

# **MEMORIA**

# **PRACTICA INDIVIDUAL 2**

Jaime Linares Barrera

2º Ingeniería Informática - Ingeniería del Software, Grupo 4

# 1. EJERCICIO 1

## 1.1. CÓDIGO EJERCICIO

```
package ejercicios;

import java.math.BigInteger;

public class Ejercicio1 {

    // Factorial: Double - Iterativo
    public static Double fEjercicio1DoubleIter(Integer n) {
        Double res = 1.;
        while(n>0) {
            res = res * n;
            n -= 1;
        }
        return res;
    }

    // Factorial: Double - Recursivo
    public static Double fEjercicio1DoubleRec(Integer n) {
        Double res;
        if(n==0) {
            res = 1.;
        } else {
            res = n * fEjercicio1DoubleIter(n-1);
        }
        return res;
    }

    // Factorial: BigInteger - Iterativo
    public static BigInteger fEjercicio1BigIntegerIter(Integer n) {
        BigInteger res = BigInteger.ONE;
        while(n>0) {
            res = res.multiply(BigInteger.valueOf(n));
            n -= 1;
        }
        return res;
    }

    // Factorial: BigInteger - Recursivo
    public static BigInteger fEjercicio1BigIntegerRec(Integer n) {
        BigInteger res;
        if(n==0) {
            res = BigInteger.ONE;
        } else {
            res = BigInteger.valueOf(n).multiply(fEjercicio1BigIntegerRec(n-1));
        }
        return res;
    }
}
```

