Problem 1

Write a static *recursive* function which takes an array of **int**s, starting index and a value; the function returns (but doesn't print!) number of occurrences of this value in the part of the array starting from the given index; for example, the following program

download *Recurs1.java*

```
public class Recurs1 {
   static int count(int[] arr, int from, int what) {
       // ...
   }

   public static void main(String[] args) {
       int[] a = {2,3,2,4,3,1,6,3,2,3};
       System.out.println("2 -> " + count(a,0,2));
       System.out.println("3 -> " + count(a,0,3));
   }
}
```

should print

```
2 -> 3
3 -> 4
```

Do not use static fields, loops or additional arrays/collections/strings!

Problem 2

Write a *recursive* function **binSearchRec**

```
public static int binSearchRec(int[] a, int what,
                               int from, int to);
```

which takes

- *Sorted* array of **int**s (a);
- an integer what;
- range of indices: from from to to.

The function returns the index of the element of the array a which is equal to what, taking into account only elements from the one with index from *inclusive* to the one with index to *exclusive*. If there is no element of this value in the array, the function returns -1.
Use the binary search algorithm (which has complexity $\log n$, not linear).

The following program

download *RecBinSearch.java*

```
public class RecBinSearch {
    public static int binSearchRec(int[] a, int what,
                                   int from, int to) {
        // ...
    }
    public static void main (String[] args) {
        int[] a = {1, 4, 5, 7, 9, 10};
        for (int i = a[0]; i <= a[a.length-1]; ++i)
            System.out.printf("what=%2d ind=%2d%n",
                    i, binSearchRec(a, i, 0, a.length));
        System.out.println("**************");
        int[] b = {-1,1,3,4,6};
        for (int i = b[0]; i <= b[b.length-1]; ++i)
            System.out.printf("what=%2d ind=%2d%n",
                    i, binSearchRec(b, i, 0, b.length));
    }
}
```

should print

```
what= 1 ind= 0
what= 2 ind=-1
what= 3 ind=-1
what= 4 ind= 1
what= 5 ind= 2
what= 6 ind=-1
what= 7 ind= 3
what= 8 ind=-1
what= 9 ind= 4
what=10 ind= 5
**************
what=-1 ind= 0
what= 0 ind=-1
what= 1 ind= 1
what= 2 ind=-1
what= 3 ind= 2
what= 4 ind= 3
what= 5 ind=-1
what= 6 ind= 4
```

Problem 3

Create a class **FuncStat** containing only public static functions:

- `public static int fiboR(int n)` calculating the n-th Fibonacci number

$$F_n = \begin{cases} n & \text{for } 0 \leqslant n < 2, \\ F_{n-1} + F_{n-2} & \text{for } n \geqslant 2 \end{cases}$$

using this recursive formula, which is rather unwise but enlightening — therefore, the function should be *recursive* (in particular, no loops are allowed in its implementation!);

- `public static int fiboI(int n)` calculating the n-th Fibonacci number without recursion (i.e., iteratively — using a loop);
- `public static int factR(int n)` calculating $n!$ recursively;
- `public static int factI(int n)` calculating $n!$ iteratively;
- `public static int gcdR(int a, int b)` calculating the GCD, i.e., the greatest common divisor, of a and b recursively;
- `public static int gcdI(int a, int b)` calculating the GCD of a and b iteratively;
- `public static int maxElem(int[] arr, int from)` returning the largest element in the part of the array arr starting from element with index from. It must be a **recursive** function. Function **max** from class **Math** may be useful but is not necessary;
- `public static int numEven(int[] arr, int from)` returning the number of even elements in the part of the array arr starting from element with index from. It must be a **recursive** function.
- `public static void reverse(int[] arr, int from)` reversing the order of elements in the part of the array arr from the element with index from to the element with index arr.length-1-from. It must be a **recursive** function. Do *not* create any auxiliary arrays!.
- `public static boolean isPalindrom(String s)` returning **true** if, and only if, string `s` is a palindrom, i.e., a word which reads the same forward and backward, as, e.g., words *radar* or *madam*. Methods **charAt** and **substring** from class **String** may be useful. It must be a **recursive** function.

Then, in the **main** function *of a separate class* **Main** test all these functions.

Remark: according to Euclid (*Elements*, Book VII), the greatest common divisor of two positive integers, a and b, can be calculated as follows:

1. if $a = b$, then the result is a (or b, as they are equal);
2. from the larger of these two numbers subtract the smaller and go to 1.

Do not use any classes from packages other than basic ***java.lang*** (in particular, no collections are allowed).

You can assume that functions will be invoked with legal arguments (e.g., no negative argument of the factorial function).

