

Tabla de contenido

Ejercicio1	2
Ejercicio1.java	2
TestEjercicio1.java	3
Resultados (consola + gráficas)	7
Ejercicio2	9
Ejercicio2.java	10
TestEjercicio2.java	11
Resultados (consola + gráficas)	16
Ejercicio3	18
Ejercicio3.java	19
TestEjercicio3.java	21
Resultados (consola)	22
Ejercicio4	23
Ejercicio4.java	23
TestEjercicio4.java	24
Resultados (consola)	25

Ejercicio1

Ejercicio1.java

```
package ejemplos;

import java.math.BigInteger;

public class Ejercicio1 {

    public static BigInteger factIterBigInteger(Integer n) {
        BigInteger factorial = BigInteger.ONE;
        int i = n;
        while(i>0) {
            factorial = factorial.multiply(BigInteger.valueOf(i));
            i--;
        }

        return factorial;
    }

    public static BigInteger factRecBigInteger(Integer n) {
        BigInteger factorial = BigInteger.ONE;

        if(n>0) {
            factorial = factRecBigInteger(n-1).multiply(BigInteger.valueOf(n));
        }

        return factorial;
    }

    public static Double factIterDouble(Integer n) {
        Double x = 1.;
        Double y = 1.;
        while(x <= n){
            y = y * x;
            x++;
        }
        return y;
    }

    public static Double factRecDouble(Integer n) {
        Double factorial = 1.;

        if(n>0) {
            factorial = factRecDouble(n-1)*n;
        }

        return factorial;
    }
}
```

TestEjercicio1.java

```

package tests;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.function.Function;

import ejemplos.Ejercicio1;
import us.lsi.common.Pair;
import us.lsi.common.Trio;
import us.lsi.curvefitting.DataCurveFitting;
import utils.GraficosAjuste;
import utils.Resultados;
import utils.TipoAjuste;

public class TestEjercicio1 {

    public static void main(String[] args) {

        System.out.println("////////////////////////////////////////\n"
            + " \t\tEjercicio_1\t\t\t \n"
            + "////////////////////////////////////////\n");
        System.out.println("Sol. Iterativa con Double: " + Ejercicio1.factIterDouble(6));
        System.out.println("Sol. Recursiva con Double: " + Ejercicio1.factRecDouble(6) +
            "\n");

        System.out.println("Sol. Iterativa con BigInteger: " +
            Ejercicio1.factIterBigInteger(6));
        System.out.println("Sol. Recursiva con BigInteger: " +
            Ejercicio1.factRecBigInteger(6));

        System.out.println("\n////////////////////////////////////////\n"
            + " \t\tEjercicio_1\t\t\t \n"
            + " \tGenerar ficheros y representar graficas\t \n"
            + "////////////////////////////////////////\n");

        //      generaFicherosTiempoEjecucionDouble();
        //      generaFicherosTiempoEjecucionBigInteger();
        //      muestraGraficas();
    }

    private static Integer nMin = 100; // n mínimo para el cálculo de potencia
    private static Integer nMaxDouble= 10000; // n máximo para el cálculo del factorial con
    DOUBLE
    private static Integer nMaxBigInteger = 5000; // n máximo para el cálculo del factorial
    con BIGINTEGER
    private static Integer numSizes = 100; // número de problemas (número de potencias
    distintas a calcular)
    private static Integer numMediciones = 10; // número de mediciones de tiempo de cada
    caso (número de experimentos)
    private static Integer numIter = 50; // número de iteraciones para cada medición de
    tiempo
    private static Integer numIterWarmup = 1000; // número de iteraciones para warmup

    // Trios de métodos a probar con su tipo de ajuste y etiqueta para el nombre de
    // los ficheros

```

```

    private static List<Trio<Function<Integer, Double>, TipoAjuste, String>> metodosDouble =
List.of(
    Trio.of(Ejercicio1::factIterDouble, TipoAjuste.POWERANB,
"factIterDouble(lineal)"),
    Trio.of(Ejercicio1::factRecDouble, TipoAjuste.POWERANB,
"factRecDouble(lineal)"));

    private static List<Trio<Function<Integer, BigInteger>, TipoAjuste, String>>
metodosBigInteger = List.of(
    Trio.of(Ejercicio1::factIterBigInteger, TipoAjuste.POWERANB,
"factIterBigInteger(lineal)"),
    Trio.of(Ejercicio1::factRecBigInteger, TipoAjuste.POWERANB,
"factRecBigInteger(lineal)"));

    public static void generaFicherosTiempoEjecucionDouble() {
        for (int i = 0; i < metodosDouble.size(); i++) {
            String ficheroSalida = String.format("ficheros/Tiempos%s.csv",
metodosDouble.get(i).third());
            testTiemposEjecucionDouble(nMin, nMaxDouble, metodosDouble.get(i).first(),
ficheroSalida);
        }
    }

    public static void generaFicherosTiempoEjecucionBigInteger() {
        for (int i = 0; i < metodosBigInteger.size(); i++) {
            String ficheroSalida = String.format("ficheros/Tiempos%s.csv",
metodosBigInteger.get(i).third());

            testTiemposEjecucionBigInteger(nMin, nMaxBigInteger,
metodosBigInteger.get(i).first(), ficheroSalida);
        }
    }

    public static void testTiemposEjecucionDouble(Integer nMin, Integer nMax,
        Function<Integer, Double> funcion,
        String ficheroTiempos
        ) {

        Map<Problema, Double> tiempos = new HashMap<Problema, Double>();
        Integer nMed = numMediciones;
        for (int iter=0; iter<nMed; iter++) {
            for (int i=0; i<numSizes; i++) {
                Double r = Double.valueOf(nMax-nMin)/(numSizes-1);
                Integer tam= (Integer.MAX_VALUE/nMax > i)
                    ? nMin + i*(nMax-nMin)/(numSizes-1)
                    : nMin + (int) (r*i) ;

                Problema p = Problema.of(tam);
                warmupDouble(funcion, 10);
                Integer nIter = numIter;
                Double[] res = new Double[nIter];
                Long t0 = System.nanoTime();
                for (int z=0; z<nIter; z++) {
                    res[z] = funcion.apply(tam);
                }
                Long t1 = System.nanoTime();
                actualizaTiempos(tiempos, p, Double.valueOf(t1-t0)/nIter);
            }
        }

        Resultados.toFile(tiempos.entrySet().stream())

```

```

        .map(x->TResultD.of(x.getKey().tam(),
                           x.getValue()))
        .map(TResultD::toString),
        ficheroTiempos, true);
    }

    public static void testTiemposEjecucionBigInteger(Integer nMin, Integer nMax,
        Function<Integer, BigInteger> funcion,
        String ficheroTiempos
    ) {

        Map<Problema, Double> tiempos = new HashMap<Problema, Double>();
        Integer nMed = numMediciones;
        for (int iter=0; iter<nMed; iter++) {
            for (int i=0; i<numSizes; i++) {
                Double r = Double.valueOf(nMax-nMin)/(numSizes-1);
                // double r = (nMax-nMin)/(numSizes-1);
                Integer tam = (Integer.MAX_VALUE/nMax > i)
                    ? nMin + i*(nMax-nMin)/(numSizes-1)
                    : nMin + (int) (r*i) ;

                Problema p = Problema.of(tam);
                warmupBigInteger(funcion, 10);
                Integer nIter = numIter;
                BigInteger[] res = new BigInteger[nIter];
                Long t0 = System.nanoTime();
                for (int z=0; z<nIter; z++) {
                    res[z] = funcion.apply(tam);
                }
                Long t1 = System.nanoTime();
                actualizaTiempos(tiempos, p, Double.valueOf(t1-t0)/nIter);
            }
        }

        ResultadosToFile(tiempos.entrySet().stream()
            .map(x->TResultD.of(x.getKey().tam(),
                               x.getValue()))
            .map(TResultD::toString),
            ficheroTiempos, true);
    }

    private static void actualizaTiempos(Map<Problema, Double> tiempos, Problema p, double
d) {
        if (!tiempos.containsKey(p)) {
            tiempos.put(p, d);
        } else if (tiempos.get(p) > d) {
            tiempos.put(p, d);
        }
    }

    private static void warmupDouble(Function<Integer, Double> pot, Integer n) {
        for (int i=0; i<numIterWarmup; i++) {
            pot.apply(n);
        }
    }

    private static void warmupBigInteger(Function<Integer, BigInteger> pot, Integer n) {
        for (int i=0; i<numIterWarmup; i++) {
            pot.apply(n);
        }
    }

```

```

    }
}

private record TResultD(Integer tam, Double t) {
    public static TResultD of(Integer tam, Double t){
        return new TResultD(tam, t);
    }

    public String toString() {
        return String.format("%d,%.0f", tam, t);
    }
}

private record Problema(Integer tam) {
    public static Problema of(Integer tam){
        return new Problema(tam);
    }
}

public static void muestraGraficas() {
    System.out.println("a*n^b*(ln n)^c + d");
    List<String> ficherosSalida = new ArrayList<>();
    List<String> labels = new ArrayList<>();

    for (int i = 0; i < metodosDouble.size(); i++) {

        String ficheroSalida = String.format("ficheros/Tiempos%s.csv",
metodosDouble.get(i).third());
        ficherosSalida.add(ficheroSalida);
        String label = metodosDouble.get(i).third();
        System.out.println(label);

        TipoAjuste tipoAjuste = metodosDouble.get(i).second();
        GraficosAjuste.show(ficheroSalida, tipoAjuste, label);

        Pair<Function<Double, Double>, String> parCurve = GraficosAjuste
            .fitCurve(DataCurveFitting.points(ficheroSalida), tipoAjuste);
        String ajusteString = parCurve.second();
        labels.add(String.format("%s      %s", label, ajusteString));

    }

    for (int i = 0; i < metodosBigInteger.size(); i++) {

        String ficheroSalida = String.format("ficheros/Tiempos%s.csv",
metodosBigInteger.get(i).third());
        ficherosSalida.add(ficheroSalida);
        String label = metodosBigInteger.get(i).third();
        System.out.println(label);

        TipoAjuste tipoAjuste = metodosBigInteger.get(i).second();
        GraficosAjuste.show(ficheroSalida, tipoAjuste, label);

        Pair<Function<Double, Double>, String> parCurve = GraficosAjuste
            .fitCurve(DataCurveFitting.points(ficheroSalida), tipoAjuste);
        String ajusteString = parCurve.second();
        labels.add(String.format("%s      %s", label, ajusteString));

    }

    GraficosAjuste.showCombined("Factorial", ficherosSalida, labels);
}

```

```
}
}
```

Resultados (consola + gráficas)

