# Primera práctica de Programación 2

La primera práctica consistirá en el diseño y posterior implementación como proyecto de Netbeans de tres problemas diferentes.

- El primero de ellos trata sobre la manipulación de vectores en Java usando recorridos como los vistos en la asignatura de Programación 1.
- El segundo es un problema de recursividad, que además sirve como *preparación* para el tercer problema
- El tercero también va sobre recursividad y es una versión más compleja del segundo problema

## Problema 1. (4 puntos)

Se quiere diseñar una función que permita sumar números enteros positivos representados como un vector de sus dígitos.
Por ejemplo, el vector {1, 2, 3} representará el número 321, es decir, la posición 0 del vector serán las unidades, la posición 1 las decenas, etc.
La longitud del vector es el número de cifras significativas del número.
Por tanto el número 321 será el vector {1,2,3} y no {1,2,3,0}, {1,2,3,0,0} etc.
De cara a ver los resultados, es conveniente que implementemos una función que nos permita escribir un vector como si fuera un número (es decir, con los dígitos en el orden normal). Para ello añadiremos a la clase el método `printAsNumber`.
Un esquema de la clase con un pequeño método principal de prueba, es:

```
1 public class VectorArithmetic extends ConsoleProgram {
2
3   public int[] sum(int[] a, int[] b) {
4     ¿?
5   }
6
7   public void printAsNumber(int[] a) {
8
9     // Prints vector a as a number considering that the
10    // units at at position 0, tens at position 1, etc.
11    // Prints an space both before and after the number.
12    // a only contains significant digits.
13
14    print(" ");
15    for (int i=a.length-1; i>=0; i--) {
```

```
16        print(a[i]);
17      }
18      print(" ");
19    }
20
21    public void run() {
22      int[] number1 = {1,5,4,9};
23      int[] number2 = {1,5,5};
24
25      printAsNumber(number1);
26      print("+");
27      printAsNumber(number2);
28      print("=");
29      printAsNumber(sum(number1,number2));
30
31      println("");
32
33      printAsNumber(number2);
34      print("+");
35      printAsNumber(number1);
36      print("=");
37      printAsNumber(sum(number1,number2));
38    }
39 }
```

Si todo va bien, el programa escribe en la pantalla:

```
9451 + 551 = 10002
551 + 9451 = 10002
```

## Una pista

Cuando analicéis el problema veréis que se debe considerar cual de los dos vectores que nos pasan como parámetro tiene más dígitos. Para no tener que estar todo el rato añadiendo condicionales que distinga cuando a tiene más dígitos que b o viceversa, podéis hacer lo siguiente al empezar el método sum:

```
1 public int[] sum(int[] a, int[] b) {
2   int[] longer;
3   int[] shorter;
4   if ( a.length >= b.length ) {
5     longer = a;
6     shorter = b;
7   } else {
8     longer = b;
9     shorter = a;
```

```
10   }
11  // Aquí os podéis olvidar de a y b.
12  // Programad el método usando longer
13  // y shorter sabiendo que longer tiene
14  // más dígitos que shorter.
15
16   ¿?
17 }
```

## Un consejo

No intentéis encontrar a la primera la mejor solución (p.e. la más elegante, la que usa menos bucles, menos condicionales, etc.). Encontrad una solución sencilla (casi inmediata) que resuelva el problema, probadla exhaustivamente, y mejoradla poco a poco.

# Problema 2. (2 puntos)

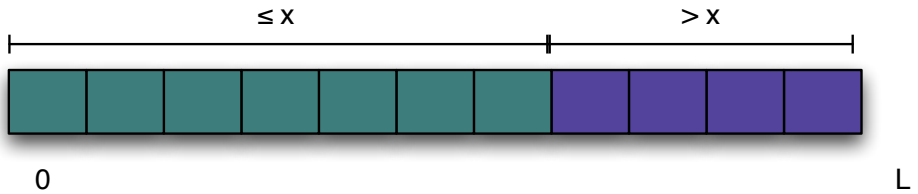
Se quiere implementar un método que reciba dos parámetros:

- un vector de enteros v
- un número entero x (que puede o no estar en v)

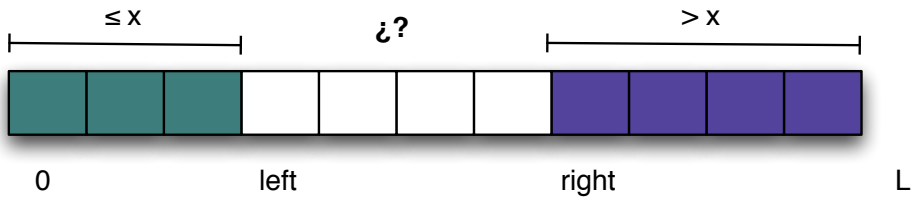
Lo que querremos es que el método reordene el vector de manera que queden a la izquierda todos los elementos que son menores o iguales que x, y a la izquierda los que son mayores que x. Obviamente para conseguirlo, deberemos modificar el vector.