For example, the following **main** function

download *FuncStat.java*

```
import java.util.Arrays; // for printing
    // ...
    public static void main (String[] args) {
        System.out.println("Wait...");
        System.out.println(FuncStat.fiboR(45));
        System.out.println(FuncStat.fiboI(45));
```

```
        System.out.println(FuncStat.factR(12));
        System.out.println(FuncStat.factI(12));
        System.out.println(FuncStat.gcdR(12125,40643));
        System.out.println(FuncStat.gcdI(12125,40643));
        int[] a = {3,8,2,9,7,4};
        System.out.println("Max     : " + FuncStat.maxElem(a,0));
        System.out.println("Num even: " + FuncStat.numEven(a,0));
        System.out.println("Before:   " + Arrays.toString(a));
        FuncStat.reverse(a,0);
        System.out.println("After :   " + Arrays.toString(a));
        System.out.println("Is 'radar' a palindrom? " +
                           FuncStat.isPalindrom("radar"));
        System.out.println("Is 'abba' a palindrom?  " +
                           FuncStat.isPalindrom("abba"));
        System.out.println("Is 'rover' a palindrom? " +
                           FuncStat.isPalindrom("rover"));
    }
```

should print (note that calculating Fibonacci number F_{45} recursively takes a while...)

```
Wait...
1134903170
1134903170
479001600
479001600
97
97
Max     : 9
Num even: 3
Before:   [3, 8, 2, 9, 7, 4]
After :   [4, 7, 9, 2, 8, 3]
Is 'radar' a palindrom? true
Is 'abba' a palindrom?  true
Is 'rover' a palindrom? false
```

Problem 4

We are responsible for preparing a survey. Each questionnaire contains the following pieces of information, which have to be encoded into *one* variable of type **short** (it is 2 bytes, i.e., 16 bits, long):

1. sex — 1 bit, as 2 possibilities: woman, man, coded as 0 or 1;
2. marital status — 2 bits, as 4 possibilities: maid/bachelor, married, divorced, widow, coded as 0, 1, 2 or 3
3. age group — 2 bits, as 4 possibilities, e.g., 18-30, 31-45, 46-60, 60+, coded as 0, 1, 2 or 3;
4. education — 2 bits, as 4 possibilities, e.g., elementary, high school, college, university, coded as 0, 1, 2 or 3;

5. place of living — 2 bits, as 4 possibilities, e.g., country, town to 50 000, city to 400 000, city over 400 000, coded as 0, 1, 2 or 3;
6. region of the country — 4 bits, as, say, there are 16 regions coded as 0-15;
7. answer to the question which is the subject of the suvey — 3 bits, as there are 8 possible answers numbered 0-7.

Write a function

```
static short encode(int sex,   int marit,
                    int age,   int edu,
                    int place, int region,
                    int answer)
```

which takes 7 numbers (as above) and encodes them all into *one* variable of type **short** which is returned by the function.
Also, write a function

```
static void info(short code)
```

which takes one **short** containing information about one questionnaire and writes this information, e.g., in the form:

```
sex:             0
marital status:  3
age group:       2
education:       3
living place:    0
region:         12
answer:          6
```

Do *not* create any arrays or **String**s. Use only bit operations (ANDing, ORing, shifting...).

Problem 5

Write a **recursive** function

```
static void egyptian(int n, int d)
```

which prints the fraction $\frac{n}{d}$ in the "egyptian" form, i.e., as the sum of a whole number and several fractions with numerators equal to 1.

Note: even for "innocently" looking fractions, the denominators can be so large that they don't fit in an integer! You can ignore such cases.

For example, the following program

download *Egyptian.java*

```
void main() {
    printEgyptian(  4,   8);
    printEgyptian(  8,   4);
    printEgyptian(  3,   4);
```

```
        printEgyptian(  7,   9);
        printEgyptian(  0,   8);
        printEgyptian(  7,   0);
        printEgyptian( 21,  31);
        printEgyptian(123,  43);
    }

    static void printEgyptian(int n, int d) {
        IO.print(n + "/" + d + " -> ");
        egyptian(n, d);
        IO.println();
    }

    static void egyptian(int n, int d) {
        // ...
    }
```

should print

```
4/8 -> 1/2
8/4 -> 2
3/4 -> 1/2 + 1/4
7/9 -> 1/2 + 1/4 + 1/36
0/8 -> 0
7/0 -> Denominator is 0!!!
21/31 -> 1/2 + 1/6 + 1/93
123/43 -> 2 + 1/2 + 1/3 + 1/37 + 1/9546
```
