Problemas de Recursividad

Problema 1.

El factorial de un número entero $n \geq 0$, denotado como $n!$, se define como $\prod_{i=1}^{n} i = 1 * 2 * \ldots * n$ cuando $n > 0$, y $0! = 1$.
Por ejemplo $6! = 1 * 2 * 3 * 4 * 5 * 6 = 720$
Diseñad una método recursiva que lo calcule e implementadlo en Java (junto con un programa que lo utilice)

Problema 2. .

Para calcular el máximo común divisor de dos números enteros puedo aplicar el **algoritmo de Euclides**, que consiste en ir restando el más pequeño del más grande hasta que queden dos números iguales, que serán el máximo común divisor de los dos números.
Por ejemplo, si comenzamos con el par de números 412 y 184, tendríamos:

412	228	44	44	44	44	44	36	28	20	12	8	**4**
184	184	184	140	96	52	8	8	8	8	8	4	**4**

Es decir, m.c.d.(412, 184) = 4

Problema 3.

Diseñar un método recursivo tal que dado un vector de números enteros retorne la suma de sus elementos.
Para poder hacer recursividad, usaremos un índice que indique el trozo de vector a sumar en cada llamada.
Es decir, el método a diseñar tendrá la forma:

```
1 public int sum(int[] elems, int pos) {
2   ¿?
3 }
```

Diseñad este método así como el que lo utiliza para calcular la suma de todo el vector. Tened en cuenta cómo hacemos para referirnos a un intervalo dentro de un vector. ¿Qué pasa si el vector está vacío (es decir, cuando elems.length vale cero)?
Usando el método recursivo, implementad el método que lo usa para calcular la suma de todo el vector, es decir:

```
4 public int sum(int[] elems) {
5   return sum(elems, ¿?);
6 }
```

Nota: Podéis considerar dos descomposiciones del vector, una en la que la zona que vais sumando crece de derecha a izquierda y otra en la que lo hace en sentido contrario.

Problema 4.

Diseñad un método recursivo que escriba al revés la cadena que se le pasa como parámetro, más un índice que se usará para recorrer la cadena.
Dicho método será de la forma:

```
1 public void printReversed(String text, int index) {
2   ¿?
3 }
```

Haced dos versiones del mismo, una en la que el índice vaya incrementándose a cada llamada y otra en la éste que vaya decrementándose. En los dos casos implementad la función que llama a la función recursiva diseñada, es decir:

```
4 public void printReversed(String text) {
5   printReversed(text, ¿?);
6 }
```

Nota: No vale invertir la cadena y luego escribirla.

Problema 5.

El ejemplo de la exponenciación mostrado en los apuntes, permite la siguiente descomposición:

- Si b es par, $a^b = a^{2*(b \text{ div } 2)} = \left(a^{b \text{ div } 2}\right)^2$
- Si b es impar, $a^b = a^{2*(b \text{ div } 2)+1} = a * \left(a^{b \text{ div } 2}\right)^2$

Acabad de diseñar la solución recursiva que la emplea, implementar la solución en Java y hacer el mismo diagrama de llamadas para el caso de 7^{13}.
Nota: Es muy interesante que intentéis resolver un mismo problema de varias maneras y comparéis entre sí las diferentes soluciones.

Problema 6.

Ya que estamos, diseñad un método tal que dada una cadena, retorne la cadena invertida (es decir, el primer carácter del resultado será el último de la cadena dada, etc.). Dicho método tendrá la forma:

```
1 public String invert(String text) {
2   ¿?
3 }
```

Para hacerlo, debéis diseñar otro tal que dado un vector de caracteres, lo invierta. Como los parámetros que son vectores se pasan por referencia, el método invert sobre vectores puede ser de la forma:

```
1 public void invert(char[] textChars) {
2   ¿?
3 }
```

Para encontrar recursividad deberéis hacer otro método que, además del char[], use uno o más índices sobre el vector.

Problema 7.

Diseñad un método tal que, dados dos vectores de enteros, retorne un booleano indicando si son iguales, es decir, si tienen los mismos valores en las mismas posiciones.
Para poder hacerlo recursivamente deberéis, como ya es habitual, hacer otro método que incluya índices para indicar los trozos de subvectores sobre los que se trabaja. Indicad qué llamada se hace al método recursivo para resolver el problema inicial.

Problema 8.

Diseñad un método tal que calcule el máximo de un vector **no vacío** de números enteros. De forma similar al problema 4, implementad el método que llama al que habéis definido recursivamente para que se calcule el máximo de todo el vector.

Problema 9.

El algoritmo chino de multiplicación. Diseñar un método que multiplique dos números enteros usando las siguientes equivalencias:

$$x * y = (2 * x) * \left(\frac{y}{2}\right) = \begin{cases} (2 * x) * (y \; div \; 2), & si \; y \; es \; par \\ (2 * x) * (y \; div \; 2) + x, & si \; y \; es \; impar \end{cases}$$

Problema 10.

Dado un vector de números enteros ordenado decrecientemente, diseñad un método tal que compruebe si el valor de alguno de los elementos del vector coincide con su índice.
Podéis hacer dos versiones:

- una que vaya comprobando elemento a elemento si dicha propiedad se cumple (para esta versión, el método recursivo usará, además del vector, un índice).
- otra que, usando dos índices, sea capaz de descartar a cada llamada la mitad del vector.

En ambos casos implementad los métodos que hacen la llamada inicial al que habéis diseñado recursivamente dando valores iniciales a los índices.
Pista: podéis pensar qué relación tiene este problema con la búsqueda dicotómica y, si la encontráis, obtendréis la solución.

Problema 11.

Un problema parecido al anterior se puede plantear cuando el vector de enteros está ordenado crecientemente y no contiene valores repetidos.
El razonamiento en este caso es más complicado que en el caso anterior (obviamente cuando se intenta hacer la versión que, a cada paso divide la longitud del intervalo donde buscar por la mitad).
Pista: la idea de la solución consiste en darse cuenta de que los valores crecen como mínimo tanto como los índices. Esto es cierto porque el vector no contiene elementos repetidos.

Problema 12.

La sucesión de Fibonacci viene definida por la siguiente recurrencia:

$$f_{n+2} = f_n + f_{n+1}$$

con valores iniciales $f_0 = 0$ y $f_1 = 1$.
Diseñad e implementad un método recursivo para calcular el enésimo término de la sucesión y mostrad el árbol de llamadas que se produce al calcular f_4 con vuestra solución.