package com.labQ1;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner input = new Scanner(System.***in***);

System.***out***.println("how many numbers do you want to enter");

int n= input.nextInt();

int[] nums =new int[n];

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

nums[i]=input.nextInt();

}

System.***out***.println("what number do you what to search and count");

int num =input.nextInt();

int result=*Search*(nums, num);

System.***out***.println("the number " + num + " is occurred in array " + result+" times");

*mergeSort*(nums);

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

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

}

}

static int Search(int[] nums, int num) {

int count=0;

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

if(nums[i]== num) {

count++;

}

}

return count;

}

public static void mergeSort(int[] array) {

if (array.length <= 1) {

return;

}

int mid = array.length / 2;

int[] leftArray = new int[mid];

int[] rightArray = new int[array.length - mid];

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

leftArray[i] = array[i];

}

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

rightArray[i - mid] = array[i];

}

*mergeSort*(leftArray);

*mergeSort*(rightArray);

*merge*(array, leftArray, rightArray);

}

public static void merge(int[] array, int[] leftArray, int[] rightArray) {

int leftLength = leftArray.length;

int rightLength = rightArray.length;

int i = 0, j = 0, k = 0;

while (i < leftLength && j < rightLength) {

if (leftArray[i] <= rightArray[j]) {

array[k++] = leftArray[i++];

}

else {

array[k++] = rightArray[j++];

}

}

while (i < leftLength) {

array[k++] = leftArray[i++];

}

while (j < rightLength) {

array[k++] = rightArray[j++];

}

}

}

Question 2

package com.labQ2;  
class Main{  
 public static void main(String[] args) {  
 Stack stack = new Stack(10);  
 stack.pop();  
 stack.push(1);  
 stack.push(2);  
 stack.push(3);  
 stack.push(4);  
  
 stack.pop();  
 System.out.println("\nAfter popping out");  
 stack.showStack();  
 }  
   
}

package com.labQ2;  
public class Stack {  
 int arr[];  
 int top;  
 int capacity;  
 Stack(int size){  
 capacity=size;  
 arr = new int[capacity];  
 top=-1;  
 }  
 boolean isFull() {  
 return top== capacity-1;  
 }  
 boolean isEmpty() {  
 return top ==-1;  
 }  
 void push(int x) {  
 if(isFull()) {  
 System.out.println("Overflow\n Terminate the program");  
 }  
 else {  
 arr[++top]=x;  
 System.out.println("Inserted "+x);  
 }  
  
  
 }  
 int pop(){  
 if(isEmpty()) {  
 System.out.println("Stack empty");  
   
 }  
 else{  
 return arr[top--];  
 }  
 return -1;  
  
 }  
 int size() {  
 return top+1;  
 }  
 void showStack() {  
 for(int i= 0; i<= top; i++) {  
 System.out.print(arr[i]+" ");  
 }  
 }  
  
}

Question 3

package com.labQ3;  
  
public class Main {  
 public static void main(String[] args) {  
 QueueUsingTwoStacks queue = new QueueUsingTwoStacks(5);  
  
 queue.enqueue(1);  
 queue.enqueue(2);  
 queue.enqueue(3);  
  
 System.out.println("Peek: " + queue.peek());  
  
 System.out.println("Dequeue: " + queue.dequeue());   
 System.out.println("Dequeue: " + queue.dequeue());   
  
 queue.enqueue(4);  
 queue.enqueue(5);  
  
 System.out.println("Peek: " + queue.peek());   
  
 System.out.println("Dequeue: " + queue.dequeue());   
 System.out.println("Dequeue: " + queue.dequeue());   
 System.out.println("Dequeue: " + queue.dequeue());   
 System.out.println("Dequeue: " + queue.dequeue());   
 }  
}

package com.labQ3;  
class Stack {  
 private int maxSize;  
 private int[] stackArray;  
 private int top;  
  
 public Stack(int size) {  
 maxSize = size;  
 stackArray = new int[maxSize];  
 top = -1;  
 }  
  
 public void push(int value) {  
 if (isFull()) {  
 System.out.println("Stack is full, cannot push element");  
 return;  
 }  
 stackArray[++top] = value;  
 }  
  
 public int pop() {  
 if (isEmpty()) {  
 System.out.println("Stack is empty, cannot pop element");  
 return -1;  
 }  
 return stackArray[top--];  
 }  
  
 public int peek() {  
 if (isEmpty()) {  
 System.out.println("Stack is empty, cannot peek element");  
 return -1;  
 }  
 return stackArray[top];  
 }  
  