## 1.2. CÓDIGO TEST

```
package ejerciciosTests;

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 ejercicios.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("** TEST EJERCICIO 1 **");

        // TEST [correcto funcionamiento de las funciones]
        System.out.println("\n*Analizamos correcto funcionamiento*");
        System.out.println("Factorial de 5(Double, Iterativo): " + Ejercicio1.fEjercicio1DoubleIter(5));
        System.out.println("Factorial de 5(Double, Recursivo): " + Ejercicio1.fEjercicio1DoubleRec(5));
        System.out.println("Factorial de 5(BigInteger, Iterativo): " + Ejercicio1.fEjercicio1BigIntegerIter(5));
        System.out.println("Factorial de 5(BigInteger, Recursivo): " + Ejercicio1.fEjercicio1BigIntegerRec(5));

        // TEST (analizamos tiempo de ejecucion con graficas)
        System.out.println("\n*Analizamos tiempo de ejecucion con graficas*\n");
        generaFicherosTiempoEjecucion();
        muestraGraficas();
    }

    // Parametros para analizar los tiempos de ejecucion
    private static Integer nMin = 25; // n minimo para el calculo del factorial
    private static Integer nMaxIter = 20000; // n maximo para el calculo del factorial en el caso iterativo
    private static Integer nMaxRec = 20000; // n maximo para el calculo del factorial en el caso recursivo
    private static Integer numSizes = 50; // numero de problemas (numero de factoriales distintas a calcular)
    private static Integer numMediciones = 10; // numero de mediciones de tiempo de cada caso (numero de experimentos)
    private static Integer numIter = 50; // numero de iteraciones para cada medicion de tiempo
    private static Integer numIterWarmup = 20000; // numero de iteraciones para warmup

    // Tríos de metodos a probar con su tipo de ajuste y etiqueta para el nombre de los ficheros
    private static List<Trio<Function<Integer, Number>, TipoAjuste, String>> metodosFactorialDouble =
        List.of(
            Trio.of(Ejercicio1::fEjercicio1DoubleRec, TipoAjuste.POWERANB, "FactorialDoubleRecursiva1"),
            Trio.of(Ejercicio1::fEjercicio1DoubleRec, TipoAjuste.POWERANB, "FactorialDoubleRecursiva"),
            Trio.of(Ejercicio1::fEjercicio1DoubleIter, TipoAjuste.POWERANB, "FactorialDoubleIterativa")
        );

    private static List<Trio<Function<Integer, Number>, TipoAjuste, String>> metodosFactorialBigInteger =
        List.of(
            Trio.of(Ejercicio1::fEjercicio1BigIntegerRec, TipoAjuste.POWERANB, "FactorialBigIntegerRecursiva"),
            Trio.of(Ejercicio1::fEjercicio1BigIntegerIter, TipoAjuste.POWERANB, "FactorialBigIntegerIterativa")
        );

    // Generando ficheros con tiempo de ejecucion
    private static <E> void generaFicherosTiempoEjecucionMetodos(List<Trio<Function<E, Number>, TipoAjuste, String>> metodos) {
        for (int i=0; i<metodos.size(); i++) {
            int numMax = i==0 ? nMaxRec : nMaxIter;
            Boolean flagExp = i==0 ? true : false;

            String ficheroSalida = String.format("ficheros/Tiempos%s.csv",
                metodos.get(i).third());

            testTiemposEjecucion(nMin, numMax,
                metodos.get(i).first(),
                ficheroSalida,
                flagExp);
        }
    }

    public static void generaFicherosTiempoEjecucion() {
        generaFicherosTiempoEjecucionMetodos(metodosFactorialDouble);
        generaFicherosTiempoEjecucionMetodos(metodosFactorialBigInteger);
    }
}
```

```

@SuppressWarnings("unchecked")
public static <E> void testTiemposEjecucion(Integer nMin, Integer nMax,
    Function<E, Number> funcionFactorial,
    String ficheroTiempos,
    Boolean flagExp) {
    Map<Problema, Double> tiempos = new HashMap<Problema, Double>();
    Integer nMed = flagExp ? 1 : 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);
            System.out.println(tam);
            warmup(funcionFactorial, 10);
            Integer nIter = flagExp ? numIter/(i+1) : numIter;
            Number[] res = new Number[nIter];
            Long t0 = System.nanoTime();
            for (int z=0; z<nIter; z++) {
                res[z] = funcionFactorial.apply((E) 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 <E> BigInteger warmup(Function<E, Number> factorial, Integer n) {
    BigInteger res = BigInteger.ZERO;
    BigInteger z = BigInteger.ZERO;
    for (int i=0; i<numIterWarmup; i++) {
        if (factorial.apply((E) n).equals(z)) z.add(BigInteger.ONE);
    }
    res = z.equals(BigInteger.ONE)? z.add(BigInteger.ONE):z;
    return res;
}

// Generando graficas
public static <E> void muestraGraficasMetodos(List<Trio<Function<E, Number>, TipoAjuste, String>> metodos,
    List<String> ficherosSalida, List<String> labels) {
    for (int i=0; i<metodos.size(); i++) {
        String ficheroSalida = String.format("ficheros/Tiempos%.csv",
            metodos.get(i).third());
        ficherosSalida.add(ficheroSalida);
        String label = metodos.get(i).third();
        System.out.println(label);

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

        // Obtener ajusteString para mostrarlo en grafica 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));
    }
}

public static void muestraGraficas() {
    List<String> ficherosSalida = new ArrayList<>();
    List<String> labels = new ArrayList<>();

    muestraGraficasMetodos(metodosFactorialDouble, ficherosSalida, labels);
    muestraGraficasMetodos(metodosFactorialBigInteger, ficherosSalida, labels);

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

// Tipos (records) auxiliares
record TResultID(Integer tam, Double t) {
    public static TResultID of(Integer tam, Double t){
        return new TResultID(tam, t);
    }

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

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