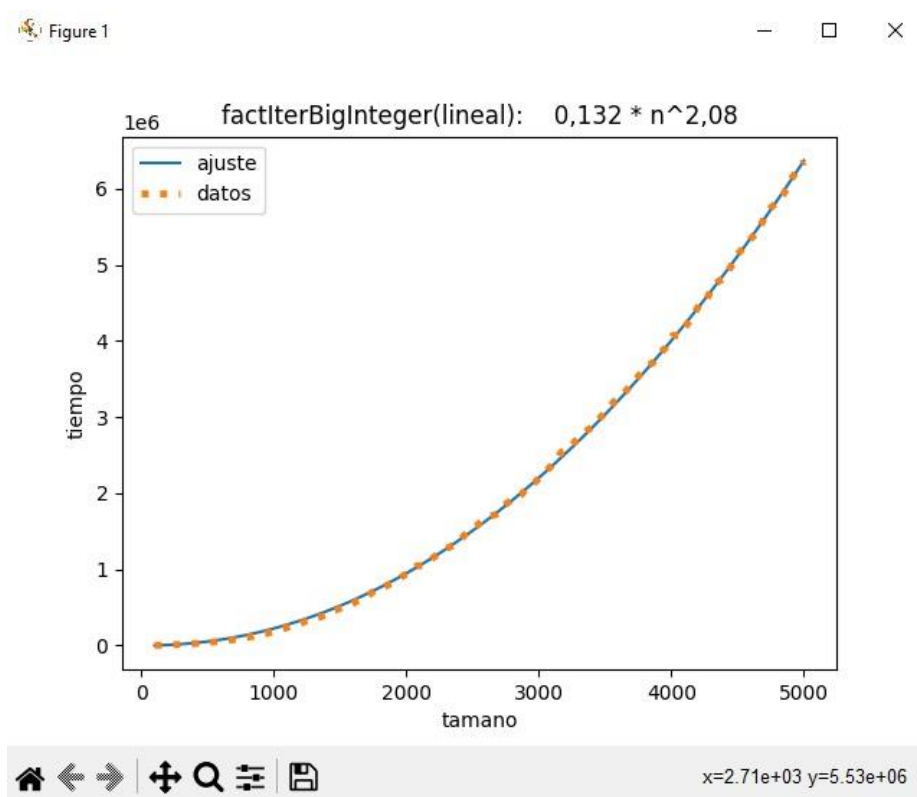
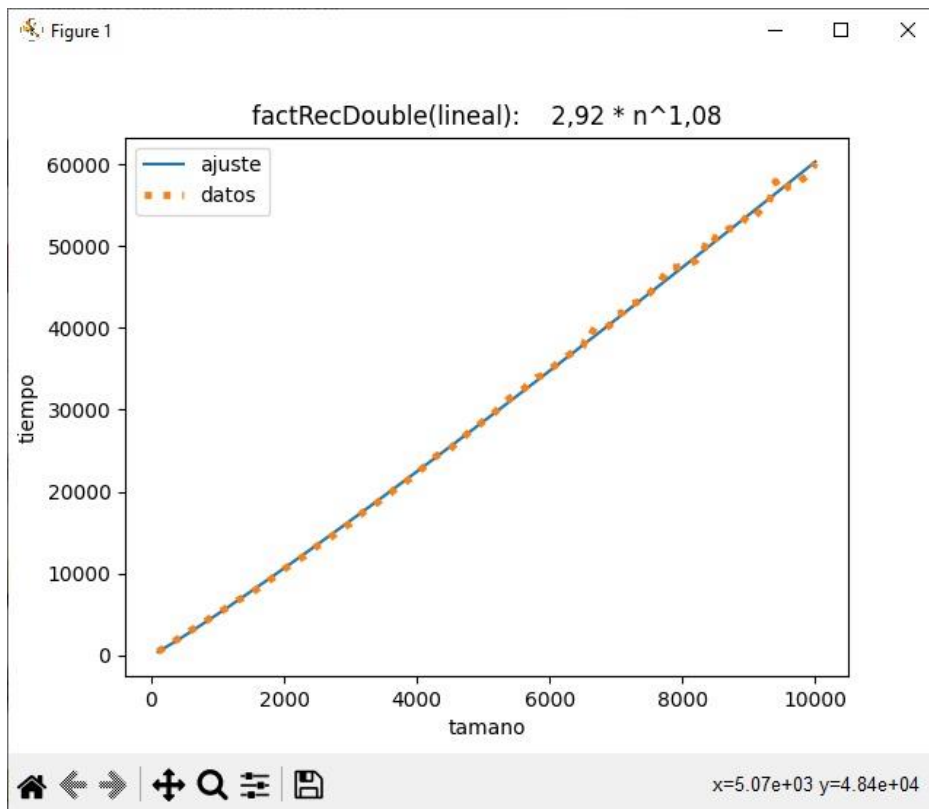
```
#####
Ejercicio 1
#####

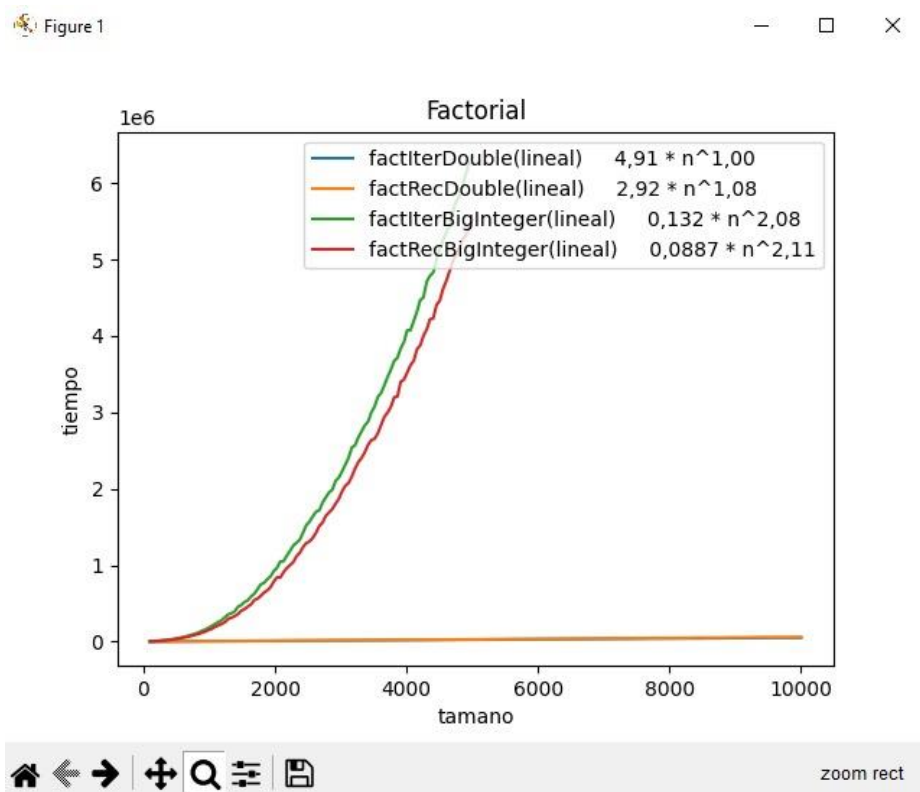
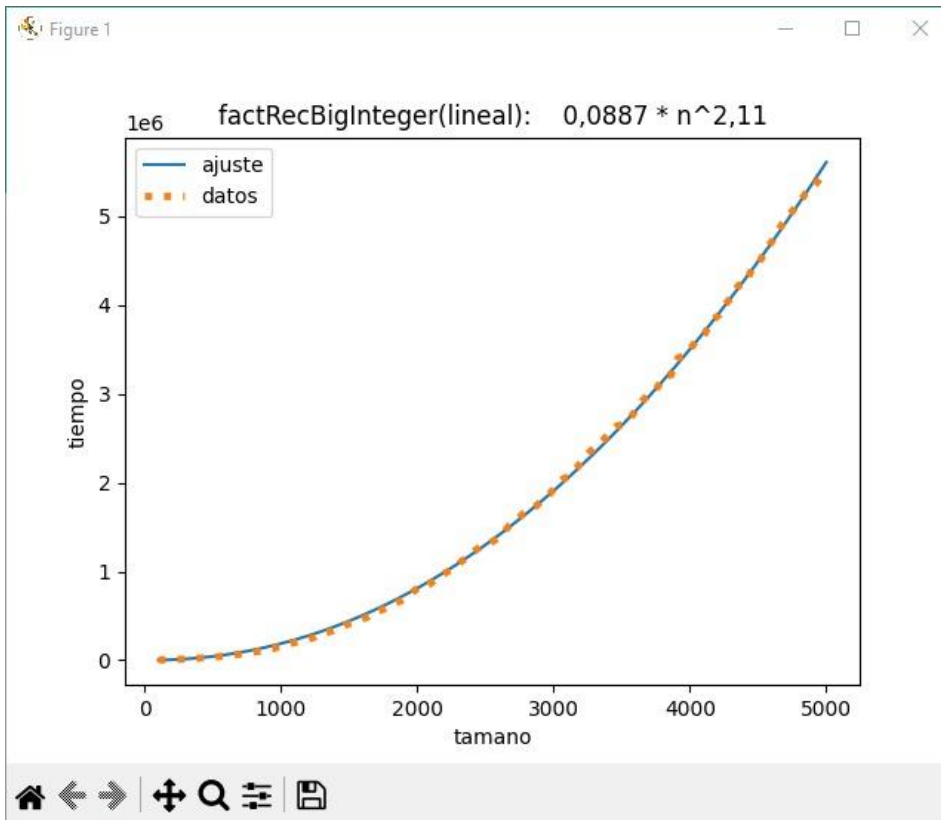
Sol. Iterativa con Double: 720.0
Sol. Recursiva con Double: 720.0

Sol. Iterativa con BigInteger: 720
Sol. Recursiva con BigInteger: 720

#####
Ejercicio 1
#####
Generar ficheros y representar gráficas

a*n*b*(ln n)^c + d
factIterDouble(lineal)
Solutions = a = 4,91,b = 1,00
DEBUG - .plot command: ret_33443d12_7bfb_41c7_bf25_9f6d9745200 = plt.plot(np.array([100.0, 200.0, 300.0, 400.0, 500.0, 600.0, 700.0, 800.0, 900.0, 1000.0, 1100.0, 1200.0, 1300.0, 1400.0, 1500.0, 1600.0, 1700.0, 1800.0, 1900.0, 2000.0, 2100.0, 2200.0, 2300.0, 2400.0, 2500.0, 2600.0, 2700.0, 2800.0, 2900.0, 3000.0, 3100.0, 3200.0, 3300.0, 3400.0, 3500.0, 3600.0, 3700.0, 3800.0, 3900.0, 4000.0, 4100.0, 4200.0, 4300.0, 4400.0, 4500.0, 4600.0, 4700.0, 4800.0, 4900.0, 5000.0, 5100.0, 5200.0, 5300.0, 5400.0, 5500.0, 5600.0, 5700.0, 5800.0, 5900.0, 6000.0, 6100.0, 6200.0, 6300.0, 6400.0, 6500.0, 6600.0, 6700.0, 6800.0, 6900.0, 7000.0, 7100.0, 7200.0, 7300.0, 7400.0, 7500.0, 7600.0, 7700.0, 7800.0, 7900.0, 8000.0, 8100.0, 8200.0, 8300.0, 8400.0, 8500.0, 8600.0, 8700.0, 8800.0, 8900.0, 9000.0, 9100.0, 9200.0, 9300.0, 9400.0, 9500.0, 9600.0, 9700.0, 9800.0, 9900.0, 10000.0]))
DEBUG - .plot command: plt.title("factIterDouble(lineal): 4,91 * n^1,00")
DEBUG - .plot command: ret_66c1fad0_376c_4b29_b977_233b1 = plt.legend()
DEBUG - .plot command: ret_c909625_0c19_40bc_be61_0e86219d2c8e = plt.xlabel("tamano")
DEBUG - .plot command: ret_3ea8d972_7544_4a53_a0e9_c0dc6f33ef37 = plt.ylabel("tiempo")
DEBUG - Commands... : [C:\Users\Alvaro\AppData\Local\Programs\Python\Python310\python.exe, C:\Users\Alvaro\AppData\Local\Temp\1669058754986-0\exec.py]
WARN -
factIterDouble(lineal)
Solutions = a = 4,91,b = 1,00
factRecDouble(lineal)
Solutions = a = 2,92,b = 1,08
DEBUG - .plot command: ret_2d4d0a8f_b4e5_4727_995c_e28960c080ca = plt.plot(np.array([100.0, 200.0, 300.0, 400.0, 500.0, 600.0, 700.0, 800.0, 900.0, 1000.0, 1100.0, 1200.0, 1300.0, 1400.0, 1500.0, 1600.0, 1700.0, 1800.0, 1900.0, 2000.0, 2100.0, 2200.0, 2300.0, 2400.0, 2500.0, 2600.0, 2700.0, 2800.0, 2900.0, 3000.0, 3100.0, 3200.0, 3300.0, 3400.0, 3500.0, 3600.0, 3700.0, 3800.0, 3900.0, 4000.0, 4100.0, 4200.0, 4300.0, 4400.0, 4500.0, 4600.0, 4700.0, 4800.0, 4900.0, 5000.0, 5100.0, 5200.0, 5300.0, 5400.0, 5500.0, 5600.0, 5700.0, 5800.0, 5900.0, 6000.0, 6100.0, 6200.0, 6300.0, 6400.0, 6500.0, 6600.0, 6700.0, 6800.0, 6900.0, 7000.0, 7100.0, 7200.0, 7300.0, 7400.0, 7500.0, 7600.0, 7700.0, 7800.0, 7900.0, 8000.0, 8100.0, 8200.0, 8300.0, 8400.0, 8500.0, 8600.0, 8700.0, 8800.0, 8900.0, 9000.0, 9100.0, 9200.0, 9300.0, 9400.0, 9500.0, 9600.0, 9700.0, 9800.0, 9900.0, 10000.0]))
DEBUG - .plot command: plt.title("factRecDouble(lineal): 2,92 * n^1,08")
DEBUG - .plot command: ret_d0259c1a_1023_4e94_82cb_ec247ab5809f = plt.legend()
DEBUG - .plot command: ret_7d66c283_88c2_4cbe_99d2_bbd8a645eb4e = plt.xlabel("tamano")
DEBUG - .plot command: ret_7c4d0baf_f0f2_4f3b_ac69_0982b39b3b27 = plt.ylabel("tiempo")
DEBUG - Commands... : [C:\Users\Alvaro\AppData\Local\Programs\Python\Python310\python.exe, C:\Users\Alvaro\AppData\Local\Temp\1669058757852-0\exec.py]
WARN -
factRecDouble(lineal)
Solutions = a = 2,92,b = 1,08
factIterBigInteger(lineal)
Solutions = a = 0,13,b = 2,08
DEBUG - .plot command: ret_2c6d0726_8895_453d_867d_163bbefeb0d5 = plt.plot(np.array([100.0, 149.0, 198.0, 248.0, 297.0, 347.0, 396.0, 446.0, 495.0, 545.0, 594.0, 644.0, 693.0, 743.0, 792.0, 842.0, 891.0, 941.0, 990.0, 1040.0, 1089.0, 1139.0, 1188.0, 1238.0, 1287.0, 1337.0, 1386.0, 1436.0, 1485.0, 1534.0, 1584.0, 1633.0, 1683.0, 1732.0, 1781.0, 1831.0, 1880.0, 1929.0, 1979.0, 2028.0, 2077.0, 2127.0, 2176.0, 2225.0, 2275.0, 2324.0, 2373.0, 2423.0, 2472.0, 2521.0, 2571.0, 2620.0, 2669.0, 2718.0, 2768.0, 2817.0, 2866.0, 2916.0, 2965.0, 3014.0, 3064.0, 3113.0, 3162.0, 3212.0, 3261.0, 3310.0, 3359.0, 3409.0, 3458.0, 3507.0, 3557.0, 3606.0, 3655.0, 3705.0, 3754.0, 3803.0, 3853.0, 3902.0, 3951.0, 4001.0, 4050.0, 4099.0, 4149.0, 4198.0, 4247.0, 4297.0, 4346.0, 4395.0, 4445.0, 4494.0, 4543.0, 4593.0, 4642.0, 4691.0, 4741.0, 4790.0, 4839.0, 4888.0, 4938.0, 4987.0, 5036.0, 5086.0, 5135.0, 5184.0, 5234.0, 5283.0, 5332.0, 5382.0, 5431.0, 5480.0, 5530.0, 5579.0, 5628.0, 5678.0, 5727.0, 5776.0, 5826.0, 5875.0, 5924.0, 5974.0, 6023.0, 6073.0, 6122.0, 6171.0, 6221.0, 6270.0, 6319.0, 6369.0, 6418.0, 6467.0, 6517.0, 6566.0, 6615.0, 6665.0, 6714.0, 6763.0, 6813.0, 6862.0, 6911.0, 6961.0, 7010.0, 7059.0, 7109.0, 7158.0, 7207.0, 7257.0, 7306.0, 7355.0, 7405.0, 7454.0, 7503.0, 7553.0, 7602.0, 7651.0, 7701.0, 7750.0, 7799.0, 7849.0, 7898.0, 7947.0, 7997.0, 8046.0, 8095.0, 8145.0, 8194.0, 8243.0, 8293.0, 8342.0, 8391.0, 8441.0, 8490.0, 8539.0, 8589.0, 8638.0, 8687.0, 8737.0, 8786.0, 8835.0, 8885.0, 8934.0, 8983.0, 9033.0, 9082.0, 9131.0, 9181.0, 9230.0, 9279.0, 9329.0, 9378.0, 9427.0, 9476.0, 9526.0, 9575.0, 9624.0, 9674.0, 9723.0, 9772.0, 9822.0, 9871.0, 9920.0, 9970.0, 10019.0, 10068.0, 10118.0, 10167.0, 10216.0, 10266.0, 10315.0, 10364.0, 10414.0, 10463.0, 10512.0, 10562.0, 10611.0, 10660.0, 10710.0, 10759.0, 10808.0, 10858.0, 10907.0, 10956.0, 11006.0, 11055.0, 11104.0, 11154.0, 11203.0, 11252.0, 11302.0, 11351.0, 11400.0, 11450.0, 11499.0, 11548.0, 11598.0, 11647.0, 11696.0, 11746.0, 11795.0, 11844.0, 11894.0, 11943.0, 11992.0, 12042.0, 12091.0, 12140.0, 12190.0, 12239.0, 12288.0, 12338.0, 12387.0, 12436.0, 12486.0, 12535.0, 12584.0, 12634.0, 12683.0, 12732.0, 12782.0, 12831.0, 12880.0, 12930.0, 12979.0, 13028.0, 13078.0, 13127.0, 13176.0, 13226.0, 13275.0, 13324.0, 13374.0, 13423.0, 13472.0, 13522.0, 13571.0, 13620.0, 13670.0, 13719.0, 13768.0, 13818.0, 13867.0, 13916.0, 13966.0, 14015.0, 14064.0, 14114.0, 14163.0, 14212.0, 14262.0, 14311.0, 14360.0, 14410.0, 14459.0, 14508.0, 14558.0, 14607.0, 14656.0, 14706.0, 14755.0, 14804.0, 14854.0, 14903.0, 14952.0, 15002.0, 15051.0, 15100.0, 15150.0, 15199.0, 15248.0, 15298.0, 15347.0, 15396.0, 15446.0, 15495.0, 15544.0, 15594.0, 15643.0, 15693.0, 15742.0, 15791.0, 15841.0, 15890.0, 15939.0, 15989.0, 16038.0, 16087.0, 16137.0, 16186.0, 16235.0, 16285.0, 16334.0, 16383.0, 16433.0, 16482.0, 16531.0, 16581.0, 16630.0, 16679.0, 16729.0, 16778.0, 16827.0, 16877.0, 16926.0, 16975.0, 17025.0, 17074.0, 17123.0, 17173.0, 17222.0, 17271.0, 17321.0, 17370.0, 17419.0, 17468.0, 17518.0, 17567.0, 17616.0, 17666.0, 17715.0, 17764.0, 17814.0, 17863.0, 17912.0, 17962.0, 18011.0, 18060.0, 18110.0, 18159.0, 18208.0, 18258.0, 18307.0, 18356.0, 18406.0, 18455.0, 18504.0, 18554.0, 18603.0, 18652.0, 18702.0, 