Tabla de contenido

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\t\tEjercicio_1\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);
                    }

              }

              Resultados.toFile(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);
```

```
        }
}
```

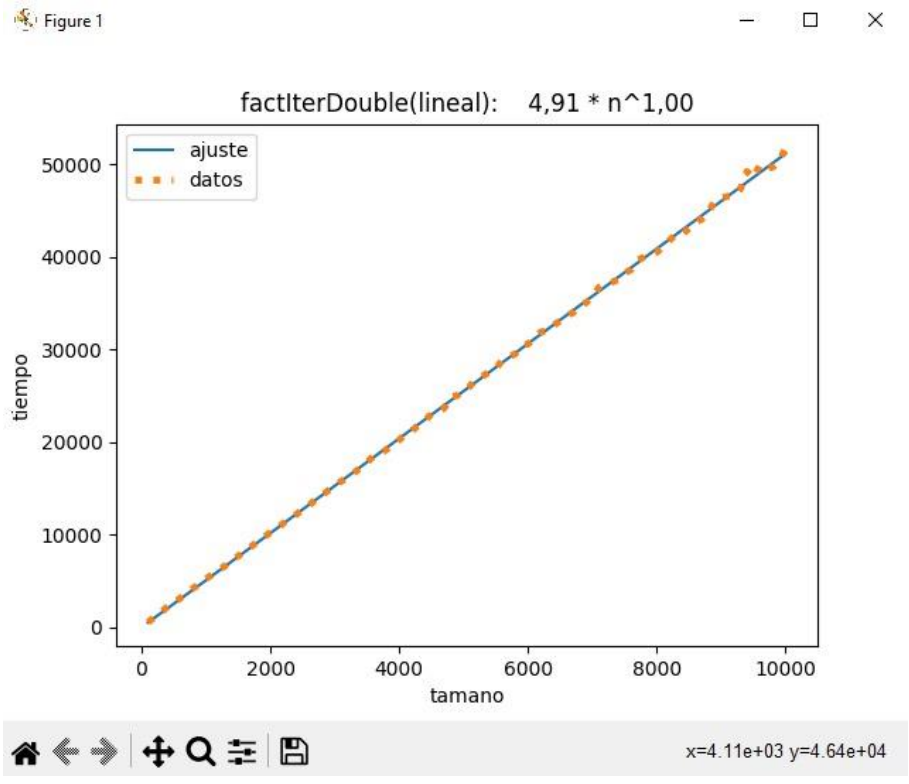
Resultados (consola + gráficas)

```
////////////////////////////////////////////////////
          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 gr@ficas
////////////////////////////////////////////////////