```

## 1.3. VOLCADO DE PANTALLA Y GRÁFICAS

```
* TEST EJERCICIO 1 *

*Analizamos correcto funcionamiento*
Factorial de 5(Double, Iterativo): 120.0
Factorial de 5(Double, Recursivo): 120.0
Factorial de 5(BigInteger, Iterativo): 120
Factorial de 5(BigInteger, Recursivo): 120

*Analizamos tiempo de ejecucion con graficas*

FactorialDoubleRecursiva
Solutions = a = 1,52,b = 0,99
DEBUG - .plot command: ret_bdd91182_7a19_425c_a378_25cc5a65f6f8 = plt.plot(np.array([25.0, 432.0, 840.0, 1247.0, 1655.0, 2063.0, 2470.0, 2878.0, 3286.0, 3693.0, 4101.0, 4509.0, 4916.0, 5323.0, 5730.0, 6137.0, 6544.0, 6951.0, 7358.0, 7765.0, 8172.0, 8579.0, 8986.0, 9393.0, 9800.0, 10207.0, 10614.0, 11021.0, 11428.0, 11835.0, 12242.0, 12649.0, 13056.0, 13463.0, 13870.0, 14277.0, 14684.0, 15091.0, 15498.0, 15905.0, 16312.0, 16719.0, 17126.0, 17533.0, 17940.0, 18347.0, 18754.0, 19161.0, 19568.0, 19975.0, 20382.0, 20789.0, 21196.0, 21603.0, 22010.0, 22417.0, 22824.0, 23231.0, 23638.0, 24045.0, 24452.0, 24859.0, 25266.0, 25673.0, 26080.0, 26487.0, 26894.0, 27301.0, 27708.0, 28115.0, 28522.0, 28929.0, 29336.0, 29743.0, 30150.0, 30557.0, 30964.0, 31371.0, 31778.0, 32185.0, 32592.0, 32999.0, 33406.0, 33813.0, 34220.0, 34627.0, 35034.0, 35441.0, 35848.0, 36255.0, 36662.0, 37069.0, 37476.0, 37883.0, 38290.0, 38697.0, 39104.0, 39511.0, 39918.0, 40325.0, 40732.0, 41139.0, 41546.0, 41953.0, 42360.0, 42767.0, 43174.0, 43581.0, 43988.0, 44395.0, 44802.0, 45209.0, 45616.0, 46023.0, 46430.0, 46837.0, 47244.0, 47651.0, 48058.0, 48465.0, 48872.0, 49279.0, 49686.0, 50093.0, 50499.0, 50906.0, 51313.0, 51720.0, 52127.0, 52534.0, 52941.0, 53348.0, 53755.0, 54162.0, 54569.0, 54976.0, 55383.0, 55790.0, 56197.0, 56604.0, 57011.0, 57418.0, 57825.0, 58232.0, 58639.0, 59046.0, 59453.0, 59860.0, 60267.0, 60674.0, 61081.0, 61488.0, 61895.0, 62302.0, 62709.0, 63116.0, 63523.0, 63930.0, 64337.0, 64744.0, 65151.0, 65558.0, 65965.0, 66372.0, 66779.0, 67186.0, 67593.0, 67999.0, 68406.0, 68813.0, 69220.0, 69627.0, 70034.0, 70441.0, 70848.0, 71255.0, 71662.0, 72069.0, 72476.0, 72883.0, 73290.0, 73697.0, 74104.0, 74511.0, 74918.0, 75325.0, 75732.0, 76139.0, 76546.0, 76953.0, 77360.0, 77767.0, 78174.0, 78581.0, 78988.0, 79395.0, 79802.0, 80209.0, 80616.0, 81023.0, 81430.0, 81837.0, 82244.0, 82651.0, 83058.0, 83465.0, 83872.0, 84279.0, 84686.0, 85093.0, 85499.0, 85906.0, 86313.0, 86720.0, 87127.0, 87534.0, 87941.0, 88348.0, 88755.0, 89162.0, 89569.0, 89976.0, 90383.0, 90790.0, 91197.0, 91604.0, 92011.0, 92418.0, 92825.0, 93232.0, 93639.0, 94046.0, 94453.0, 94860.0, 95267.0, 95674.0, 96081.0, 96488.0, 96895.0, 97302.0, 97709.0, 98116.0, 98523.0, 98930.0, 99337.0, 