18751.0, 18800.0, 18850.0, 18899.0, 18948.0, 19000.0, 19048.0, 19097.0, 19147.0, 19196.0, 19245.0, 19295.0, 19344.0, 19393.0, 19443.0, 19492.0, 19541.0, 19591.0, 19640.0, 19689.0, 19738.0, 19788.0, 19837.0, 19886.0, 19936.0, 19985.0, 20034.0, 20084.0, 20133.0, 20182.0, 20232.0, 20281.0, 20330.0, 20380.0, 20429.0, 20478.0, 20528.0, 20577.0, 20626.0, 20676.0, 20725.0, 20774.0, 20824.0, 20873.0, 20922.0, 20972.0, 21021.0, 21070.0, 21120.0, 21169.0, 21218.0, 21268.0, 21317.0, 21366.0, 21416.0, 21465.0, 21514.0, 21564.0, 21613.0, 21662.0, 21712.0, 21761.0, 21810.0, 21860.0, 21909.0, 21958.0, 22008.0, 22057.0, 22106.0, 22156.0, 22205.0, 22254.0, 22304.0, 22353.0, 22402.0, 22452.0, 22501.0, 22550.0, 22600.0, 22649.0, 22698.0, 22748.0, 22797.0, 22846.0, 22896.0, 22945.0, 22994.0, 23044.0, 23093.0, 23142.0, 23192.0, 23241.0, 23290.0, 23340.0, 23389.0, 23438.0, 23488.0, 23537.0, 23586.0, 23636.0, 23685.0, 23734.0, 23784.0, 23833.0, 23882.0, 23932.0, 23981.0, 24030.0, 24080.0, 24129.0, 24178.0, 24228.0, 24277.0, 24326.0, 24376.0, 24425.0, 24474.0, 24524.0, 24573.0, 24622.0, 24672.0, 24721.0, 24770.0, 24820.0, 24869.0, 24918.0, 24968.0, 25017.0, 25066.0, 25116.0, 25165.0, 25214.0, 25264.0, 25313.0, 25362.0, 25412.0, 25461.0, 25510.0, 25560.0, 25609.0, 25658.0, 25708.0, 25757.0, 25806.0, 25856.0, 25905.0, 25954.0, 26004.0, 26053.0, 26103.0, 26152.0, 26201.0, 26251.0, 26300.0, 26349.0, 26399.0, 26448.0, 26497.0, 26547.0, 26596.0, 26645.0, 26695.0, 26744.0, 26793.0, 26843.0, 26892.0, 26941.0, 26991.0, 27040.0, 27089.0, 27139.0, 27188.0, 27237.0, 27287.0, 27336.0, 27385.0, 27435.0, 27484.0, 27533.0, 27583.0, 27632.0, 27681.0, 27731.0, 27780.0, 27829.0, 27879.0, 27928.0, 27977.0, 28027.0, 28076.0, 28125.0, 28175.0, 28224.0, 28273.0, 28323.0, 28372.0, 28421.0, 28471.0, 28520.0, 28569.0, 28619.0, 28668.0, 28717.0, 28767.0, 28816.0, 28865.0, 28915.0, 28964.0, 29013.0, 29063.0, 29112.0, 29161.0, 29211.0, 29260.0, 29309.0, 29359.0, 29408.0, 29457.0, 29507.0, 29556.0, 29605.0, 29655.0, 29704.0, 29753.0, 29803.0, 29852.0, 29901.0, 29951.0, 30000.0, 30049.0, 30099.0, 30148.0, 30197.0, 30247.0, 30296.0, 30345.0, 30395.0, 30444.0, 30493.0, 30543.0, 30592.0, 30641.0, 30691.0, 30740.0, 30789.0, 30839.0, 30888.0, 30937.0, 30987.0, 31036.0, 31085.0, 31135.0, 31184.0, 31233.0, 31283.0, 31332.0, 31381.0, 31431.0, 31480.0, 31529.0, 31579.0, 31628.0, 31677.0, 31727.0, 31776.0, 31825.0, 31875.0, 31924.0, 31973.0, 32023.0, 32072.0, 32121.0, 32171.0, 32220.0, 32269.0, 32319.0, 32368.0, 32417.0, 32467.0, 32516.0, 32565.0, 32615.0, 32664.0, 32713.0, 32763.0, 32812.0, 32861.0, 32911.0, 32960.0, 33009.0, 33059.0, 33108.0, 33157.0, 33207.0, 33256.0, 33305.0, 33355.0, 33404.0, 33453.0, 33503.0, 33552.0, 33601.0, 33651.0, 33700.0, 33749.0, 33799.0, 33848.0, 33897.0, 33946.0, 33996.0, 34045.0, 34094.0, 34144.0, 34193.0, 34242.0, 34292.0, 34341.0, 34390.0, 34440.0, 34489.0, 34538.0, 34588.0, 34637.0, 34686.0, 34736.0, 34785.0, 34834.0, 34884.0, 34933.0, 34982.0, 35032.0, 35081.0, 35130.0, 35180.0, 35229.0, 35278.0, 35328.0, 35377.0, 35426.0, 35476.0, 35525.0, 35574.0, 35624.0, 35673.0, 35722.0, 35772.0, 35821.0, 35870.0, 35920.0, 35969.0, 36018.0, 36068.0, 36117.0, 36166.0, 36216.0, 36265.0, 36314.0, 36364.0, 36413.0, 36462.0, 36512.0, 36561.0, 36610.0, 36660.0, 36709.0, 36758.0, 36808.0, 36857.0, 36906.0, 36956.0, 37005.0, 37054.0, 37104.0, 37153.0, 37202.0, 37252.0, 37301.0, 37350.0, 37400.0, 37449.0, 37498.0, 37548.0, 37597.0, 37646.0, 37696.0, 37745.0, 37794.0, 37844.0, 37893.0, 37942.0, 37992.0, 38041.0, 38090.0, 38140.0, 38189.0, 38238.0, 38288.0, 38337.0, 38386.0, 38436.0, 38485.0, 38534.0, 38584.0, 38633.0, 38682.0, 38732.0, 38781.0, 38830.0, 38880.0, 38929.0, 38978.0, 39028.0, 39077.0, 39126.0, 39176.0, 39225.0, 39274.0, 39324.0, 39373.0, 39422.0, 39472.0, 39521.0, 39570.0, 39620.0, 39669.0, 39718.0, 39768.0, 39817.0, 39866.0, 39916.0, 39965.0, 40014.0, 40064.0, 40113.0, 40162.0, 40212.0, 40261.0, 40310.0, 40360.0, 40409.0, 40458.0, 40508.0, 40557.0, 40606.0, 40656.0, 40705.0, 40754.0, 40804.0, 40853.0, 40902.0, 40952.0, 40999.0, 41049.0, 41098.0, 41147.0, 41196.0, 41246.0, 41295.0, 41344.0, 41394.0, 41443.0, 41492.0, 41542.0, 41591.0, 41640.0, 41690.0, 41739.0, 41788.0, 41838.0, 41887.0, 41936.0, 41986.0, 42035.0, 42084.0, 42134.0, 42183.0, 42232.0, 42282.0, 42331.0, 42380.0, 42430.0, 42479.0, 42528.0, 42578.0, 42627.0, 42676.0, 42726.0, 42775.0, 42824.0, 42874.0, 42923.0, 42972.0, 43022.0, 43071.0, 43120.0, 43170.0, 43219.0, 43268.0, 43318.0, 43367.0, 43416.0, 43466.0, 43515.0, 43564.0, 43614.0, 43663.0, 43712.0, 43762.0, 43811.0, 43860.0, 43910.0, 43959.0, 44008.0, 44058.0, 44107.0, 44156.0, 44206.0, 44255.0, 44304.0, 44354.0, 44403.0, 44452.0, 44502.0, 44551.0, 44600.0, 44650.0, 44699.0, 44748.0, 44798.0, 44847.0, 44896.0, 44946.0, 44995.0, 45044.0, 45094.0, 45143.0, 45192.0, 45242.0, 45291.0, 45340.0, 45390.0, 45439.0, 45488.0, 45538.0, 45587.0, 45636.0, 45686.0, 45735.0, 45784.0, 45834.0, 45883.0, 45932.0, 45982.0, 46031.0, 46080.0, 46130.0, 46179.0, 46228.0, 46278.0, 46327.0, 46376.0, 46426.0, 46475.0, 46524.0, 46574.0, 46623.0, 46673.0, 46722.0, 46771.0, 46821.0, 46870.0, 46919.0, 46969.0, 47018.0, 47067.0, 47117.0, 47166.0, 47215.0, 47265.0, 47314.0, 47363.0, 47413.0, 47462.0, 47511.0, 47561.0, 47610.0, 47659.0, 47708.0, 47758.0, 47807.0, 47856.0, 47906.0, 47955.0, 48004.0, 48054.0, 48103.0, 48152.0, 48202.0, 48251.0, 48300.0, 48350.0, 48399.0, 48448.0, 48498.0, 48547.0, 48596.0, 48646.0, 48695.0, 48744.0, 48794.0, 48843.0, 48892.0, 48942.0, 48991.0, 49040.0, 49090.0, 49139.0, 49188.0, 49238.0, 49287.0, 49336.0, 49386.0, 49435.0, 49484.0, 49534.0, 49583.0, 49632.0, 49682.0, 49731.0, 49780.0, 49830.0, 49879.0, 49928.0, 49978.0, 50027.0, 50076.0, 50126.0, 50175.0, 50224.0, 50274.0, 50323.0, 50372.0, 50422.0, 50471.0, 50520.0, 50570.0, 50619.0, 50668.0, 50718.0, 50767.0, 50816.0, 50866.0, 50915.0, 50964.0, 51014.0, 51063.0, 51112.0, 51162.0, 51211.0, 51260.0, 51310.0, 51359.0, 51408.0, 51458.0, 51507.0, 51556.0, 51606.0, 51655.0, 51704.0, 51754.0, 51803.0, 51852.0, 51902.0, 51951.0, 51999.0, 52049.0, 52098.0, 52147.0, 52197.0, 52246.0, 52295.0, 52345.0, 52394.0, 52443.0, 52493.0, 52542.0, 52591.0, 52641.0, 52690.0, 52739.0, 52789.0, 52838.0, 52887.0, 52937.0, 52986.0, 53035.0, 53085.0, 53134.0, 53183.0, 53233.0, 53282.0, 53331.0, 53381.0, 53430.0, 53479.0, 53529.0, 53578.0, 53627.0, 53677.0, 53726.0, 53775.0, 53825.0, 53874.0, 53923.0, 53973.0, 54022.0, 54071.0, 54121.0, 54170.0, 54219.0, 54269.0, 54318.0, 54367.0, 54417.0, 54466.0, 54515.0, 54565.0, 54614.0, 54663.0, 54713.0, 54762.0, 54811.0, 54861.0, 54910.0, 54959.0, 55009.0, 55058.0, 55107.0, 55156.0, 5520
```