a*n^b*(ln n)^c + d
factIterDouble(lineal)
Solutions = a = 4,91,b = 1,00
DEBUG - .plot command: ret_33443d12_7bfb_41c7_bf25_9f16d9745200 = 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.
DEBUG - .plot command: ret_82e74a85_cbe8_46b4_a399_398eefe29854 = 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.
DEBUG - .plot command: plt.title("factIterDouble(lineal):    4,91 * n^1,00")
DEBUG - .plot command: ret_66c1fad2_376c_4b29_b977_237b7bc416df = plt.legend()
DEBUG - .plot command: ret_ca909625_0c19_40bc_be61_0a06219d2c8e = plt.xlabel("tamano")
DEBUG - .plot command: ret_3ea8d972_7544_4a53_a0e9_c0dcf633ef37 = 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_e26860c080ca = 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.
DEBUG - .plot command: ret_eecda664_5999_4f9d_8ac0_f980d9c599d5 = 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.
DEBUG - .plot command: plt.title("factRecDouble(lineal):    2,92 * n^1,08")
DEBUG - .plot command: ret_dc259c2a_1b23_4e94_82cb_ec247eb5809f = plt.legend()
DEBUG - .plot command: ret_7d66e283_a8c2_4cbe_99d2_bbdaa645eb4a = 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\1669058757052-0\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_0895_453d_867d_163bbebef0d5 = 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,
DEBUG - .plot command: ret_8b878739_ada7_4031_9f13_3cfefeff405e = 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,
DEBUG - .plot command: plt.title("factIterBigInteger(lineal):    0,132 * n^2,08")
DEBUG - .plot command: ret_b7d3c01f_0eb8_4139_a02e_30ca7974cdc1 = plt.legend()
DEBUG - .plot command: ret_cf86d002_1569_4f33_af3b_33ba74ca38c2 = plt.xlabel("tamano")