Por ejemplo, si v = { 1, 10, 23, 15, 3, 8} y x = 8, soluciones posibles son dejar el vector como {1, 8, 3, 23, 10, 15} o {8,1,3,23, 15, 10} entre otras.

Fijaos que el problema se parece al de la búsqueda binaria, pero ahora lo que hemos de hacer es modificar el vector para que al final se cumpla que (fijaos que cualquiera de esas zonas puede estar vacía):




Como es un problema de recursividad sobre vectores tendremos que añadir parámetros que serán los índices del vector, y que separarán el vector en varias zonas. Es decir:




Para modificar el vector podéis usar solamente el método swap.

Por tanto, la clase que tenéis que implementar como solución tendrá la siguiente estructura:

```
18 public class ReorganizeVector extends ConsoleProgram {
19
20   public void swap(int[] v, int i, int j) {
21     // Both i and j valid indexes on array v
22     int temp = v[i];
23     v[i] = v[j];
24     v[j] = temp;
25   }
26
27   public void reorganize2(int[] v, int x) {
28     // Llamada inicial al método con índices
29     reorganize2(v, x, ¿?, ¿?);
30   }
31
32   public void reorganize2(int[] v,
33                           int x,
34                           int left,
35                           int right) {
36      ¿?
37   }
38
39   public void println(int[] v) {
40     // Prints all elements of an array
41     print("{");
42     for (int i = 0; i<v.length; ++i) {
43       print(v[i]);
44       if ( i != v.length-1) { print(", ");}
45     }
46     println("}");
47   }
48
49   public void run() {
50     int[] v = {1, 10, 23, 15, 3, 8};
51     println(v);
52     reorganize2(v, 8);
53     println(v);
54   }
55 }
```

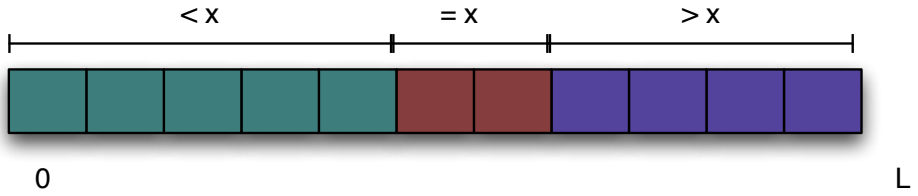
Fijaos que no importa en absoluto en qué lugar empiezan o acaban las zonas sino que lo único que quiero es reorganizar el vector. Por ello el método recursivo devuelve `void` y su efecto se nota por las

modificaciones que hace sobre el vector (que se pasa, como todos los vectores, **por referencia**).

## Problema 3. (4 puntos)

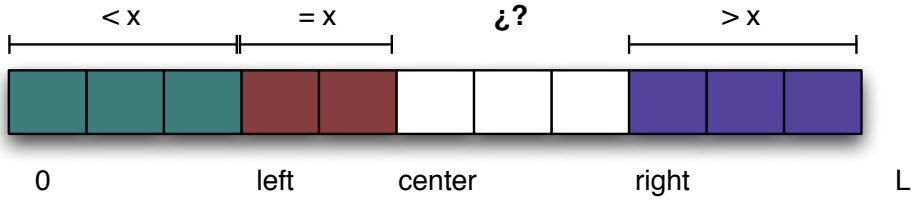
Vamos a complicar un poco el problema 2 y ahora vamos a demandar lo siguiente: queremos que al final haya tres zonas en el vector, la de los elementos más pequeños que x, la de los iguales a x y la de los que son mayores que x.

Es decir, al final se desea que el vector quede de la siguiente manera:




De forma similar a cuando teníamos que separar el vector en dos zonas, añadiremos una zona más que será la que indique la zona del vector que es desconocida (y que deberemos hacer más pequeña a cada paso).

Por ello consideraremos los siguientes índices y zonas:




Para no tener que copiar los métodos `swap` y `println` del problema anterior, podemos implementar la solución de este problema en la misma clase que la solución del problema 2. Por tanto añadiremos los métodos:

```
1 public void reorganize3(int[] v, int x) {
2   reorganize3(v, x, ¿?, ¿?, ¿?);
3 }
4
5 public void reorganize3(int[] v,
6                          int x,
7                          int left,
8                          int center,
9                          int right) {
10  ¿?
11 }
```

## Formato de la entregas:

La entrega se hará vía **sakai** y consistirá en un archivo (comprimido con **ZIP**) que contenga:

- los **proyectos de netbeans** que solucionen los problemas
- un **pequeño informe** (máximo 3 páginas) en **PDF** con la explicación de cómo habéis llegado a la solución y justificando su funcionamiento (podéis usar diagramas similares a los que hay en los apuntes) y todo aquello que consideréis relevante hacer constar.
- si queréis hacer los diagramas a mano y luego escanearlos, por nosotros no hay problema alguno (si se entienden, claro).
- la práctica se realiza **individualmente**.
- la práctica es un **20%** de la nota de la asignatura.
- la fecha límite es el **lunes 4 de abril a las 8 de la mañana**.