Ejercicio2

Ejercicio2.java

```
package ejemplos;

import java.util.Comparator;
import java.util.List;

import us.lsi.common.IntPair;
import us.lsi.common.List2;
import us.lsi.common.Preconditions;
import us.lsi.math.Math2;

public class Ejercicio2 {

    public static <E extends Comparable<? super E>> void sort(List<E> lista, Integer umbral)
    {
        Comparator<? super E> ord = Comparator.naturalOrder();
        quickSort(lista, 0, lista.size(), ord, umbral);
    }

    private static <E> void quickSort(List<E> lista, int i, int j, Comparator<? super E>
ord, Integer umbral) {
        Preconditions.checkArgument(j >= i);
        if (j - i <= umbral) {
            ordenaBase(lista, i, j, ord);
        } else {
            E pivote = escogePivote(lista, i, j);
            IntPair p = banderaHolandesa(lista, pivote, i, j, ord);
            quickSort(lista, i, p.first(), ord, umbral);
            quickSort(lista, p.second(), j, ord, umbral);
        }
    }

    public static <T> void ordenaBase(List<T> lista, Integer inf, Integer sup, Comparator<?
super T> ord) {
        for (int i = inf; i < sup; i++) {
            for (int j = i + 1; j < sup; j++) {
                if (ord.compare(lista.get(i), lista.get(j)) > 0) {
                    List2.intercambia(lista, i, j);
                }
            }
        }
    }

    private static <E> E escogePivote(List<E> lista, int i, int j) {
        E pivote = lista.get(Math2.getEnteroAleatorio(i, j));
        return pivote;
    }

    public static <E> IntPair banderaHolandesa(List<E> ls, E pivote, Integer i, Integer j,
Comparator<? super E> cmp) {
        Integer a = i, b = i, c = j;
        while (c - b > 0) {
            E elem = ls.get(b);
            if (cmp.compare(elem, pivote) < 0) {
                List2.intercambia(ls, a, b);
                a++;
                b++;
            }
        }
    }
}
```

```

        } else if (cmp.compare(elem, pivote) > 0) {
            List2.intercambia(ls, b, c - 1);
            c--;
        } else {
            b++;
        }
    }
    return IntPair.of(a,b);
}
}

