Ejercicio 1. Recursividad

Una progresión geométrica de números enteros se define como una secuencia en la que cada elemento se obtiene multiplicando el anterior por una constante denominada "razón" o factor de la progresión.

Se pide la implementación de un método que decida si los elementos de una lista de enteros forman una progresión geométrica.

Una definición recursiva viene dada por:

$$esProgGeom(l) = \begin{cases} false, & l.size() < 2 \\ pg(l, l[1]/l[0]), & l.size() \geq 2 \end{cases}$$

$$pg(l,r) = pgi(l,r,l.size() - 1)$$

$$pgi(l,r,i) = \begin{cases} true, & i = 1 \\ pgi(l,r,i-1) \wedge (l[i] = l[i-1] * r), & i > 1 \end{cases}$$

a) Implemente el algoritmo recursivo no final en base a la definición anterior.
b) Transforme el algoritmo a recursivo final, dando la definición y la implementación.
c) Implemente la versión iterativa del algoritmo recursivo final.

NOTA: En todos los casos se deben incluir todos los métodos implicados.

Soluciones:

a) Algoritmo Recursivo No Final

```
// Recursivo No Final

public static Boolean esProgGeomRNF(List<Integer> ls) {
    Boolean res;
    if (ls.size() < 2)
        res = false;
    else
        res = pg(ls, ls.get(1)/ls.get(0));
    return res;
}

private static Boolean pg(List<Integer> ls, Integer r) {
      return pgi(ls, r, ls.size()-1);
}

private static Boolean pgi(List<Integer> ls, Integer r, Integer i) {
      Boolean res;
      if (i==1) {
            res = true;
      } else {
            res = pgi(ls, r, i-1) && (ls.get(i) == ls.get(i-1)*r);
      }
      return res;
}
```

b) Algoritmo Recursivo Final:

a. Definición

$$esProgGeom(l) = \begin{cases} false, & l.size() < 2 \\ pg(l, l[1]/l[0]), & l.size() \geq 2 \end{cases}$$

$$pg(l,r) = pgif(true, l, r, l.size() - 1)$$

$$pgif(ac, l, r, i) = \begin{cases} ac\,, & i = 1 \\ pgif(ac \wedge (l[i] = l[i-1] * r), l, r, i-1), & i > 1 \end{cases}$$

b. Implementación

```
public static Boolean esProgGeomRF(List<Integer> ls) {
      Boolean res;
      if (ls.size() < 2)
            res = false;
      else
            res = pg(ls, ls.get(1)/ls.get(0));
      return res;
}

private static Boolean pg(List<Integer> ls, Integer r) {
```

```
            return pgif(true, ls, r, ls.size()-1);
        }

        // pgif puede tener dos versiones, pgif1 y pgif2

private static Boolean pgif1(Boolean ac, List<Integer> ls, Integer r, Integer i) {
    Boolean res;
    if (i==1) {
        res = ac;
    } else {
        res = pgif1(ac && (ls.get(i) == ls.get(i-1)*r), ls, r, i-1);
    }
    return res;
}
private static Boolean pgif2(Boolean ac, List<Integer> ls, Integer r, Integer i) {
    Boolean res;
    if (i==1) {
        res = ac;
    } else {
        ac = (ls.get(i) == ls.get(i-1)*r);
        if (!ac)
            res = ac;
        else
            res = pgif2(ac, ls, r, i-1);
    }
    return res;
}
```

c) Algoritmo Iterativo

```
        // Iterativo

        // Dos versiones, según se tome pgif1 o pgif2

        public static Boolean esProgGeomIter1(List<Integer> ls) {
            Boolean res;
            if (ls.size() < 2)
                res = false;
            else {
                Integer r = ls.get(1)/ls.get(0);
                Integer i = ls.size()-1;
                Boolean ac = true;
                while (i > 1) {
                    ac = ac && (ls.get(i) == ls.get(i-1)*r);
                    i--;
                }
                res = ac;
            }
            return res;
        }

        public static Boolean esProgGeomIter2(List<Integer> ls) {
            Boolean res;
            if (ls.size() < 2)
                res = false;
```

```
        else {
                Integer r = ls.get(1)/ls.get(0);
                Integer i = ls.size()-1;
                Boolean ac = true;
                while (i > 1) {
                        ac = (ls.get(i) == ls.get(i-1)*r);
                        if (!ac)
                           break;
                        i--;
                }
                res = ac;
        }
        return res;
}
```