99744.0, 100151.0, 100558.0, 100965.0, 101372.0, 101779.0, 102186.0, 102593.0, 102999.0, 103406.0, 103813.0, 104220.0, 104627.0, 105034.0, 105441.0, 105848.0, 106255.0, 106662.0, 107069.0, 107476.0, 107883.0, 108290.0, 108697.0, 109104.0, 109511.0, 109918.0, 110325.0, 110732.0, 111139.0, 111546.0, 111953.0, 112360.0, 112767.0, 113174.0, 113581.0, 113988.0, 114395.0, 114802.0, 115209.0, 115616.0, 116023.0, 116430.0, 116837.0, 117244.0, 117651.0, 118058.0, 118465.0, 118872.0, 119279.0, 119686.0, 120093.0, 120499.0, 120906.0, 121313.0, 121720.0, 122127.0, 122534.0, 122941.0, 123348.0, 123755.0, 124162.0, 124569.0, 124976.0, 125383.0, 125790.0, 126197.0, 126604.0, 127011.0, 127418.0, 127825.0, 128232.0, 128639.0, 129046.0, 129453.0, 129860.0, 130267.0, 130674.0, 131081.0, 131488.0, 131895.0, 132302.0, 132709.0, 133116.0, 133523.0, 133930.0, 134337.0, 134744.0, 135151.0, 135558.0, 135965.0, 136372.0, 136779.0, 137186.0, 137593.0, 137999.0, 138406.0, 138813.0, 139220.0, 139627.0, 140034.0, 140441.0, 140848.0, 141255.0, 141662.0, 142069.0, 142476.0, 142883.0, 143290.0, 143697.0, 144104.0, 144511.0, 144918.0, 145325.0, 145732.0, 146139.0, 146546.0, 146953.0, 147360.0, 147767.0, 148174.0, 148581.0, 148988.0, 149395.0, 149802.0, 150209.0, 150616.0, 151023.0, 151430.0, 151837.0, 152244.0, 152651.0, 153058.0, 153465.0, 153872.0, 154279.0, 154686.0, 155093.0, 155499.0, 155906.0, 156313.0, 156720.0, 157127.0, 157534.0, 157941.0, 158348.0, 158755.0, 159162.0, 159569.0, 159976.0, 160383.0, 160790.0, 161197.0, 161604.0, 162011.0, 162418.0, 162825.0, 163232.0, 163639.0, 164046.0, 164453.0, 164860.0, 165267.0, 165674.0, 166081.0, 166488.0, 166895.0, 167302.0, 167709.0, 168116.0, 168523.0, 168930.0, 169337.0, 169744.0, 170151.0, 170558.0, 170965.0, 171372.0, 171779.0, 172186.0, 172593.0, 172999.0, 173406.0, 173813.0, 174220.0, 174627.0, 175034.0, 175441.0, 175848.0, 176255.0, 176662.0, 177069.0, 177476.0, 177883.0, 178290.0, 178697.0, 179104.0, 179511.0, 179918.0, 180325.0, 180732.0, 181139.0, 181546.0, 181953.0, 182360.0, 182767.0, 183174.0, 183581.0, 183988.0, 184395.0, 184802.0, 185209.0, 185616.0, 186023.0, 186430.0, 186837.0, 187244.0, 187651.0, 188058.0, 188465.0, 188872.0, 189279.0, 189686.0, 190093.0, 190499.0, 190906.0, 191313.0, 191720.0, 192127.0, 192534.0, 192941.0, 193348.0, 193755.0, 194162.0, 194569.0, 194976.0, 195383.0, 195790.0, 196197.0, 196604.0, 197011.0, 197418.0, 197825.0, 198232.0, 198639.0, 199046.0, 199453.0, 199860.0, 200267.0, 200674.0, 201081.0, 201488.0, 201895.0, 202302.0, 202709.0, 203116.0, 203523.0, 203930.0, 204337.0, 204744.0, 205151.0, 205558.0, 205965.0, 206372.0, 206779.0, 207186.0, 