```

TestEjercicio2.java

```

package tests;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Random;
import java.util.function.BiConsumer;
import java.util.function.Function;
import java.util.stream.Collectors;

import ejemplos.Ejercicio2;
import us.lsi.common.Files2;
import us.lsi.common.List2;
import us.lsi.common.Pair;
import us.lsi.common.Trio;
import us.lsi.curvefitting.DataCurveFitting;
import utils.GraficosAjuste;
import utils.Resultados;
import utils.TipoAjuste;

public class TestEjercicio2 {

    private static Integer numMediciones = 2; // número de mediciones de tiempo de cada caso
    (número de experimentos)
    private static Integer numIter = 5; // número de iteraciones para cada medición de
    tiempo
    private static Integer numIterWarmup = 50; // número de iteraciones para warmup

    private static Integer numListas = 30; // Número de listas
    private static Integer sizeMin = 50; // Tamaño mínimo de listas
    private static Integer sizeMax = 50000; // Tamaño máximo de listas
    private static Integer umbral = 4; // Umbral fijo para análisis de complejidad

    private static Integer numUmbrales = 30; // número de umbrales para análisis de
    distintos umbrales
    private static Integer nMinUmbral = 1; // n mínimo umbral
    private static Integer nMaxUmbral = 100; // n máximo umbral

    private static Random rr = new Random(System.nanoTime()); // para inicializarlo una sola
    vez y compartirlo con los metodos que lo requieran

    private static String ficheroListaEntrada = "ficheros/ListasAlgSort.txt";

```

```

        private static List<Trio<BiConsumer<List<Integer>, Integer>, TipoAjuste, String>>
metodosComplejidad = List.of(
            Trio.of(Ejercicio2::sort, TipoAjuste.NLOGN_0, "Quicksort(complejidad)") //
Análisis de complejidad del algoritmo de ordenación con listas de distintos tamaños y umbral
fijo
        );

        private static List<Trio<BiConsumer<List<Integer>, Integer>, TipoAjuste, String>>
metodosComplejidadUmbrales = List.of(
            Trio.of(Ejercicio2::sort, TipoAjuste.NLOGN_0,
"Quicksort_50k_u4(complejidad)", // Análisis de complejidad del algoritmo de ordenación con
listas de distintos tamaños y umbral fijo
            Trio.of(Ejercicio2::sort, TipoAjuste.NLOGN_0,
"Quicksort_50k_u25(complejidad)",
            Trio.of(Ejercicio2::sort, TipoAjuste.NLOGN_0,
"Quicksort_50k_u100(complejidad)",
            Trio.of(Ejercicio2::sort, TipoAjuste.NLOGN_0,
"Quicksort_50k_u500(complejidad)")
        );

        private static List<Trio<BiConsumer<List<Integer>, Integer>, TipoAjuste, String>>
metodosUmbral = List.of(
            Trio.of(Ejercicio2::sort, TipoAjuste.NLOGN_0, "Quicksort(analisisumbral)")
// Análisis del tamaño umbral con una única lista de tamaño sizeMax
        );

    public static void main(String[] args) {
        // generaFicheroListasEnteros(ficheroListaEntrada);
        // generaFicherosTiempoEjecucion(metodosComplejidad);
        // generaFicherosTiempoEjecucion(metodosUmbral);
        // generaFicherosTiempoEjecucion(metodosComplejidadUmbrales);
        muestraGraficas(metodosComplejidad, "Complejidad ordenación");
        muestraGraficas(metodosUmbral, "Análisis del tamaño del umbral");
        muestraGraficasDistintosUmbralesQS();
    }

    public static void generaFicheroListasEnteros(String fichero) {
        Resultados.cleanFile(fichero);
        for (int i=0; i<numListas; i++) {
            int div = numListas<2? 1:(numListas-1);
            int tam = sizeMin + i*(sizeMax-sizeMin)/div;
            List<Integer> ls = generaListaEnteros(tam);
            String sls = ls.stream().map(x-
>x.toString()).collect(Collectors.joining(","));
            ResultadosToFile(String.format("%d#%s",tam,sls), fichero, false);
        }
    }

    public static List<Integer> generaListaEnteros(Integer sizeList) {
        List<Integer> ls = new ArrayList<Integer>();
        for (int i=0;i<sizeList;i++) {
            ls.add(0+rr.nextInt(1000000-0));
        }
        return ls;
    }
}

```

```

    public static void muestraGraficas(List<Trio<BiConsumer<List<Integer>, Integer>,
TipoAjuste, String>> metodos, String title) {

        System.out.println("a*n^b*(ln n)^c + d");
        List<String> ficherosSalida = new ArrayList<>();
        List<String> labels = new ArrayList<>();

        for (int i=0; i<metodos.size(); i++) {

            String ficheroSalida = String.format("ficheros/Tiempos%s.csv",
                metodos.get(i).third());
            String label = metodos.get(i).third();
            ficherosSalida.add(ficheroSalida);
            System.out.println(label);
            TipoAjuste tipoAjuste = metodos.get(i).second();

            if(!ficheroSalida.contains("umbral")) {
                GraficosAjuste.show(ficheroSalida, tipoAjuste, label);
                // Obtener ajusteString para mostrarlo en gráfica combinada
                Pair<Function<Double, Double>, String> parCurve =
GraficosAjuste.fitCurve(
                    DataCurveFitting.points(ficheroSalida), tipoAjuste);
                String ajusteString = parCurve.second();
                labels.add(String.format("%s      %s", label, ajusteString));
            } else {
                labels.add(label);
            }

        }

        GraficosAjuste.showCombined(title, ficherosSalida, labels);
    }

    public static void muestraGraficasDistintosUmbralesQS() {
        List<String> ficherosSalida =
List.of("ficheros/TiemposQuicksort_50k_u4(complejidad).csv",
        "ficheros/TiemposQuicksort_50k_u25(complejidad).csv",
        "ficheros/TiemposQuicksort_50k_u100(complejidad).csv",
        "ficheros/TiemposQuicksort_50k_u500(complejidad).csv");
        List<String> labels = List.of("Umbral 4", "Umbral 25", "Umbral 100", "Umbral
500");

        GraficosAjuste.showCombined("Comparativa complejidad Quicksort con distintos
umbrales", ficherosSalida, labels);
    }

    public static void generaFicherosTiempoEjecucion(List<Trio<BiConsumer<List<Integer>,
Integer>, TipoAjuste, String>> metodos) {
        for (int i=0; i<metodos.size(); i++) {

            String ficheroSalida = String.format("ficheros/Tiempos%s.csv",
                metodos.get(i).third());
            System.out.println(ficheroSalida);
            if(!ficheroSalida.contains("umbral")) {
                testTiemposEjecucionComplejidad(
                    metodos.get(i).first(),
                    ficheroSalida

```

```

        );
    }
    else {
        testTiemposEjecucionUmbral(
            metodos.get(i).first(),
            ficheroSalida
        );
    }
}

}

public static void testTiemposEjecucionComplejidad(
    BiConsumer<List<Integer>, Integer> funcion,
    String ficheroTiempos
) {

    Map<Problema, Double> tiempos = new HashMap<Problema, Double>();
    Map<Integer, Double> tiemposMedios; // tiempos medios por cada tamaño

    List<String> lineasListas = Files2.linesFromFile(ficheroListaEntrada);

    Integer nMed = numMediciones;
    for (int iter=0; iter<nMed; iter++) {
        for (int i=0; i<lineasListas.size(); i++) {
            System.out.println(iter + " " + i);
            String lineaLista = lineasListas.get(i);
            List<String> ls = List2.parse(lineaLista, "#", Function.identity());
            Integer tamLista = Integer.parseInt(ls.get(0));
            List<Integer> le = List2.parse(ls.get(1), ",", Integer::parseInt);

            Problema p = Problema.of(tamLista, i, tamLista);
            warmup(funcion, le, 4);

            Integer nIter = numIter;
            Long t0 = System.nanoTime();
            for (int z=0; z<nIter; z++) {
                funcion.accept(le, umbral);
            }
            Long t1 = System.nanoTime();
            actualizaTiempos(tiempos, p, Double.valueOf(t1-t0)/nIter);
        }
    }

    tiemposMedios = tiempos.entrySet().stream()
        .collect(Collectors.groupingBy(x->x.getKey().tam(),
            Collectors.averagingDouble(x->x.getValue())
        ));

    Resultados.toFile(tiemposMedios.entrySet().stream()
        .map(x->TResultMedD.of(x.getKey(), x.getValue()))
        .map(TResultMedD::toString),
        ficheroTiempos,
        true);
}

```

```

public static void testTiemposEjecucionUmbral(
    BiConsumer<List<Integer>, Integer> funcion,
    String ficheroTiempos
) {

    Map<Problema, Double> tiempos = new HashMap<Problema, Double>();
    Map<Integer, Double> tiemposMedios; // tiempos medios por cada tamaño

    List<String> lineasListas = Files2.linesFromFile(ficheroListaEntrada);

    Integer nMed = numMediciones;
    for (int iter=0; iter<nMed; iter++) {
        String lineaLista = lineasListas.get(lineasListas.size()-1);
        List<String> ls = List2.parse(lineaLista, "#", Function.identity());
        Integer tamLista = Integer.parseInt(ls.get(0));
        List<Integer> le = List2.parse(ls.get(1), ",", Integer::parseInt);
        for (int j=0; j<numUmbrales; j++) {
            System.out.println(j);
            Double r = Double.valueOf(nMaxUmbral-nMinUmbral)/(numUmbrales-1);
            Integer tam = (Integer.MAX_VALUE/nMaxUmbral > j)
                ? nMinUmbral + j*(nMaxUmbral-nMinUmbral)/(numUmbrales-1)
                : nMinUmbral + (int) (r*j) ;

            Problema p = Problema.of(tam,0,tamLista);
            warmup(funcion, le, 4);

            Integer nIter = numIter;
            Long t0 = System.nanoTime();
            for (int z=0; z<nIter; z++) {
                funcion.accept(le, tam);
            }
            Long t1 = System.nanoTime();
            actualizaTiempos(tiempos, p, Double.valueOf(t1-t0)/nIter);

        }
    }

    tiemposMedios = tiempos.entrySet().stream()
        .collect(Collectors.groupingBy(x->x.getKey().tam(),
            Collectors.averagingDouble(x->x.getValue())
        ));

    Resultados.toFile(tiemposMedios.entrySet().stream()
        .map(x->TResultMedD.of(x.getKey(),x.getValue()))
        .map(TResultMedD::toString),
        ficheroTiempos,
        true);

}

private static void actualizaTiempos(Map<Problema, Double> tiempos, Problema p, double
d) {
    if (!tiempos.containsKey(p)) {

