;ptr=ptr.next)//traversal
  { if(ptr.next==null)
    { ptr.next=List2.start;
      break; //required
    }
  }
}
}//concat
```

```
Node split()
```

```
{ Node start2=null, ptr=null;
  int splitPos=5; //input
  int ctr=0;
  for(ptr=start;ptr!=null;ptr=ptr.next)//traversal
  { ctr++;
    if(ctr==splitPos)
    { start2=ptr.next;
      ptr.next=null;
      break;
    }
  }
  return start2;
}
}//split
```

```
void reverse()
```

```
{ Node ptr=null, pptr=start, nptr=start.next;
  start.next=null;
  while(nptr!=null)
  { ptr=nptr;
    nptr=ptr.next;
    ptr.next=pptr;
    pptr=ptr;
  }
  start=ptr;
}
}//reverse
```

```
void swapFirstLast()
```

```
{ Node first, second, last, secondLast;
  first=second=last=secondLast=null;
  first=start;
  second=first.next;
```

```
for(Node ptr=start; ptr.next!=null; ptr=ptr.next)//run till sec last node
{ if(ptr.next.next==null) //if second last node
  { secondLast=ptr;
    last=ptr.next;
  }
}
last.next=second;
secondLast.next=first;
first.next=null;
start=last;
} //swapFirstLast
```

```
void swap2nodes(int node1, int node2)
```

```
{ int count=0;
  Node first, beforeFirst, second, beforeSecond, temp;
  first=beforeFirst=second=beforeSecond=temp=null;
  for(Node ptr=start; ptr!=null; ptr=ptr.next)
  { count++;
    if(count==node1-1) beforeFirst=ptr;
    if(count==node1) first=ptr;
    if(count==node2-1) beforeSecond=ptr;
    if(count==node2) second=ptr;
  }
  beforeFirst.next=second;
  temp=first.next;
  first.next=second.next;
  second.next=temp;
  beforeSecond.next=first;
} //swap2nodes
```

```
void sort()
```

```
{ for(Node ptr1=start; ptr1!=null; ptr1=ptr1.next)
  { for(Node ptr2=start; ptr2.next!=null; ptr2=ptr2.next)
    { if(ptr1.info<ptr2.info)
      { int t=ptr1.info; ptr1.info=ptr2.info; ptr2.info=t;
        } //if
      } //inner
    } //outer
  } //sort
public static void main(String agrs[])
{ LinkedList obj=new LinkedList();
  LinkedList obj2=new LinkedList();
  obj.create();
  obj.display();
```

```
    obj.insert();
    obj.display();
    obj.delete();
    obj.display();
    obj2.create();
    obj.concat(obj2);
    obj.display();
    obj.split();
    obj.display();
    obj.reverse();
    obj.display();
    obj.swapFirstLast();
    obj.display();
    obj.swap2nodes(2,4);
    obj.display();
    obj.reverse();
    obj.display();
    obj.sort();
    obj.display();
} //main
} //class IList
```

```
/* Output
1 2 3 4 5
1 2 99 3 4 5
1 2 3 4 5
1 2 3 4 5 1 2 3 4 5
1 2 3 4 5
5 4 3 2 1
1 4 3 2 5
1 2 3 4 5
5 4 3 2 1
1 2 3 4 5
*/
```

//Linked List (using recursion)

```
/*SEE
    DISPLAY(),
    SUM() AND
    COUNT()
*/
class Node
```

```
{ public int info;  
  public Node next;  
}
```

```
public class LinkedListRec  
{ Node start;  
  LinkedListRec()  
  { start=null;  
  }  
}
```

```
Node create()
```

```
{ Node ptr=null;  
  int nodes=(int)(Math.random()*10);  
  for(int i=1; i<=nodes; i++)//creation  
  { if(i==1) start=ptr=new Node();  
    else ptr=ptr.next;  
    ptr.info=(int)(Math.random()*100);  
    if(i==nodes) ptr.next=null;  
    else ptr.next=new Node();  
  }  
  return start;  
}//create
```

```
void display(Node ptr)
```

```
{ System.out.print(ptr.info+" ");  
  if(ptr.next!=null)  
  { display(ptr.next);  
  }  
}//display
```

```
int sum(Node ptr)
```

```
{ if(ptr.next==null)  
  { return ptr.info;  
  }  
  else  
  { return ptr.info+sum(ptr.next);  
  }  
}//sum
```

```
int count(Node ptr)
```

```
{ if(ptr.next==null)  
  { return 1;  
  }  
  else
```

```
        { return 1+count(ptr.next);
        }
    }//count

    public static void main()
    {   LinkedListRec obj=new LinkedListRec();
        Node list1=new Node();
        list1=obj.create();
        obj.display(list1);
        System.out.println();
        System.out.println("Number of nodes = "+obj.count(list1));
        System.out.println("Sum of nodes = "+obj.sum(list1));
    }//main()
}//class

