Q1.

```
class SearchInArray{

    public static int linearSearch(int arr[], int key){
        for (int i = 0; i < arr.length; i++) {
            if(arr[i]==key)
                return i;
        }
        return -1;
    }

    public static int binarySearch(int arr[], int key){
        int s=0;
        int e=arr.length;
        while (s<=e) {
            int mid=s+(e-s)/2;
            if(arr[mid]==key)
                return mid;
            if(arr[mid]<key)
                s=mid+1;
            else
                e=mid-1;
        }
        return -1;
    }
    public static void main(String[] args) {
        int arr[]={1,2,3,4,5};
        System.out.println(linearSearch(arr, 5));
        System.out.println(binarySearch(arr, 4));
    }
}
```

Q2.

```
class BinarySearch{
 int binarySearch(int arr[], int l, int r, int x)
 {
  while (l <= r) {
   int mid = (l + r) / 2;


   if (arr[mid] == x) {
    return mid;

   } else if (arr[mid] > x) {
    r = mid - 1;


   } else {
   l = mid + 1;
   }
  }


  return -1;
 }
 public static void main(String args[])
 {
  BinarySearch ob = new BinarySearch();

  int arr[] = { 21, 33, 4, 10, 40 };
  int n = arr.length;
  int x = 10;
  int result = ob.binarySearch(arr, 0, n - 1, x);

  if (result == -1)
   System.out.println("Element not present");
  else
   System.out.println("Element found at index "
       + result);
 }
}
```

Q3.

```
public class SortElementsByFrequency {
    public static List<Integer> sortByFrequency(int[] arr) {
        Map<Integer, Integer> frequencyMap = new HashMap<>();

        for (int num : arr) {
            frequencyMap.put(num, frequencyMap.getOrDefault(num, 0) + 1);
        }

        List<Integer> sortedList = new ArrayList<>(frequencyMap.keySet());
        Collections.sort(sortedList, (a, b) -> {
            int freqCompare = frequencyMap.get(b).compareTo(frequencyMap.get(a));
            if (freqCompare != 0) {
                return freqCompare;
            } else {
                return a.compareTo(b);
            }
        });

        return sortedList;
    }

    public static void main(String[] args) {
        int[] arr = {3, 3, 1, 1, 1, 5, 5, 5, 5, 2, 2, 4, 4, 4, 4, 4};
        List<Integer> sortedList = sortByFrequency(arr);
        System.out.println("Sorted elements by frequency: " + sortedList);
    }
}
```

Q4.

```
class SortArray{

    public static void sort012(int[] arr) {
        int low = 0;
        int high = arr.length - 1;
        int mid = 0;
        int temp;
        while (mid <= high) {
            switch (arr[mid]) {
                case 0: {
                    temp = arr[low];
                    arr[low] = arr[mid];
                    arr[mid] = temp;
                    low++;
                    mid++;
                    break;
                }
                case 1: {
                    mid++;
                    break;
                }
                case 2: {
                    temp = arr[mid];
                    arr[mid] = arr[high];
                    arr[high] = temp;
                    high--;
                    break;
                }
            }
        }
    }
    public static void main(String[] args) {
        int arr[]={0,2,0,1,2,1,0,2};
        sort012(arr);
        System.out.println(Arrays.toString(arr));
    }
}
```

Q5.

```
class ParenthesisBalancing{

    public static boolean parenthesisBalancing(String s){
        if(s.length()==0 || s.length()==1)
            return false;

        Stack<Character> st=new Stack<>();
        for (int i = 0; i < s.length(); i++) {
            char ch=s.charAt(i);
            if(ch==’(’)
                st.push(’)’);
            else if(ch==’[’)
                st.push(’]’);
            else if(ch==’{’)
                st.push(’}’);
            else if(st.isEmpty() || st.pop()!=ch)
                return false;

        }

        return true;
    }
    public static void main(String[] args) {
        String s="({})";
        System.out.println(parenthesisBalancing(s));
    }
}
```

Q.6

```
public class StackImplementation {
    private int maxSize;
    private int[] stackArray;
    private int top;

    public StackImplementation(int size) {
        this.maxSize = size;
        this.stackArray = new int[maxSize];
        this.top = -1;
    }

    public void push(int value) {
        if (top + 1 < maxSize) {
            stackArray[++top] = value;
        } else {
            System.out.println("Stack Overflow");
        }
    }

    public int pop() {
        if (!isEmpty()) {
            return stackArray[top--];
        } else {
            throw new EmptyStackException();
        }
    }

    public int peek() {
        if (!isEmpty()) {
            return stackArray[top];
        } else {
            throw new EmptyStackException();
        }
    }

    public boolean isEmpty() {
        return (top == -1);
    }
}
```