DEBUG - .plot command: ret_5502ea33_b196_4505_97e7_40ca12377c18 = plt.ylabel("tiempo")
DEBUG - Commands... : [C:\Users\Alvaro\AppData\Local\Programs\Python\Python310\python.exe, C:\Users\Alvaro\AppData\Local\Temp\1669058759056-0\exec.py]
WARN -
factIterBigInteger(lineal)
Solutions = a = 0,09,b = 2,08
factRecBigInteger(lineal)
Solutions = a = 0,13,b = 2,11
DEBUG - .plot command: ret_cf8f1328_fc3f_4009_9edd_993d289f7823 = 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,
DEBUG - .plot command: ret_bb867af5_a070_4639_bed7_640ec92fd0d5 = 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,
DEBUG - .plot command: plt.title("factRecBigInteger(lineal):    0,0087 * n^2,11")
DEBUG - .plot command: ret_9f1b654b_33d4_442e_8180_e69acb3542e9 = plt.legend()
DEBUG - .plot command: ret_1e7b38a4_4117_42c9_871f_a8c3d6e3a121 = plt.xlabel("tamano")
DEBUG - .plot command: ret_4b738906_642a_4a23_8697_9b6f5592db66 = plt.ylabel("tiempo")
DEBUG - Commands... : [C:\Users\Alvaro\AppData\Local\Programs\Python\Python310\python.exe, C:\Users\Alvaro\AppData\Local\Temp\1669058760924-0\exec.py]
WARN -
```



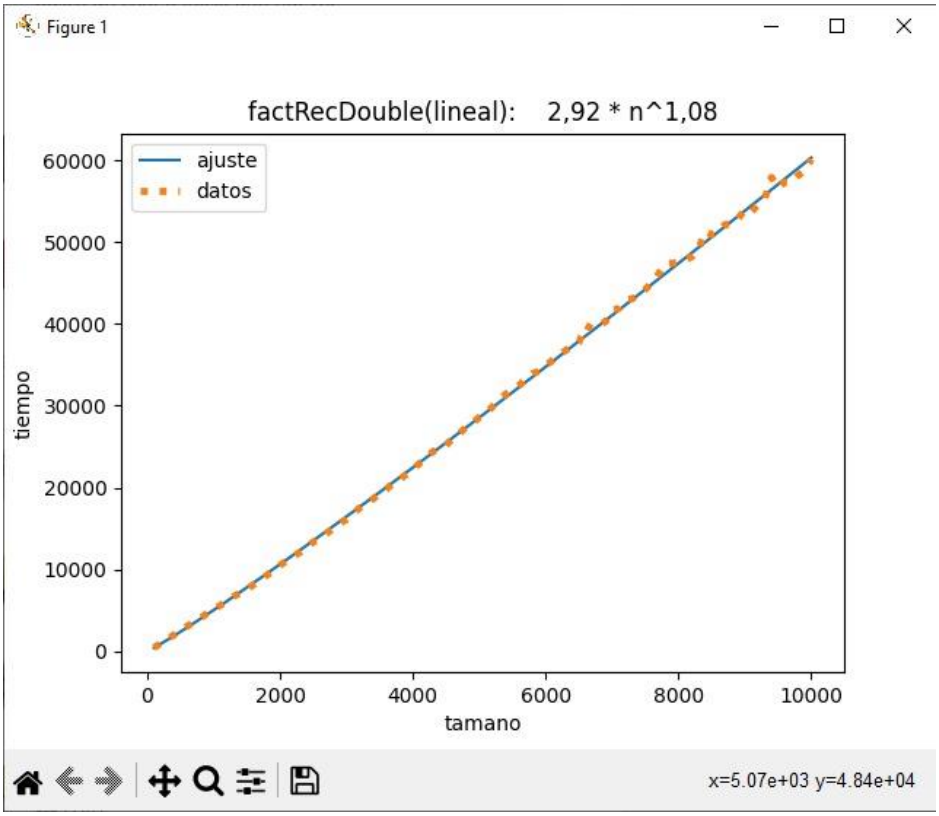


Figure 1
factRecDouble(lineal): 2,92 * n^1,08
ajuste
datos
tiempo
tamano
0
10000
20000
30000
40000
50000
60000
0
2000
4000
6000
8000
10000
x=5.07e+03 y=4.84e+04

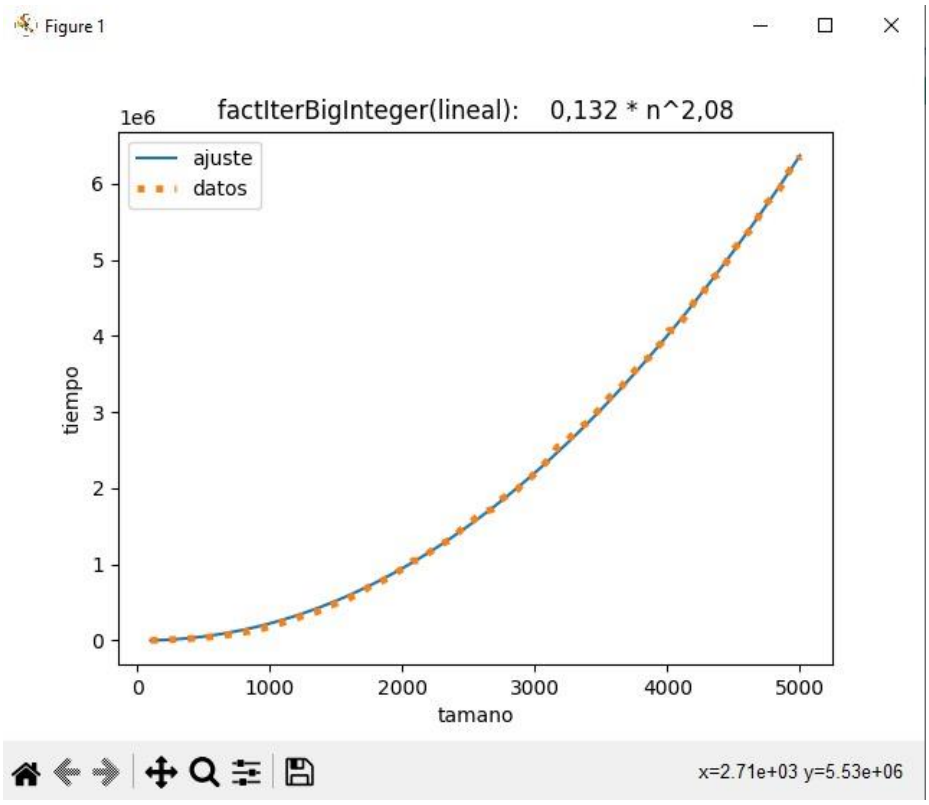


Figure 1
factIterBigInteger(lineal): 0,132 * n^2,08
1e6
ajuste
datos
tiempo
tamano
0
1
2
3
4
5
6
0
1000
2000
3000
4000
5000
x=2.71e+03 y=5.53e+06

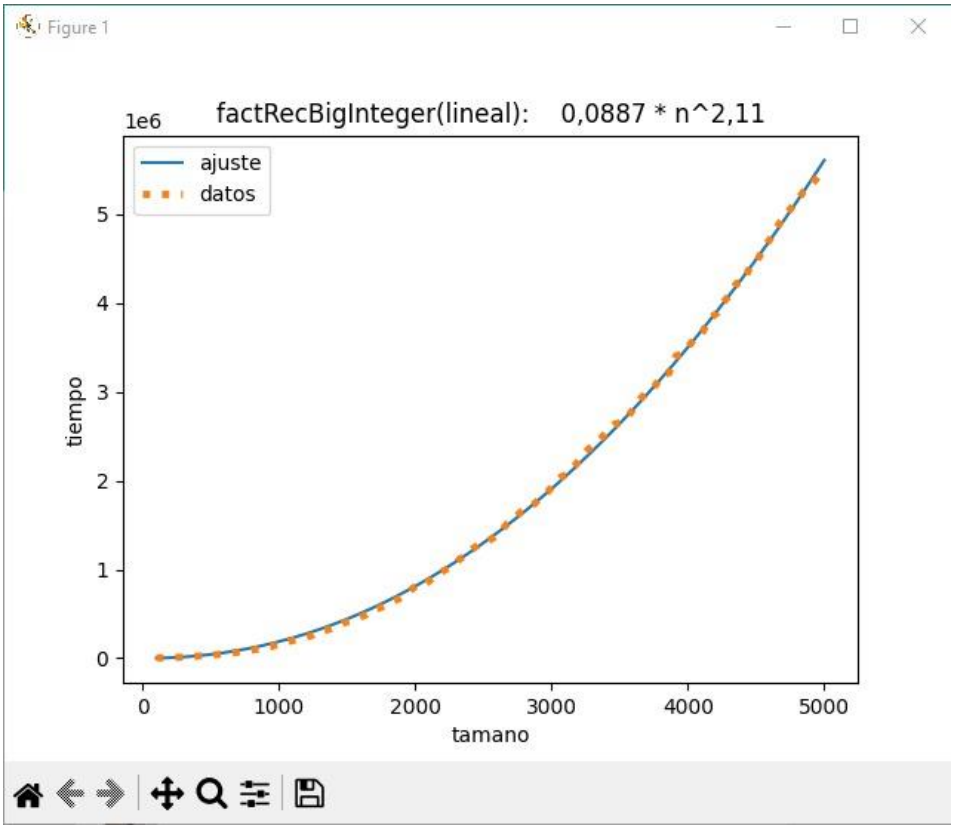


Figure 1
factRecBigInteger(lineal): 0,0887 * n^2,11
1e6
ajuste
datos
tiempo
tamano
0
1000
2000
3000
4000
5000
