Ejercicio 1

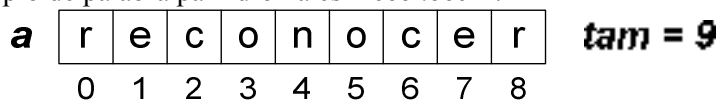
Dado el siguiente fragmento de código, se pide calcular su orden exacto.

```
i = n;
while (i >= 1) {
    j = 1;
    while (j <= i ) {
      s = s + j;
      j = j + 2;
    }
    i = i / 3;
}
```

Tiempo estimado: 30 minutos **3 puntos**

Ejercicio 2

Una cadena de caracteres se denomina **palíndroma** cuando se puede leer igual en ambos sentidos. Un ejemplo de palabra palíndroma es "*reconocer*".



Se desea implementar un **algoritmo recursivo** que determine si una cadena presenta o no dicha propiedad. Para dicha implementación, se declaran y utilizan las siguientes variables:

- **a**: `char *`, (equivalente a `char []`) variable global que contiene la palabra a analizar. La forma de acceder a los elementos de `a` es la habitual en los arrays: mediante `a[i]` se accede a la posición `i`-ésima del array, comenzando en la posición 0 y terminando en la posición tamaño(a)-1.
- **tam**: `int`, variable global que almacena el tamaño de la cadena `a`.

De forma recursiva, se puede definir la función `palind(int i)` como se muestra a continuación:

$$palind(i)\begin{cases} true, & i >= tam/2 \\ false, & a[i]\ != a[tam-i-1] \\ palind(i+1), & en\ otro\ caso \end{cases}$$

Acorde con esta función, para determinar si una cadena **a** es palíndroma (`palindroma(a)`) se realiza la llamada inicial `palind(0)` a la función recursiva (`palindroma(a) = palind(0)`).

Teniendo en cuenta la definición anterior, **SE PIDE** realizar la implementación de los siguientes métodos que forman parte del archivo Palindroma.c (*reverso del folio*):

a) `logico` **`esCasoBase`**`(int i)`
b) `logico` **`solucionBase`**`(int i)`
c) `int` **`siguiente`**`(int i)`
d) `logico` **`combina`**`(int i, logico rSig)`

Palindroma.c

```
#include "palindroma.h"

char* a; // Array de caracteres, a[i] accede a la posición i-ésima
int tam; // Tamaño de a

logico palindroma(char* cadena) {
      tam = strlen(cadena); // Almacena en tam el tamaño de a
      a = cadena; // Almacena en el array a la cadena a analizar
      return palind(0); // Realiza la llamada inicial a palind
}
logico palind(int i) { // función recursiva (f en los apuntes de
clase)
    int iSig;
    logico r,rSig;
    if (esCasoBase(i)) {
        r = solucionBase(i);
    }
    else {
        iSig = siguiente(i);
        rSig = palind(iSig);
        r = combina(i, rSig);
    }
    return r;
}
logico esCasoBase(int i){
      // TO DO
}
logico solucionBase(int i){
      // TO DO
}

int siguiente(int i){
      // TO DO
}

logico combina(int i, logico rSig){
      // TO DO
}
```

Tiempo: 30 minutos **3 puntos**

Ejercicio 3

Dada una lista ordenada con respecto a un orden, encontrar la posición de un elemento dado si existe, o -1 si no lo encuentra. El problema se generaliza añadiendo dos propiedades: `I`, entero en `[0,Datos.size()]` *y* `J`, entero en `[I,Datos.size()]`. El problema generalizado busca el elemento en el trozo de lista definido por el intervalo **`[I,J)`.**

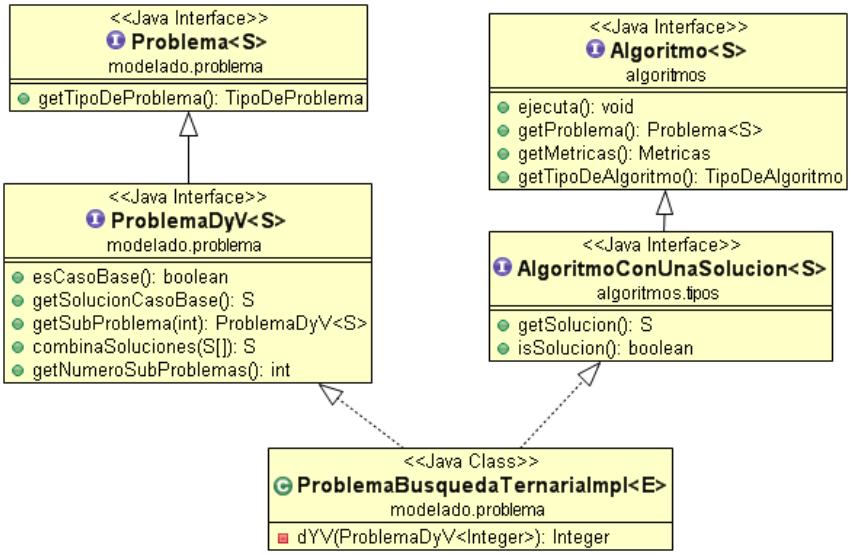
El conjunto de problemas tiene tres propiedades comunes: `Datos` (de tipo `List<E>`) que contiene la lista ordenada de elementos, `Orden` (de tipo `Comparator<E>`) que contiene un orden con respeto al cual está ordenada la lista y `Elemento` (de tipo `E`) el elemento a buscar. Además cada problema tiene las dos propiedades individuales `I`, entero en