 public boolean isEmpty() {  
 return (top == -1);  
 }  
  
 public boolean isFull() {  
 return (top == maxSize - 1);  
 }  
}  
  
class QueueUsingTwoStacks {  
 private Stack stack1;  
 private Stack stack2;  
  
 public QueueUsingTwoStacks(int capacity) {  
 stack1 = new Stack(capacity);  
 stack2 = new Stack(capacity);  
 }  
  
 public void enqueue(int value) {  
 while (!stack1.isEmpty()) {  
 stack2.push(stack1.pop());  
 }  
 stack1.push(value);  
 while (!stack2.isEmpty()) {  
 stack1.push(stack2.pop());  
 }  
 }  
  
 public int dequeue() {  
 if (stack1.isEmpty()) {  
 System.out.println("Queue is empty, cannot dequeue element");  
 return -1;  
 }  
 return stack1.pop();  
 }  
  
 public int peek() {  
 if (stack1.isEmpty()) {  
 System.out.println("Queue is empty, cannot peek element");  
 return -1;  
 }  
 return stack1.peek();  
 }  
}

package com.labQ3;  
class Stack {  
 private int maxSize;  
 private int[] stackArray;  
 private int top;  
  
 public Stack(int size) {  
 maxSize = size;  
 stackArray = new int[maxSize];  
 top = -1;  
 }  
  
 public void push(int value) {  
 if (isFull()) {  
 System.out.println("Stack is full, cannot push element");  
 return;  
 }  
 stackArray[++top] = value;  
 }  
  
 public int pop() {  
 if (isEmpty()) {  
 System.out.println("Stack is empty, cannot pop element");  
 return -1;  
 }  
 return stackArray[top--];  
 }  
  
 public int peek() {  
 if (isEmpty()) {  
 System.out.println("Stack is empty, cannot peek element");  
 return -1;  
 }  
 return stackArray[top];  
 }  
  
 public boolean isEmpty() {  
 return (top == -1);  
 }  
  
 public boolean isFull() {  
 return (top == maxSize - 1);  
 }  
}  
  
class QueueUsingTwoStacks {  
 private Stack stack1;  
 private Stack stack2;  
  
 public QueueUsingTwoStacks(int capacity) {  
 stack1 = new Stack(capacity);  
 stack2 = new Stack(capacity);  
 }  
  
 public void enqueue(int value) {  
 while (!stack1.isEmpty()) {  
 stack2.push(stack1.pop());  
 }  
 stack1.push(value);  
 while (!stack2.isEmpty()) {  
 stack1.push(stack2.pop());  
 }  
 }  
  
 public int dequeue() {  
 if (stack1.isEmpty()) {  
 System.out.println("Queue is empty, cannot dequeue element");  
 return -1;  
 }  
 return stack1.pop();  
 }  
  
 public int peek() {  
 if (stack1.isEmpty()) {  
 System.out.println("Queue is empty, cannot peek element");  
 return -1;  
 }  
 return stack1.peek();  
 }  
}  
questi0n 4

package com.labQ4;  
class Mmain{  
 public static void main(String[] args) {  
 char[] chars = {'a','A','B','C','d'};  
 for(int i=0; i<chars.length; i++) {  
 System.out.print(chars[i]+" ");  
 }  
 }  
static void bubbleSort(int array[]) {  
 int size = array.length;  
 for (int i = 0; i < size - 1; i++)  
 for (int j = 0; j < size - i - 1; j++)  
  
   
 if (array[j] > array[j + 1]) {  
  
 // swapping occurs if elements  
 // are not in the intended order  
 int temp = array[j];  
 array[j] = array[j + 1];  
 array[j + 1] = temp;  
 }  
 }  
}

Question 6

package com.labQ6;  
class Main{  
 public static void main(String[] args) {  
 int[] array= {3, 7, 1, 9, 4};  
 int indexToDelete=7;  
 deletedElement(array,indexToDelete);  
 }  
  
 private static int[] deletedElement(int[] array, int indexToDelete) {  
 int[] newArray = new int[array.length - 1];   
 if(indexToDelete>0 && indexToDelete <array.length) {  
 for (int i = 0, j = 0; i < array.length; i++) {   
 if (i != indexToDelete) {   
 newArray[j++] = array[i];   
 }  
 }  
 }  
 else{  
 new IllegalStateException("Invalid index " + array);   
 return newArray;  
 }  
  
}