0
1
2
3
4
5

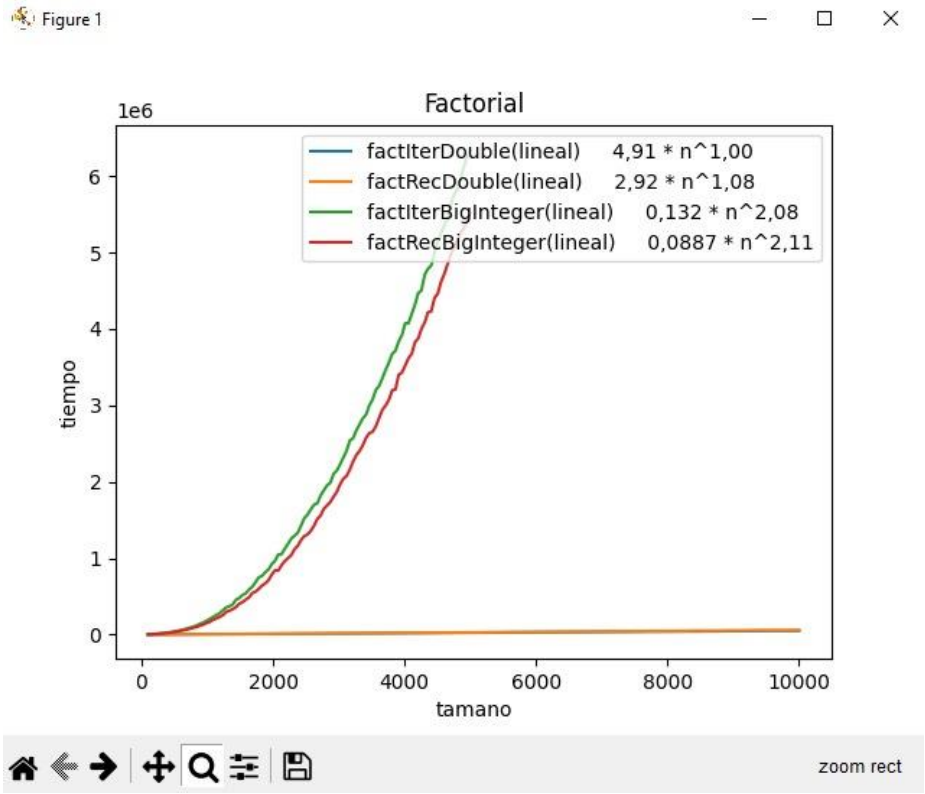


Figure 1
Factorial
1e6
factIterDouble(lineal) 4,91 * n^1,00
factRecDouble(lineal) 2,92 * n^1,08
factIterBigInteger(lineal) 0,132 * n^2,08
factRecBigInteger(lineal) 0,0887 * n^2,11
tiempo
tamano
0
2000
4000
6000
8000
10000
0
1
2
3
4
5
6
zoom rect

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(","));
			Resultados.toFile(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_c9f727b0_d2c1_4847_8d5b_31f203721261 = plt.legend()
DEBUG - .plot command: ret_c16df795_9724_45eb_9154_cfdf7f14746b = plt.xlabel("tamano")
DEBUG - .plot command: ret_2e33e11c_e2f2_4d3f_849d_980704ac8d30 = 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 ordenaci@n")
DEBUG - .plot command: ret_8b79f95f_0a0b_40b1_9fad_51e31d9bfdc5 = 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("An@lisis del tama@o 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_6f678fa3_66d5_4a03_98a0_ab5b89dc855f = plt.legend()
DEBUG - .plot command: ret_3064ce74_2086_4cc8_a19b_facdcd8c7cc0 = plt.xlabel("tamano")
DEBUG - .plot command: ret_206ca177_3573_49dd_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 -
DEBUG - .plot command: plt.title("Comparativa complejidad Quicksort con distintos umbrales")
DEBUG - .plot command: ret_96db5269_2174_4855_b7d2_aa23dd42fc81 = 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_bc29_07bc4d0faa80 = 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_b941_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_da561123_285d_4b49_8b16_39eefe64ecd9 = 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_d1617b89_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 -
```