`[0,Datos.size()]` *y* J, entero en `[I,Datos.size()]`. En este conjunto de problemas dos problemas de este tipo son iguales si tienen iguales la propiedad `I` y `J`.
Se trata de encontrar la posición del elemento dentro de la lista, si está. La solución al problema es un entero en `[-1,Datos.size())`. Si la solución es `-1`, el `Elemento` no está en `Datos`.
Para resolver el problema mediante **`Divide y Vencerás`**, consiste en dividir la estructura en tres partes (llamémosle `tercio`), comprobar si el elemento que está en la posición `tercio` ó en la posición `doble de tercio` es el que buscamos y, en caso negativo, comparar el elemento buscado con el que está en la posición `tercio`. Si es anterior, repetir con el primer `tercio` de la estructura. Si está después que el elemento la posición `tercio`, pero anterior al elemento de la posición `doble de tercio`, repetir buscando en el segundo `tercio` de la estructura. Y si es posterior, repetir buscando en el tercer (último) `tercio` de la estructura.

Se pide:

1. Rellene la Ficha adjunta siguiendo la información del problema `Búsqueda Ternaria` detallado anteriormente.
2. Implemente los métodos pertenecientes a la clase **`ProblemaBusquedaTernariaImpl:`**
   a. **`private Integer`** `dYV(ProblemaDyV<Integer> p)`. Resuelve la técnica Divide y Vencerás sin memoria.
   b. **`public boolean`** `esCasoBase()`. Indica si el problema que se está resolviendo es tan simple que no necesita recursión para su resolución.
   c. **`public Integer`** `getSolucionCasoBase()`. Devuelve la solución al problema, resuelto de forma directa (sin llamada recursiva).
   d. **`public int`** `getNumeroSubProblemas()`. Indica el número de subproblemas.
   e. **`public ProblemaDyV<Integer>`** `getSubproblema(int i)`. Devuelve el subproblema iésimo en el que se descompone el problema que se está resolviendo (que no es caso base).
   f. **`public Integer`** `combinaSoluciones (Integer... soluciones)`. Combina cada solución de la lista de soluciones, (cada una de ellas, es solución de un subproblema) de manera que juntas resuelvan el problema completo.

.



Tiempo estimado: 60 minutos **4 puntos**

Solución

Ejercicio 1

$$T(n) \approx \sum_{i\in I}\sum_{j\in J} K = K\sum_{i\in I}\frac{(i-1)}{2} \approx \frac{K}{2}\sum_{i\in I} i = \frac{K}{2}\left(n + \frac{n}{3} + \frac{n}{9} + \cdots 1\right)$$

$$\approx K'(1 + 3 + 3^2 + \cdots + n) = K'\sum_{x=0}^{log_3 n} 3^x$$

$$= K'(3^0 + \sum_{x=1}^{log_3 n} 3^x) = K'\left(\frac{3n-1}{3-1}\right) = K'\left(\frac{3n-1}{2}\right)$$

$$\in \Theta(n)$$

Donde $I = \left\{n, \frac{n}{3}, \frac{n}{3^2}, \ldots, 1\right\} \approx \{1, 3, 3^2, \ldots, n\}$ y $J = \{1, 3, 5, \ldots, i\}$

En el caso en el que n<1, el coste será constante.