```

```

        tiempos.put(p, d);
    } else if (tiempos.get(p) > d) {
        tiempos.put(p, d);
    }
}

private static void warmup(BiConsumer<List<Integer>, Integer> f, List<Integer> ls,
Integer umbral) {
    for (int i=0; i<numIterWarmup; i++) {
        f.accept(ls, umbral);
    }
}

record TResultD(Integer tam, Integer numList, Integer numCase, Double t) {
    public static TResultD of(Integer tam, Integer numList, Integer numCase, Double
t){
        return new TResultD(tam, numList, numCase, t);
    }

    public String toString() {
        return String.format("%d,%d,%d,%.0f", tam, numList, numCase, t);
    }
}

record TResultMedD(Integer tam, Double t) {
    public static TResultMedD of(Integer tam, Double t){
        return new TResultMedD(tam, t);
    }

    public String toString() {
        return String.format("%d,%.0f", tam, t);
    }
}

record Problema(Integer tam, Integer numList, Integer numCase) {
    public static Problema of(Integer tam, Integer numList, Integer numCase){
        return new Problema(tam, numList, numCase);
    }
}
}

```

Resultados (consola + gráficas)

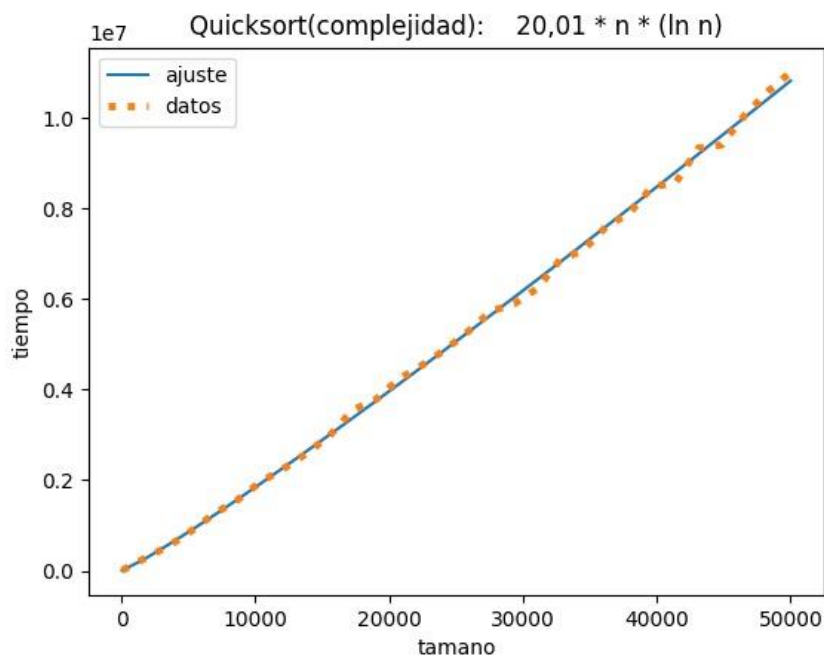
```

a*n*b*(ln n)^c + d
Quicksort(complejidad)
Solutions = a = 20,01
DEBUG - .plot command: ret_22551d4f_6792_4977_ae50_ac445404925d = plt.plot(np.array([50.0, 1772.0, 3494.0, 5217.0, 6939.0, 8662.0, 10384.0, 12106.0, 13829.0, 15551.0, 17274.0, 18996.0, 20718.0, 22441.0, 24163.0, 25886.0
DEBUG - .plot command: ret_0447b158_0871_43e1_9ef5_ccdfad2549a2 = plt.plot(np.array([50.0, 1772.0, 3494.0, 5217.0, 6939.0, 8662.0, 10384.0, 12106.0, 13829.0, 15551.0, 17274.0, 18996.0, 20718.0, 22441.0, 24163.0, 25886.0
DEBUG - .plot command: plt.title("Quicksort(complejidad): 20,01 * n * (ln n)")
DEBUG - .plot command: ret_c9727208_d0c1_4847_8d5b_34f203721261 = plt.legend()
DEBUG - .plot command: ret_c16df795_9724_45eb_9154_cfd7f714746b = plt.xlabel("tamano")
DEBUG - .plot command: ret_2e3b1e1c_e2f2_4d3f_849d_980784ac8d30 = plt.ylabel("tiempo")
DEBUG - Commands... : [C:\Users\Alvaro\AppData\Local\Programs\Python\Python310\python.exe, C:\Users\Alvaro\AppData\Local\Temp\1669062601399-0\exec.py]
WARN -
Quicksort(complejidad)
Solutions = a = 20,01
DEBUG - .plot command: plt.title("Complejidad ordenaci3n")
DEBUG - .plot command: ret_8b79f95f_0a0b_40b1_9fad_51e31d9b9fd5 = plt.plot(np.array([50.0, 1772.0, 3494.0, 5217.0, 6939.0, 8662.0, 10384.0, 12106.0, 13829.0, 15551.0, 17274.0, 18996.0, 20718.0, 22441.0, 24163.0, 25886.0
DEBUG - .plot command: ret_3fb3e84c_3be3_4413_905d_0ffaa34c7309 = plt.legend()
DEBUG - .plot command: ret_5111d27f_a45b_4957_9fee_58ae26f7c4a5 = plt.xlabel("tamano")
DEBUG - .plot command: ret_d358f86b_5871_4960_b86f_1782b88edc5e = plt.ylabel("tiempo")
DEBUG - Commands... : [C:\Users\Alvaro\AppData\Local\Programs\Python\Python310\python.exe, C:\Users\Alvaro\AppData\Local\Temp\1669062604719-0\exec.py]
WARN -
a*n*b*(ln n)^c + d
Quicksort(analisisumbral)
DEBUG - .plot command: plt.title("Análisis del tamaño del umbral")
DEBUG - .plot command: ret_ad556de9_bde2_477d_ae9e_9ce982b17d74 = plt.plot(np.array([1.0, 4.0, 7.0, 11.0, 14.0, 18.0, 21.0, 24.0, 28.0, 31.0, 35.0, 38.0, 41.0, 45.0, 48.0, 52.0, 55.0, 59.0, 62.0, 65.0, 69.0, 72.0, 76.0,
DEBUG - .plot command: ret_6f878fa3_66d5_4a03_96a8_ab5b89dc835f = plt.legend()
DEBUG - .plot command: ret_26f67844_b0fc_4a40_b041_34e3911817a2 = plt.plot(np.array([50.0, 1772.0, 3494.0, 5217.0, 6939.0, 8662.0, 10384.0, 12106.0, 13829.0, 15551.0, 17274.0, 18996.0, 20718.0, 22441.0, 24163.0, 25886.0
DEBUG - .plot command: ret_206ca177_3573_48dd_a992_9789a0684124 = plt.ylabel("tiempo")
DEBUG - Commands... : [C:\Users\Alvaro\AppData\Local\Programs\Python\Python310\python.exe, C:\Users\Alvaro\AppData\Local\Temp\1669062606975-0\exec.py]
WARN -
a*n*b*(ln n)^c + d
Quicksort(analisisumbral)
DEBUG - .plot command: plt.title("Comparativa complejidad Quicksort con distintos umbrales")
DEBUG - .plot command: ret_96db5269_2174_4855_b7d2_a823dd42f3c81 = plt.plot(np.array([50.0, 1772.0, 3494.0, 5217.0, 6939.0, 8662.0, 10384.0, 12106.0, 13829.0, 15551.0, 17274.0, 18996.0, 20718.0, 22441.0, 24163.0, 25886.0
DEBUG - .plot command: ret_1bc688f2_57e0_4288_b229_0704d08fa090 = plt.plot(np.array([50.0, 1772.0, 3494.0, 5217.0, 6939.0, 8662.0, 10384.0, 12106.0, 13829.0, 15551.0, 17274.0, 18996.0, 20718.0, 22441.0, 24163.0, 25886.0
DEBUG - .plot command: ret_e2f67844_b0fc_4a40_b041_34e3911817a2 = plt.plot(np.array([50.0, 1772.0, 3494.0, 5217.0, 6939.0, 8662.0, 10384.0, 12106.0, 13829.0, 15551.0, 17274.0, 18996.0, 20718.0, 22441.0, 24163.0, 25886.0
DEBUG - .plot command: ret_da5e1123_285d_4b49_8b16_39eeef64ecd9 = plt.plot(np.array([50.0, 1772.0, 3494.0, 5217.0, 6939.0, 8662.0, 10384.0, 12106.0, 13829.0, 15551.0, 17274.0, 18996.0, 20718.0, 22441.0, 24163.0, 25886.0
DEBUG - .plot command: ret_d8ee16dc_2af5_4956_9786_220b35c3e35d = plt.legend()
DEBUG - .plot command: ret_5ab2677f_ceb0_4be5_a340_4361c82105ae = plt.xlabel("tamano")
DEBUG - .plot command: ret_d1e17b89_9631_4119_b7f2_4780ad299afa = plt.ylabel("tiempo")
DEBUG - Commands... : [C:\Users\Alvaro\AppData\Local\Programs\Python\Python310\python.exe, C:\Users\Alvaro\AppData\Local\Temp\1669062608864-0\exec.py]
WARN -

```


Figure 1

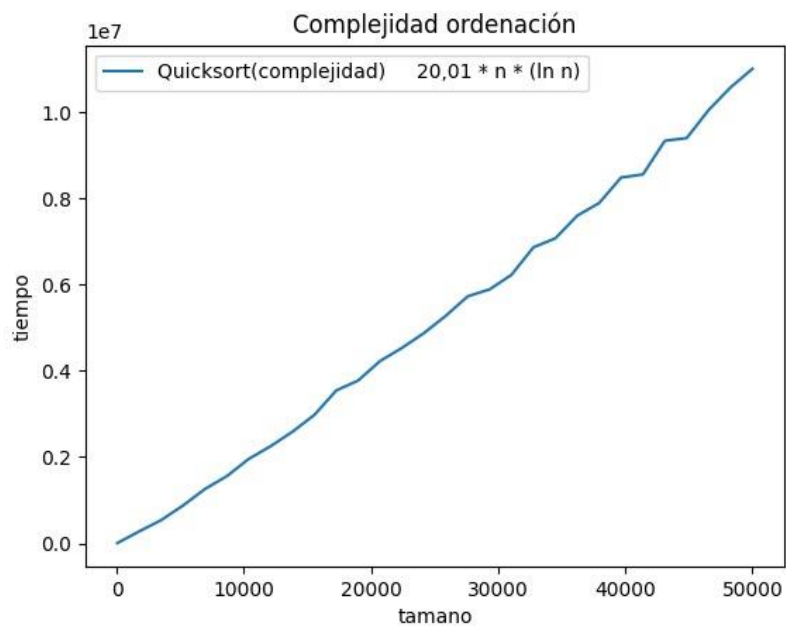
— □ ×



Home navigation icons: back, forward, search, etc.