/* OUTPUT
41 18
Number of nodes = 2
Sum of nodes = 59
*/
```

//Linked Queue

```
class LinkedListQueue
{   class Node
    {   int info;
        Node next;
    }
    Node front, rear;

    LinkedListQueue()
    {   front=null;
        rear=null;
    }
}
```

void insert(int n)

```
{   if(rear==null && front==null)
    {   rear=front=new Node();
        if(front==null)
            System.out.println("Overflow");
        else
            {   rear.info=n;
                rear.next=null;
            }
    }
}
```



```
        queue.delete();
        queue.delete();
        queue.insert(7);
        queue.fifo();
    } //main
} //class LinkedQueue
```

```
/*OUTPUT
1 2 3 4 5 6
3 4 5 6 7
*/
```

//Linked Stack

```
class Node
{   int info;
    Node next;
}
public class LinkedStack
{   Node top;
    LinkedStack()
    {   top=null;
    }
}
```

void push(int n)

```
{   Node temp=new Node();
    if(temp==null)
    {   System.out.println("Overflow");
        return;
    }
    temp.info=n;
    temp.next=null;
    if(top==null) top=temp;
    else
    {   temp.next=top;
        top=temp;
    }
} //push
```

int pop()

```
{   if(top==null)
    {   System.out.println("Underflow");
        return 0;
    }
}
```

```
    int temp=top.info;
    top=top.next;
    return temp;
} //pop
```

```
void lifo() //traverse
```

```
{ Node ptr=top;
  while(ptr!=null)
  { System.out.print(ptr.info+" ");
    ptr=ptr.next;
  }
  System.out.println();
}
```

```
public static void main()
{   LinkedStack stack=new LinkedStack();
    stack.push(5);
    stack.push(6);
    stack.push(7);
    stack.lifo();
    stack.pop();
    stack.pop();
    stack.push(5);
    stack.lifo();
}
} //class IStack
```

```
/* OUTPUT
7 6 5
5 5
*/
```

```
//Merge Sort
```

```
class MergeSort
{   public static void main()
    {   MergeSort obj=new MergeSort();
        int a[]={38, 27, 43, 3, 9, 82, 10};
        for(int i=0; i<a.length; i++) System.out.print(a[i]+" "); System.out.println();
        obj.mergeSort(a);
        for(int i=0; i<a.length; i++) System.out.print(a[i]+" "); System.out.println();
    } //main
    void mergeSort(int array[])
```

```
{ if(array.length > 1)
  { int elementsInA1 = array.length/2;
    int elementsInA2 = array.length-elementsInA1;
    int arr1[] = new int[elementsInA1];
    int arr2[] = new int[elementsInA2];
    for(int i = 0; i < elementsInA1; i++) arr1[i] = array[i];
    for(int i = 0; i < elementsInA2; i++) arr2[i] = array[elementsInA1+i];
    mergeSort(arr1);
    mergeSort(arr2);
    int i = 0, j = 0, k = 0;
    while(arr1.length != j && arr2.length != k)
      { if(arr1[j] < arr2[k]) array[i++] = arr1[j++];
        else array[i++] = arr2[k++];
      }
    while(arr1.length != j) array[i++] = arr1[j++];
    while(arr2.length != k) array[i++] = arr2[k++];
  }
}
//if
//return array;
}
//mergeSort()
}
//class
```

//Quick sort

```
public class QuickSort
{ public static void main()
  { int a[]={5,7,4,3,6,8,9,1,2,0};
    QuickSort obj=new QuickSort();
    for(int i=0; i<a.length; i++) System.out.print(a[i]+" "); System.out.println();
    obj.quickSort(a,0,a.length-1);
    for(int i=0; i<a.length; i++) System.out.print(a[i]+" "); System.out.println();
  }
void quickSort(int array[], int start, int end)
{ int i = start;
  int k = end;
  if (end - start >= 1)
  { int pivot = array[start];
    while (k > i)
    { while (array[i] <= pivot && i <= end && k > i) i++;
      while (array[k] > pivot && k >= start && k >= i) k--;
      if (k > i) swap(array, i, k);
    }
    swap(array, start, k);
    quickSort(array, start, k - 1);
    quickSort(array, k + 1, end);
  }
}
```

```
    }  
  }//quickSort()  
void swap(int array[], int index1, int index2)  
{  int temp = array[index1];  
    array[index1] = array[index2];  
    array[index2] = temp;  
  }//swap()  
}//class
```

END