Q7.

```
public class Queue {
 int size =5;
 int Q[]=new int[size];
 int rear, front;
 public Queue() {
  front= rear= -1;
 }boolean isEmpty(){
  if(front==-1)
   return true;
  else
   return false;
 }boolean isfull(){
  if(front==-1 && rear == size-1)
   return true;
  else
   return false;
 }
 void enqueue(int x) {
  if(isfull()) {
   System.out.println("Queue is full.");
  }else {
   if(front == -1)
    front = 0;
   rear++;
   Q[rear]=x;
   System.out.println(x+" : Inserted ");
  }
 }int dequeue(){
  int x;
  if (isEmpty()){
   System.out.println("Queue is Empty !");
   return -1;
  }else {
   x=Q[front];
   if(front >= rear) {
    front =-1;
    rear  =-1;
   }else {
    front++;
   }System.out.println(x+" : Deleted");
   return x;
  }
 }
 void display() {
  if(isEmpty()) {
   System.out.println("Queue is Empty !");
  }else {
   for(int i =front;i<=rear;i++) {
    System.out.println(Q[i]);
   }
  }
}
```

Q8.

```
class DequeDemo {
    int capacity;
    int[] deque;
    int front, rear;
```

```
public DequeDemo(int capacity) {
    this.capacity = capacity;
    deque = new int[capacity];
    front = -1;
    rear = 0;
}

boolean isEmpty() {
    return front == -1;
}

boolean isFull() {
    return (front == 0 && rear == capacity - 1) || front == rear + 1;
}

void insertFront(int key) {
    if (isFull()) {
        System.out.println("Deque is full");
        return;
    }

    if (front == -1) {
        front = 0;
        rear = 0;
    } else if (front == 0) {
        front = capacity - 1;
    } else {
        front = front - 1;
    }
    deque[front] = key;
    System.out.println(key + " inserted at front");
}

void insertRear(int key) {
    if (isFull()) {
        System.out.println("Deque is full");
        return;
    }

    if (front == -1) {
        front = 0;
        rear = 0;
    } else if (rear == capacity - 1) {
        rear = 0;
    } else {
        rear = rear + 1;
    }
    deque[rear] = key;
    System.out.println(key + " inserted at rear");
}

void deleteFront() {
    if (isEmpty()) {
        System.out.println("Deque is empty");
```

```
            return;
        }

        if (front == rear) {
            front = -1;
            rear = -1;
        } else if (front == capacity - 1) {
            front = 0;
        } else {
            front = front + 1;
        }
        System.out.println("Front element deleted");
    }

    void deleteRear() {
        if (isEmpty()) {
            System.out.println("Deque is empty");
            return;
        }

        if (front == rear) {
            front = -1;
            rear = -1;
        } else if (rear == 0) {
            rear = capacity - 1;
        } else {
            rear = rear - 1;
        }
        System.out.println("Rear element deleted");
    }

    void display() {
        if (isEmpty()) {
            System.out.println("Deque is empty");
            return;
        }

        int i = front;
        while (true) {
            System.out.print(deque[i] + " ");
            if (i == rear) break;
            i = (i + 1) % capacity;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        DequeDemo dq = new DequeDemo(5);
        dq.insertRear(10);
        dq.insertRear(20);
        dq.insertFront(5);
        dq.insertFront(15);
        dq.display();

        dq.deleteRear();
        dq.deleteFront();
        dq.display();
    }
}
```

Q9.

```
class StackUsingQueues {
    private Queue<Integer> q1 = new LinkedList<>();
    private Queue<Integer> q2 = new LinkedList<>();
    private int top;

    public void push(int x) {
        q2.add(x);
        top = x;
        while (!q1.isEmpty()) {
            q2.add(q1.remove());
        }
        Queue<Integer> temp = q1;
        q1 = q2;
        q2 = temp;
    }

    public int pop() {
        int popValue = q1.remove();
        if (!q1.isEmpty()) {
            top = q1.peek();
        }
        return popValue;
    }

    public int peek() {
        return top;
    }

    public boolean empty() {
        return q1.isEmpty();
    }
}
```

Q.10

```
class QueueUsingStacks {
    private Stack<Integer> stack1 = new Stack<>();
    private Stack<Integer> stack2 = new Stack<>();

    public void enqueue(int x) {
        stack1.push(x);
    }
    public int dequeue() {
        if (stack2.isEmpty()) {
            while (!stack1.isEmpty()) {
                stack2.push(stack1.pop());
            }
        }
        return stack2.pop();
    }

    public int peek() {
        if (stack2.isEmpty()) {
            while (!stack1.isEmpty()) {
                stack2.push(stack1.pop());
            }
        }
        return stack2.peek();
    }

    public boolean empty() {
        return stack1.isEmpty() && stack2.isEmpty();
    }
}
```