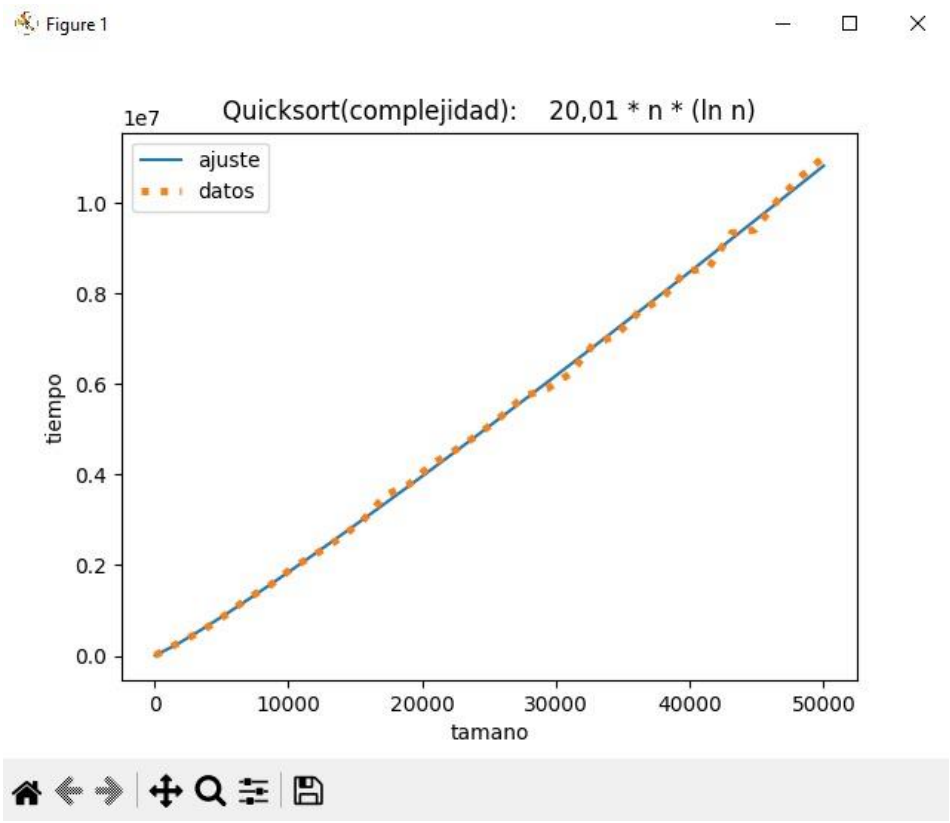


Figure 1
Quicksort(complejidad): 20,01 * n * (ln n)
1e7
ajuste
datos
tiempo
tamano
0.0
0.2
0.4
0.6
0.8
1.0
0
10000
20000
30000
40000
50000