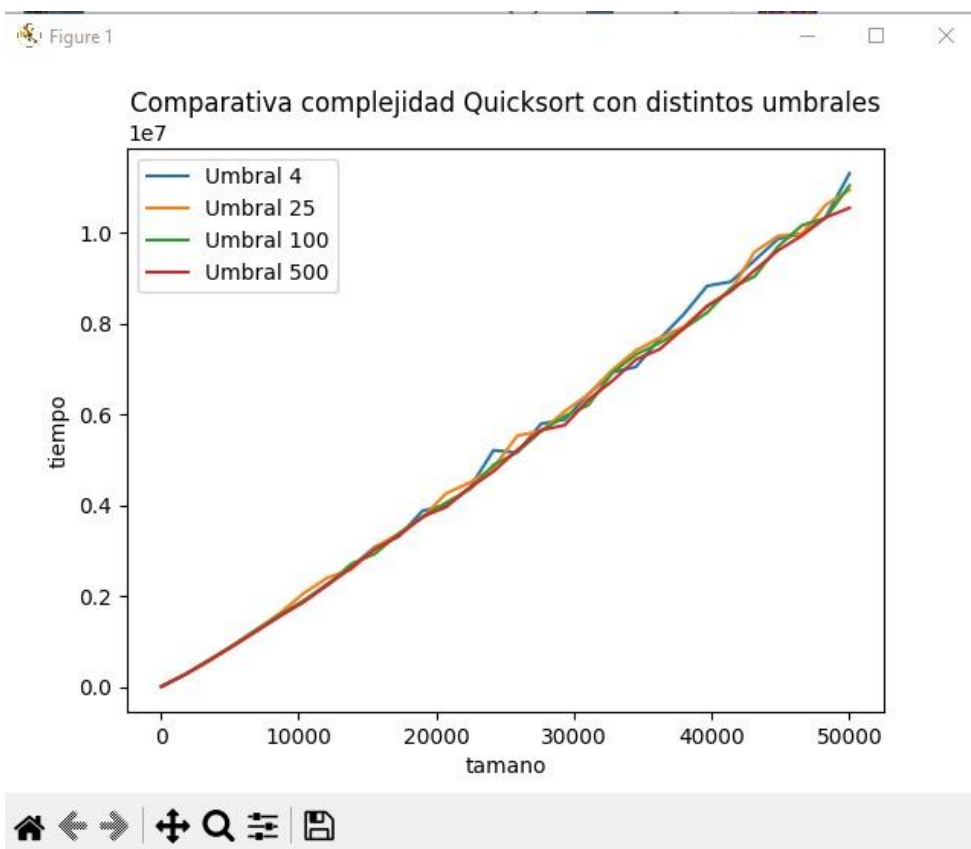
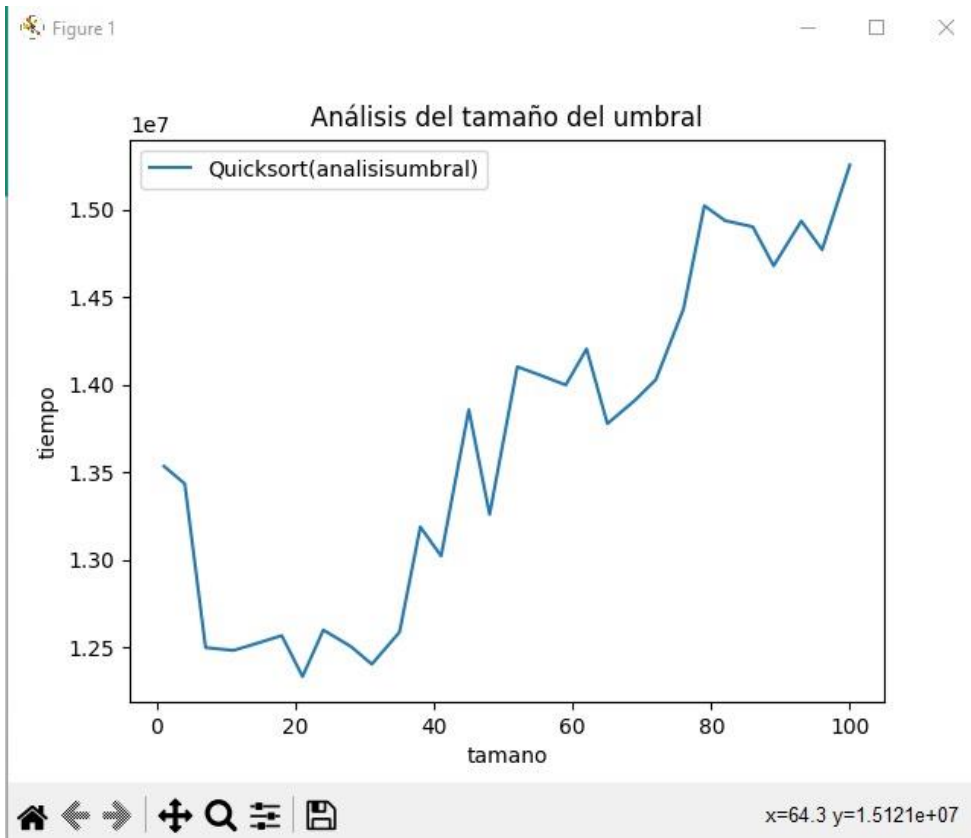
Figure 1

— □ ×



Home navigation icons: back, forward, search, etc.

x=4.19e+04 y=-3.1e+05



Ejercicio3

Ejercicio3.java

```
package ejemplos;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Function;

import us.lsi.tiposrecursivos.BinaryTree;
import us.lsi.tiposrecursivos.BinaryTree.BEmpty;
import us.lsi.tiposrecursivos.BinaryTree.BLeaf;
import us.lsi.tiposrecursivos.BinaryTree.BTree;
import us.lsi.tiposrecursivos.Tree;
import us.lsi.tiposrecursivos.Tree.TEmpty;
import us.lsi.tiposrecursivos.Tree.TLeaf;
import us.lsi.tiposrecursivos.Tree.TNary;

public class Ejercicio3 {

    // Dados un árbol binario de caracteres y un carácter, diseñe un algoritmo que
    // devuelva una lista con todas las cadenas que se forman desde la raíz a una hoja no vacía,
    // excluyendo aquellas cadenas que contengan dicho carácter. Proporcione una solución también
    // para árboles n-arios.

    /* EJERCICIO 3 Binario*/
    public static List<String> ejercicio3BI(BinaryTree<Character> tree, Character character)
    {

        return ejercicio3BI_Aux(tree, character, new ArrayList<>(), "");
    }

    public static List<String> ejercicio3BI_Aux(BinaryTree<Character> tree, Character
    character, List<String> res, String acum) {
        Function<String, Boolean> contiene = elem ->
        {
            return elem.contains(character.toString());
        };
        return switch(tree) {
            case BEmpty<Character> t ->{
                res.remove(acum);
                yield res;
            }

            case BLeaf<Character> t ->{
                Character elemento_actual = t.label();
                acum = acum + elemento_actual;
                if(!contiene.apply(acum)) {
                    res.add(acum);
                    yield res;
                }else {
                    res.remove(acum);
                    yield res;
                }
            }

        }
    }
}
```

```

    case BTree<Character> t -> {
        Character elemento_actual = t.label();
        acum = acum + elemento_actual;
        if(!contiene.apply(acum)) {
            String acum2 = acum;
            ejercicio3BI_Aux(t.left(), caracter, res, acum);
            ejercicio3BI_Aux(t.right(), caracter, res, acum2);
        }
        yield res;
    }
};

/* EJERCICIO 3 Arbol*/

public static List<String> ejercicio3Arbol(Tree<Character> tree, Character caracter) {

    return ejercicio3Arbol_Aux(tree, caracter, new ArrayList<>(), "");
}

public static List<String> ejercicio3Arbol_Aux(Tree<Character> tree, Character caracter,
List<String> res, String acum) {
    Function<String, Boolean> contiene = elem ->
    {
        return elem.contains(caracter.toString());
    };
    return switch(tree) {
        case TEmpty<Character> t ->{
            res.remove(acum);
            yield res;
        }

        case TLeaf<Character> t ->{
            Character elemento_actual = t.label();
            acum = acum + elemento_actual;
            if(!contiene.apply(acum)) {
                res.add(acum);
                yield res;
            }else {
                res.remove(acum);
                yield res;
            }
        }

        case TNary<Character> t -> {
            Character elemento_actual = t.label();
            acum = acum + elemento_actual;
            if(!contiene.apply(acum)) {
                String acum2 = acum;
                t.elements().forEach(tc -> ejercicio3Arbol_Aux(tc, caracter, res,
acum2));
            }
            yield res;
        }
    };
}

}
}

```

TestEjercicio3.java

```

package tests;
import java.util.List;
import java.util.stream.Collectors;

import ejemplos.Ejercicio3;
import us.lsi.common.Files2;
import us.lsi.tiposrecursivos.BinaryTree;
import us.lsi.tiposrecursivos.Tree;

public class TestEjercicio3 {

    public static void main(String[] args) {

        System.out.println("////////////////////////////////////////\n"
            + " \t\tEjercicio3: Arbol Biario\t\t\t \n"
            + "////////////////////////////////////////\n");
        List<String> lineasFicheroBinario =
Files2.streamFromFile("./ficheros/Ejercicio3DatosEntradaBinario.txt")
            .collect(Collectors.toList());
        System.out.println("Datos entrada Binario: " + lineasFicheroBinario);
        for (String linea : lineasFicheroBinario) {
            String[] element = linea.split("#");
            Character caracter = element[1].charAt(0);
            BinaryTree<Character> arbol = BinaryTree.parse(element[0], s ->
s.charAt(0));

            System.out.println("Arbol Binario: " + arbol + " Caracter: " +
caracter);

            System.out.println(Ejercicio3.ejercicio3BI(arbol, caracter));
        }

        System.out.println("////////////////////////////////////////\n"
            + " \t\tEjercicio3: Arbol N-ario\t\t\t \n"
            + "////////////////////////////////////////\n");
        List<String> lineasFicherNario =
Files2.streamFromFile("./ficheros/Ejercicio3DatosEntradaNario.txt")
            .collect(Collectors.toList());
        System.out.println("Datos entrada Nario: " + lineasFicherNario);
        for (String linea : lineasFicherNario) {
            String[] element = linea.split("#");
            Character caracter = element[1].charAt(0);
            Tree<Character> arbol = Tree.parse(element[0], s -> s.charAt(0));
            System.out.println("Arbol Nario: " + arbol + " Caracter: " +
caracter);

            System.out.println(Ejercicio3.ejercicio3Arbol(arbol, caracter));
        }

    }

}

