```
//merge sort
public class MergeSort {

    public static void mergeSort(int[] array) {
        if (array.length > 1) {
            int mid = array.length / 2;
            int[] left = Arrays.copyOfRange(array, 0, mid);
            int[] right = Arrays.copyOfRange(array, mid, array.length);

            mergeSort(left);
            mergeSort(right);

            merge(array, left, right);
        }
    }

    public static void merge(int[] array, int[] left, int[] right) {
        int i = 0, j = 0, k = 0;
        while (i < left.length && j < right.length) {
            if (left[i] <= right[j]) {
                array[k] = left[i];
                i++;
            } else {
                array[k] = right[j];
                j++;
            }
            k++;
        }

        while (i < left.length) {
            array[k] = left[i];
            i++;
            k++;
        }

        while (j < right.length) {
            array[k] = right[j];
            j++;
            k++;
        }
    }

    public static void main(String[] args) {
        int[] array = { 3, 2, 1, 4, 5 };
        mergeSort(array);
        System.out.println(Arrays.toString(array));
    }
}


//Bubble sort
import java.util.*;
class Sorting{
    public static void printArrays(int array[]){
           for(int i=0;i<array.length;i++){
               System.out.print(array[i] + " ");
           }
        System.out.println("");
    }
    public static void main(String[] args){
        int[] array={7,3,4,2,9,0};
        for(int i=0;i<array.length-1;i++){
            for(int j=0;j<array.length-i-1;j++){
                if(array[j]>array[j+1]){
                    int temp=array[j];
                    array[j]=array[j+1];
                    array[j+1]=temp;
                }
            }
        }
        printArrays(array);
    }
}

//Selection sort
import java.util.*;
class Sorting{
    public static void printArrays(int array[]){
        for(int i=0;i<array.length;i++){
            System.out.print(array[i] + " ");
        }
        System.out.println("");
    }
    public static void main(String[] args){
        int[] array={7,3,4,2,9,0};
        for(int i=0;i<array.length-1;i++){
            int smallest=i;
            for(int j=0;j<array.length-i-1;j++){
                if(array[j]>array[smallest]){
                    smallest=j;

                }
            }
            int temp = array[smallest];
            array[smallest] = array[i];
            array[i] = temp;

        }
        printArrays(array);
    }
}

//Insertion sort
import java.util.*;


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


   public static void main(String args[]) {
       int arr[] = {7, 8, 1, 3, 2};
```

```
        //insertion sort
        for(int i=1; i<arr.length; i++) {
            int current = arr[i];
            int j = i - 1;
                while(j >= 0 && arr[j] > current) {
                    //Keep swapping
                    arr[j+1] = arr[j];
                    j--;
                }
            arr[j+1] = current;
        }
        printArray(arr);
    }
}

//Heap sort
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        int[] arr = {3, 2, 1, 5, 4};
        sort(arr);
        System.out.println(Arrays.toString(arr));
    }
    public static void sort(int[] arr) {
        int n = arr.length;

        for (int i = n / 2 - 1; i >= 0; i--) {
            heapify(arr, n, i);
        }
        for (int i = n - 1; i >= 0; i--) {
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            heapify(arr, i, 0);
        }
    }
    public static void heapify(int[] arr, int n, int i) {
        int largest = i;
        int left = 2 * i + 1;
        int right = 2 * i + 2;

        if (left < n && arr[left] > arr[largest]) {
            largest = left;
        }

        if (right < n && arr[right] > arr[largest]) {
            largest = right;
        }
        if (largest != i) {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;

            heapify(arr, n, largest);
        }
    }
```

```java
}

//Quick sort

import java.util.Arrays;
public class Main {
    public static void main(String[] args) {
        int[] arr = {3, 2, 1, 5, 4};
        sort(arr, 0, arr.length - 1);
        System.out.println(Arrays.toString(arr));
    }
    public static void sort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);

            sort(arr, low, pi - 1);
            sort(arr, pi + 1, high);
        }
    }
    public static int partition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] <= pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        return i + 1;
    }
}


// Implementation of stack

import java.util.ArrayList;

public class Main<T> {
    private ArrayList<T> list = new ArrayList<>();
    public void push(T item) {
        list.add(item);
    }
    public T pop() {
        if (list.isEmpty()) {
            throw new RuntimeException("Stack is empty");
        }
        return list.remove(list.size() - 1);
    }
    public T peek() {
        if (list.isEmpty()) {
            throw new RuntimeException("Stack is empty");
        }
        return list.get(list.size() - 1);
    }
    public int size() {
        return list.size();
    }
    public boolean isEmpty() {
        return list.isEmpty();
    }
}

//Implementation of queue

public class Main {
    static class Queue {
        static int arr[];
        static int size;
        static int rear;


        Queue(int size) {
            this.size = size;
            arr = new int[size];
            rear = -1;
        }


        public static boolean isEmpty() {
            return rear == -1;
        }


        public static boolean isFull() {
            return rear == size-1;
        }


        public static void add(int data) {
            if(isFull()) {
                System.out.println("Overflow");
                return;
            }


            arr[++rear] = data;
        }


        //O(n)
        public static int remove() {
            if(isEmpty()) {
                System.out.println("empty queue");
                return -1;
            }
            int front = arr[0];
            for(int i=0; i<rear; i++) {
                arr[i] = arr[i+1];
            }
            return front;
        }
        public static int peek() {
            if(isEmpty()) {
                System.out.println("empty queue");
                return -1;
            }

            return arr[0];
        }
    }
    public static void main(String args[]) {
        Queue q = new Queue(5);
        q.add(1);
        q.add(2);
        q.add(3);
        System.out.println(q.remove());
        System.out.println(q.peek());
    }
}

//Implementation of single and double linked list
class Node {
    int data;
    Node next;

    public Node(int data) {
        this.data = data;
        next = null;
    }
}

// Singly Linked List class
class SinglyLinkedList {
    Node head;

    public void add(int data) {
        Node newNode = new Node(data);
        newNode.next = head;
        head = newNode;
    }

    public void remove(int data) {
        Node current = head;
        Node prev = null;
        while (current != null) {
            if (current.data == data) {
                if (prev != null) {
                    prev.next = current.next;
                } else {
                    head = current.next;
                }
                break;
            }
            prev = current;
            current = current.next;
        }
    }

    public void printList() {
        Node current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
    }
}

class DNode {
    int data;
    DNode next;
    DNode prev;

    public DNode(int data) {
        this.data = data;
        next = null;
        prev = null;
    }
}

class DoublyLinkedList {
    DNode head;
    DNode tail;

    public void add(int data) {
        DNode newNode = new DNode(data);
        if (head == null) {
            head = newNode;
            tail = newNode;
        } else {
            tail.next = newNode;
            newNode.prev = tail;
            tail = newNode;
        }
    }

    public void remove(int data) {
        DNode current = head;
        while (current != null) {
            if (current.data == data) {
                if (current.prev != null) {
                    current.prev.next = current.next;
                } else {
                    head = current.next;
                }
                if (current.next != null) {
                    current.next.prev = current.prev;
                } else {
                    tail = current.prev;
                }
                break;
            }
            current = current.next;
        }
    }

    public void printList() {
        DNode current = head;
        while (current != null) {
            System.out.print(current.data + " ");
            current = current.next;
        }
    }
}
```