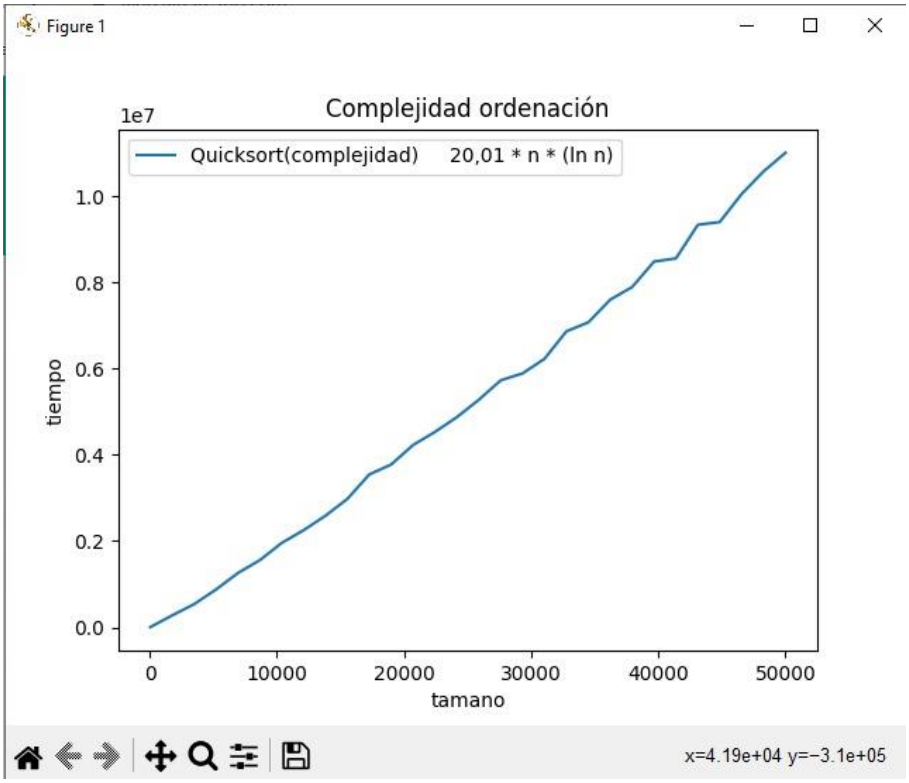


Figure 1
Complejidad ordenación
1e7
Quicksort(complejidad) 20,01 * n * (ln n)
tiempo
tamano
0.0
0.2
0.4
0.6
0.8
1.0
0
10000
20000
30000
40000
50000
x=4.19e+04 y=−3.1e+05