```

Resultados (consola)

```

////////////////////////////////////
Ejercicio3: Arbol Biario
////////////////////////////////////

Datos entrada Binario: [A(B,C)#D, A(B,C)#C, A(B,C)#A, A(B(C,D),E(F,_))#H, A(B(C,D),E(F,_))#D, A(B(C,D(E,F(G,H))),I(J,K))#H, A(B(C,D(E,F(G,H))),I(J,K))#C]
Arbol Binario: A(B,C) Caracter: D
[AB, AC]
Arbol Binario: A(B,C) Caracter: C
[AB]
Arbol Binario: A(B,C) Caracter: A
[]
Arbol Binario: A(B(C,D),E(F,_)) Caracter: H
[ABC, ABD, AEF]
Arbol Binario: A(B(C,D),E(F,_)) Caracter: D
[ABC, AEF]
Arbol Binario: A(B(C,D(E,F(G,H))),I(J,K)) Caracter: H
[ABC, ABDE, ABDFG, AIJ, AIK]
Arbol Binario: A(B(C,D(E,F(G,H))),I(J,K)) Caracter: C
[ABDE, ABDFG, ABDFH, AIJ, AIK]
////////////////////////////////////
Ejercicio3: Arbol N-ario
////////////////////////////////////

Datos entrada Nario: [A(B,C,D)#A, A(B,C,D)#C, A(B,C,D)#D, A(B(C,D,E),F(G,H,I),J(K,L))#F, A(B(C,D,E),F(G,H,I),J(K,L))#K, A(B(C,D(E,F(G,H,I),J),K))#D, A(B(C,D(E,F(G,H,I),J),K))#I]
Arbol Nario: A(B,C,D) Caracter: A
[]
Arbol Nario: A(B,C,D) Caracter: C
[AB, AD]
Arbol Nario: A(B,C,D) Caracter: D
[AB, AC]
Arbol Nario: A(B(C,D,E),F(G,H,I),J(K,L)) Caracter: F
[ABC, ABD, ABE, AJK, AJL]
Arbol Nario: A(B(C,D,E),F(G,H,I),J(K,L)) Caracter: K
[ABC, ABD, ABE, AFG, AFH, AFI, AJL]
Arbol Nario: A(B(C,D(E,F(G,H,I),J),K)) Caracter: D
[ABC, ABK]
Arbol Nario: A(B(C,D(E,F(G,H,I),J),K)) Caracter: I
[ABC, ABDE, ABDFG, ABDFH, ABDJ, ABK]

```

Ejercicio4

Ejercicio4.java

```
package ejemplos;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.function.Function;

import us.lsi.tiposrecursivos.BinaryTree;
import us.lsi.tiposrecursivos.BinaryTree.BEmpty;
import us.lsi.tiposrecursivos.BinaryTree.BLeaf;
import us.lsi.tiposrecursivos.BinaryTree.BTree;
import us.lsi.tiposrecursivos.Tree;
import us.lsi.tiposrecursivos.Tree.TEmpty;
import us.lsi.tiposrecursivos.Tree.TLeaf;
import us.lsi.tiposrecursivos.Tree.TNary;

public class Ejercicio4 {

    // Dado un árbol binario de cadena de caracteres, diseñe un algoritmo que devuelva cierto
    // si se cumple que, para todo nodo, el número total de vocales contenidas en el subárbol
    // izquierdo es igual al del subárbol derecho. Proporcione una solución también para
    // árboles
    // n-arios.

    record Tupla (Boolean cumple, Integer num_voc) {
        public static Tupla of(Boolean cumple, Integer num_voc) {
            return new Tupla(cumple, num_voc);
        }
    }

    /* EJERCICIO 4 Binario*/
    public static Boolean ejercicio4BI(BinaryTree<String> tree) {

        return ejercicio4BI_Aux(tree).cumple();
    }

    public static Tupla ejercicio4BI_Aux(BinaryTree<String> tree) {
        List<Character> vocales = Arrays.asList('a', 'e', 'i', 'o', 'u');
        Function<String, Integer> numVocales = elem ->
        {
            Integer num = 0;

            for (int i = 0; i < elem.length(); i++) {
                if(vocales.contains(elem.charAt(i))) num++;
            }

            return num;
        };
        Tupla res;
        return switch(tree) {
            case BEmpty<String> t -> Tupla.of(true, 0);

            case BLeaf<String> t -> Tupla.of(true, numVocales.apply(t.label()));

            case BTree<String> t -> {
                Tupla izq = ejercicio4BI_Aux (t.left());
                Tupla der = ejercicio4BI_Aux (t.right());
            }
        };
    }
}
```

```

        res = Tupla.of(izq.cumple() && der.cumple() && (izq.num_voc() == der.num_voc()),
            izq.num_voc() + der.num_voc() + numVocales.apply(t.label()));
        yield res;
    }
};

}

/* EJERCICIO 4 Arbol */
public static Boolean ejercicio4Arbol(Tree<String> tree) {
    Tupla res = Tupla.of(true, 0);
    return ejercicio4Arbol_Aux(tree, res).cumple();
}

public static Tupla ejercicio4Arbol_Aux(Tree<String> tree, Tupla res) {
    List<Character> vocales = Arrays.asList('a', 'e', 'i', 'o', 'u');
    Function<String, Integer> numVocales = elem ->
    {
        Integer num = 0;

        for (int i = 0; i < elem.length(); i++) {
            if (vocales.contains(elem.charAt(i))) num++;
        }

        return num;
    };

    return switch (tree) {
        case TEmpty<String> t -> Tupla.of(true, 0);

        case TLeaf<String> t -> Tupla.of(true, numVocales.apply(t.label()));

        case TNary<String> t -> {
            List<Tupla> listaTuplas = new ArrayList<>();
            t.elements().forEach(tc -> {
                listaTuplas.add(ejercicio4Arbol_Aux(tc, res));
            });

            Boolean igual = listaTuplas.stream().distinct().count() <= 1;
            Integer vocalesTotal = listaTuplas.stream().mapToInt(l ->
                l.num_voc()).sum(); // lista de tuplas la transformo a lista de vocales y la sumo

            yield Tupla.of(igual, vocalesTotal);
        };
    };
}
}

```

TestEjercicio4.java

```

package tests;

import java.util.List;
import java.util.stream.Collectors;

import ejemplos.Ejercicio4;
import us.lsi.common.Files2;
import us.lsi.tiposrecursivos.BinaryTree;
import us.lsi.tiposrecursivos.Tree;

public class TestEjercicio4 {
    public static void main(String[] args) {

```



```

System.out.println("////////////////////////////////////////\n"
    + " \t\tEjercicio4: Arbol Biarrio\t\t\t \n"
    + "////////////////////////////////////////\n");
List<String> lineasFicheroBinario =
Files2.streamFromFile("./ficheros/Ejercicio4DatosEntradaBinario.txt")
    .collect(Collectors.toList());
System.out.println("Datos entrada Binario: " + lineasFicheroBinario);
for (String linea : lineasFicheroBinario) {
    BinaryTree<String> arbol = BinaryTree.parse(linea);
    System.out.println("Arbol: " + arbol);
    System.out.println(Ejercicio4.ejercicio4BI(arbol));
}

System.out.println("////////////////////////////////////////\n"
    + " \t\tEjercicio4: Arbol N-ario\t\t\t \n"
    + "////////////////////////////////////////\n");
List<String> lineasFicherNario =
Files2.streamFromFile("./ficheros/Ejercicio4DatosEntradaNario.txt")
    .collect(Collectors.toList());
System.out.println("Datos entrada Nario: " + lineasFicherNario);
for (String linea : lineasFicherNario) {
    Tree<String> arbol = Tree.parse(linea);
    System.out.println("Arbol: " + arbol);
    System.out.println(Ejercicio4.ejercicio4Arbol(arbol));
}
}
}

```

Resultados (consola)

```

////////////////////////////////////////
Ejercicio4: Arbol Biarrio
////////////////////////////////////////
Datos entrada Binario: [pepe(pepa, pepe), pepe(pepa, pep), ada(eda(ola,ale),eda(ele,ale)), ada(eda(ola,ale),eda(ele,al)), cafe(taza(bote,bolsa),perro(gato,leon)), cafe(taza(bote,bolsa),perro(gato,_)), cafe(taza(bote,bolsa),perro(gato,tortuga))]
Arbol: pepe(pepa,pepe)
true
Arbol: pepe(pepa,pep)
false
Arbol: ada(eda(ola,ale),eda(ele,ale))
true
Arbol: ada(eda(ola,ale),eda(ele,al))
false
Arbol: cafe(taza(bote,bolsa),perro(gato,leon))
true
Arbol: cafe(taza(bote,bolsa),perro(gato,_))
false
Arbol: cafe(taza(bote,bolsa),perro(gato,tortuga))
false
////////////////////////////////////////
Ejercicio4: Arbol N-ario
////////////////////////////////////////
Datos entrada Nario: [pepe(pepa, pepe, pepo), pepe(pepa, pepe, pep), ada(eda(ola,ale,elo),eda(ele,ale,alo)), ada(eda(ola,ale,elo),eda(ele,ale,al)), cafe(taza(bote,bolsa,vaso),perro(gato,leon,tigre)), cafe(taza(bote,bolsa,vaso),perro(gato,leon)), cafe(taza(bote,bolsa,vaso),perro(gato,tortuga))]
Arbol: pepe(pepa,pepe,pepo)
true
Arbol: pepe(pepa,pepe,pep)
false
Arbol: ada(eda(ola,ale,elo),eda(ele,ale,alo))
true
Arbol: ada(eda(ola,ale,elo),eda(ele,ale,al))
false
Arbol: cafe(taza(bote,bolsa,vaso),perro(gato,leon,tigre))
true
Arbol: cafe(taza(bote,bolsa,vaso),perro(gato,leon))
false
Arbol: cafe(taza(bote,bolsa,vaso),perro(gato,tortuga))
false

```