VL01, Lösung 1

b) Der Algorithmus als Pseudocode:

```
// Unterprogramm zum Durchstreichen aller Vielfachen von a
durchstreichenVielfache(a, n)
for b ← 2*a to n step a do
   durchstreichen(b)
end for
```

c) Aufgrund des wahlfreien Zugriffs auf Elemente ist ein Array besonders geeignet.

VL01, Lösung 2

Java-Klasse `SiebDesEratosthenes` (ausführliche Variante):

```
// Ausgabe aller Primzahlen bis zu einer eingegebenen Obergrenze

import java.util.Scanner;

public class PrimzahlTest
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.print("Bitte Zahl eingeben: ");
        int n = sc.nextInt();
        SiebDesEratosthenes sieb = new SiebDesEratosthenes(n);
        int[] primzahlen = sieb.berechnePrimzahlen();

        for (int a = 0; a < primzahlen.length; a++)
            System.out.println(primzahlen[a]);
    }
}

// Eigentliches Verfahren als eigene Klasse: Sieb des Eratosthenes

import java.util.*;

public class SiebDesEratosthenes
{
    // true bedeutet durchgestrichen
    private boolean[] zahlen;

    public SiebDesEratosthenes(int n)
    {
        zahlen = new boolean[n];
    }

    // Interne Methoden
    private void durchstreichen(final int zahl)
    {
        zahlen[zahl-1] = true;
    }

    private boolean isDurchgestrichen(final int zahl)
    {
        return zahlen[zahl-1];
    }
```

```
    private void durchstreichenVielfache(final int zahl)
    {
        for (int a = 2*zahl; a <= zahlen.length; a += zahl)
            durchstreichen(a);
    }

    // Von außen sichtbare Primzahl-Berechnung
    public int[] berechnePrimzahlen()
    {
        durchstreichen(1);

        // Primzahlen durch Durchstreichen der Vielfachen ermitteln
        int a = 2;
        while (a*a <= zahlen.length)
        {
            if (!isDurchgestrichen(a))
                durchstreichenVielfache(a);

            a++;
        }

        // Primzahlen für Rückgabe aufbereiten
        int anzPrimzahlen = 0;

        for (int b = 1; b <= zahlen.length; b++)
            if (!isDurchgestrichen(b))
                anzPrimzahlen++;

        // Rückgabe-Array erzeugen und füllen
        int[] primzahlen = new int[anzPrimzahlen];
        int pos = 0;

        for (int b = 1; b <= zahlen.length; b++)
            if (!isDurchgestrichen(b))
                primzahlen[pos++] = b;

        return primzahlen;
    }
}
```

Java-Klasse `SiebDesEratosthenes` (kompakte und effiziente Variante):

```
// Ausgabe aller Primzahlen bis zu einer eingegebenen Obergrenze
// Verfahren: Sieb des Eratosthenes

import java.util.Scanner;

public class SiebDesErathosthenes
{
    public static void main(String[] args)
    {
        System.out.print("Bitte Zahl eingeben: ");
        final int n = new Scanner(System.in).nextInt();

        // true bedeutet durchgestrichen
        boolean[] zahlen = new boolean[n];

        // 1 durchstreichen
        zahlen[0] = true;

        // Primzahlen berechnen
        int a = 2;
        while (a*a <= n)
        {
            if (!zahlen[a-1])
                // Vielfache markieren
                // Wir können damit bei a*a beginnen:
                // - 2*a wurde bereits als Vielfaches von 2 gestrichen
                // - 3*a wurde bereits als Vielfaches von 3 gestrichen
                // - …
                // - (a-1)*a wurde bereits als Vielfaches von a-1 gestr.
                // - a*a ist das erste zu streichende Vielfache
                for (int b = a*a; b <= n; b += a)
                    zahlen[b-1] = true;

            a++;
        }

        // Primzahlen ausgeben
        for (int b = 1; b <= n; b++)
            if (!zahlen[b-1])
                System.out.println(b);
    }
}
```