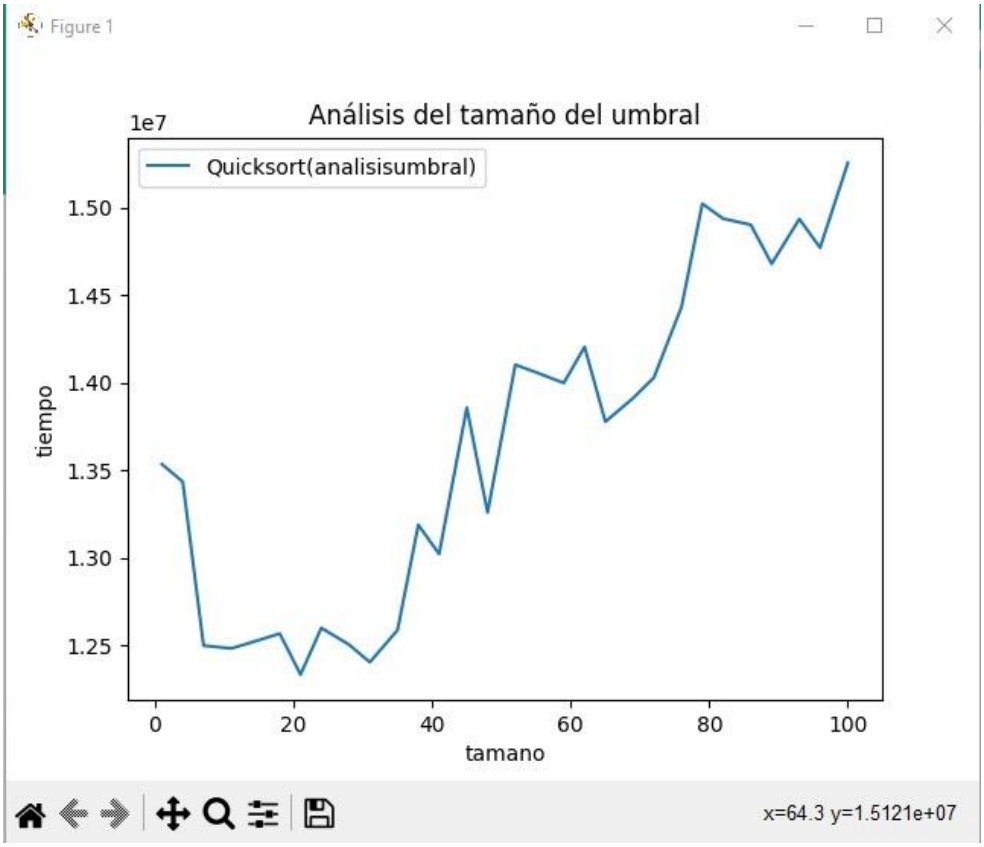


Figure 1
Análisis del tamaño del umbral
1e7
Quicksort(analisisumbral)
1.50
1.45
1.40
1.35
1.30
1.25
tiempo
0
20
40
60
80
100
tamano
x=64.3 y=1.5121e+07

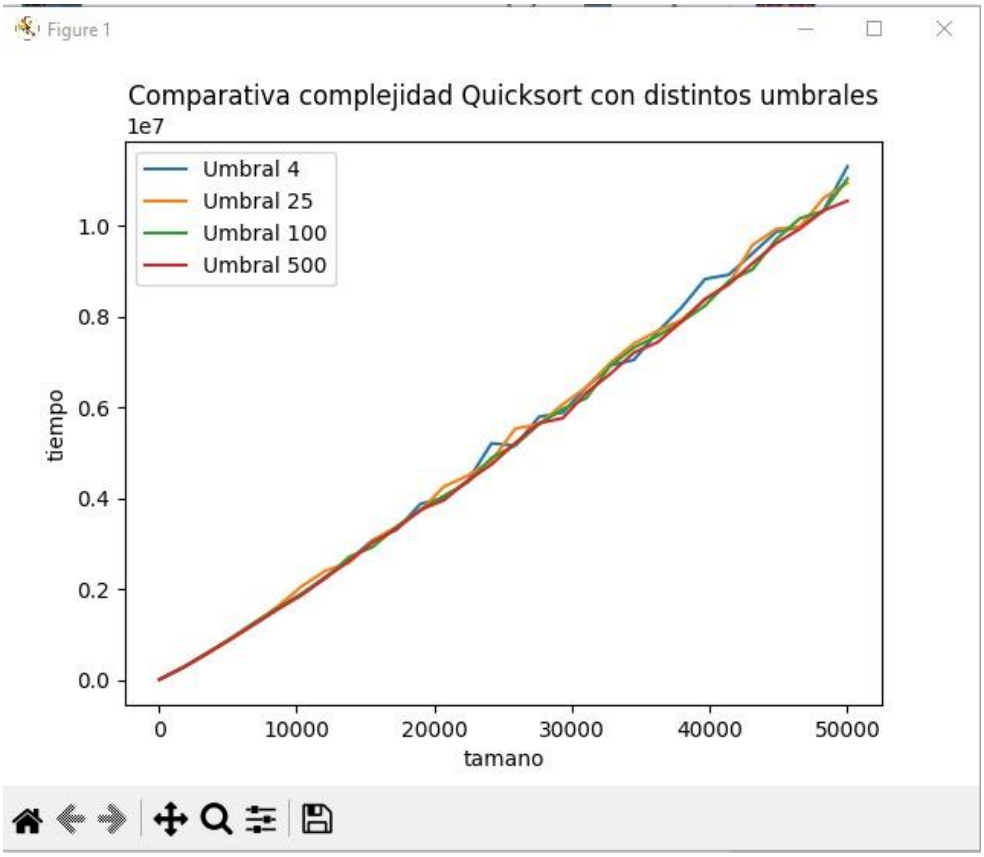


Figure 1
Comparativa complejidad Quicksort con distintos umbrales
1e7
Umbral 4
Umbral 25
Umbral 100
Umbral 500
1.0
0.8
0.6
0.4
0.2
0.0
tiempo
0
10000
20000
30000
40000
50000
tamano

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 caracter)
{

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

	public static List<String> ejercicio3BI_Aux(BinaryTree<Character> tree, Character caracter, List<String> res, String acum) {
		Function<String, Boolean> contiene = elem ->
		{
			return elem.contains(caracter.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 Biario\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 Biario
////////////////////////////////////////////////

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,bols
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,bols
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
```