Ejercicio 2

a) logico **esCasoBase**(int i) **1 punto**

```
logico esCasoBase(int i){
    return (i >= tam/2 || a[i] != a[tam-i-1]);
}
```

b) logico **solucionBase**(int i) **1 punto**

```
logico solucionBase(int i){
      logico res;
      if(i >= tam/2) {
            res = true;
      }
      else { // a[i] != a[tam-i-1]
            res = false;
      }
      return res;
}
```

c) int **siguiente**(int i) **0.5 puntos**

```
int siguiente(int i){
      return i + 1;
}
```

d) `logico` **`combina`**`(int i, logico rSig)` **`0.5 puntos`**

```
logico combina(int i, logico rSig){
      return rSig; // Recursividad final
}
```

Ejercicio 3

1. Rellene la Ficha adjunta siguiendo la información del problema Búsqueda Ternaria detallado anteriormente. **1.5 puntos**

Ficha	
Búsqueda Ternaria	
Técnica: Divide y Vencerás sin Memoria	
Representación: Cada problema es un objeto	
Tamaño: N =J-I	
Propiedades Compartidas	*Datos, List<E>* *Orden, Comparator<E>* *Elemento, E*
Propiedades Individuales	*I, entero en [0,Datos.size()]* *J, entero en [I,Datos.size()]*
Solución: Entero en [-1,Datos.size())	
$bt(d,o,e) = b(0,Datos.Size)$ $b(i,j) = \begin{cases} -1, & i = j \\ s+i, & i < j, e = d(s+i) \\ 2s+i, & i < j, e = d(2s+i) \\ b(i, s+i), & i < j, e < d(s+i) \\ b(s+i+1, 2s+i), & i < j, d(s+i) < e < d(2s+i) \\ b(2s+i+1, j), & i < j, e > d(2s+i) \end{cases}$ $s = \frac{j-i}{3}$	
Recurrencia	$T(n) = T\left(\frac{n}{3}\right) + c$
Complejidad	$\Theta(\log n)$

2. Implemente los métodos pertenecientes a la clase **ProblemaBusquedaTernariaImpl: 2.5 puntos**

a) **private Integer** dYV(ProblemaDyV<Integer> p).

```
private Integer dYV(ProblemaDyV<Integer> p) {
    Integer s;

    if (p.esCasoBase()) {

          s = p.getSolucionCasoBase();

    } else {

          int numeroDeSubProblemas = p.getNumeroSubProblemas();

          Integer[] soluciones = Utiles.creaArray(tipoSolucion,

                      numeroDeSubProblemas);

          for (int i = 0; i < numeroDeSubProblemas; i++) {

                s = dYV(p.getSubProblema(i));

                soluciones[i] = s;

          }

          s = p.combinaSoluciones(soluciones);

    }

    return s;

}
```

b) **public boolean** esCasoBase().

```
public boolean esCasoBase() {

      boolean esCB = false;

      if (I.compareTo(J) == 0) {

            esCB = true;

      } else {

            if (orden.compare(datos.get(getS() + I), elemento) == 0) {

                  esCB = true;

            } else {

                  if (orden.compare(datos.get(2*getS()+I), elemento)==0) {

                        esCB = true;

                  }

            }

      }

      return esCB;

}
```

c) **public Integer** getSolucionCasoBase().

```
public Integer getSolucionCasoBase() {
    Integer resultado;
    if (I.compareTo(J) == 0) {
          resultado = -1;
    } else {
          if (orden.compare(datos.get(getS() + I), elemento) == 0) {
                resultado = getS() + I;
          } else {
                resultado = 2 * getS() + I;
          }
    }
    return resultado;
}
```

d) public int getNumeroSubProblemas(). Indica el número de subproblemas.

```
public int getNumeroSubProblemas() {
      return 1;
}
```

e) public ProblemaDyV<Integer> getSubproblema(int i). Devuelve el subproblema iésimo en el que se descompone el problema que se está resolviendo (que no es caso base).

```
public ProblemaDyV<Integer> getSubProblema(int i) {
ProblemaDyV<Integer> subp;
switch (i) {
case 0:
if (orden.compare(elemento, datos.get(getS() + I)) < 0) {
      subp = new ProblemaBusquedaTernariaImpl2<E>(I,getS()+I, elemento);
} else {
      if ((orden.compare(elemento, datos.get(I + getS())) > 0)
                  && (orden.compare(elemento,datos.get(I+2 *getS()))<0)) {
            subp = new ProblemaBusquedaTernariaImpl2<E>(I + getS() + 1,
                                          I + 2 * getS(), elemento);
      } else {
```

```
                subp = new ProblemaBusquedaTernariaImpl2<E>(I + 2 * getS()
                                                    + 1, J, elemento);
        }
}
break;
default:
        throw new SituacionIlegal("Sólo hay 1 subproblema");
return subp;
}
```

f) **public Integer** combinaSoluciones(soluciones).

```
public Integer combinaSoluciones(Integer... soluciones) {
        Integer sol = soluciones[0];
        return sol;
}
```

NOTA: Para la implementación de los métodos anteriores, es necesario el cálculo del tercio:

```
public int getS() {
        return (J - I) / 3;
}
```