207593.0, 207999.0, 208406.0, 208813.0, 209220.0, 209627.0, 210034.0, 210441.0, 210848.0, 211255.0, 211662.0, 212069.0, 212476.0, 212883.0, 213290.0, 213697.0, 214104.0, 214511.0, 214918.0, 215325.0, 215732.0, 216139.0, 216546.0, 216953.0, 217360.0, 217767.0, 218174.0, 218581.0, 218988.0, 219395.0, 219802.0, 220209.0, 220616.0, 221023.0, 221430.0, 221837.0, 222244.0, 222651.0, 223058.0, 223465.0, 223872.0, 224279.0, 224686.0, 225093.0, 225499.0, 225906.0, 226313.0, 226720.0, 227127.0, 227534.0, 227941.0, 228348.0, 228755.0, 229162.0, 229569.0, 229976.0, 230383.0, 230790.0, 231197.0, 231604.0, 232011.0, 232418.0, 232825.0, 233232.0, 233639.0, 234046.0, 234453.0, 234860.0, 235267.0, 235674.0, 236081.0, 236488.0, 236895.0, 237302.0, 237709.0, 238116.0, 238523.0, 238930.0, 239337.0, 239744.0, 240151.0, 240558.0, 240965.0, 241372.0, 241779.0, 242186.0, 242593.0, 242999.0, 243406.0, 243813.0, 244220.0, 244627.0, 245034.0, 245441.0, 245848.0, 246255.0, 246662.0, 247069.0, 247476.0, 247883.0, 248290.0, 248697.0, 249104.0, 249511.0, 249918.0, 250325.0, 250732.0, 251139.0, 251546.0, 251953.0, 252360.0, 252767.0, 253174.0, 253581.0, 253988.0, 254395.0, 254802.0, 255209.0, 255616.0, 256023.0, 256430.0, 256837.0, 257244.0, 257651.0, 258058.0, 258465.0, 258872.0, 259279.0, 259686.0, 260093.0, 260499.0, 260906.0, 261313.0, 261720.0, 262127.0, 262534.0, 262941.0, 263348.0, 263755.0, 264162.0, 264569.0, 264976.0, 265383.0, 265790.0, 266197.0, 266604.0, 267011.0, 267418.0, 267825.0, 268232.0, 268639.0, 269046.0, 269453.0, 269860.0, 270267.0, 270674.0, 271081.0, 271488.0, 271895.0, 272302.0, 272709.0, 273116.0, 273523.0, 273930.0, 274337.0, 274744.0, 275151.0, 275558.0, 275965.0, 276372.0, 276779.0, 277186.0, 277593.0, 277999.0, 278406.0, 278813.0, 279220.0, 279627.0, 280034.0, 280441.0, 280848.0, 281255.0, 281662.0, 282069.0, 282476.0, 282883.0, 283290.0, 283697.0, 284104.0, 284511.0, 284918.0, 285325.0, 285732.0, 286139.0, 286546.0, 286953.0, 287360.0, 287767.0, 288174.0, 288581.0, 288988.0, 289395.0, 289802.0, 290209.0, 290616.0, 291023.0, 291430.0, 291837.0, 292244.0, 292651.0, 293058.0, 293465.0, 293872.0, 294279.0, 294686.0, 295093.0, 295499.0, 295906.0, 296313.0, 296720.0, 297127.0, 297534.0, 297941.0, 298348.0, 298755.0, 299162.0, 299569.0, 299976.0, 300383.0, 300790.0, 301197.0, 301604.0, 302011.0, 302418.0, 302825.0, 303232.0, 303639.0, 304046.0, 304453.0, 304860.0, 305267.0, 305674.0, 306081.0, 306488.0, 306895.0, 307302.0, 307709.0, 308116.0, 308523.0, 308930.0, 309337.0, 309744.0, 310151.0, 310558.0, 310965.0, 311372.0, 311779.0, 312186.0, 312593.0, 312999.0, 313406.0, 313813.0, 314220.0, 314627.0, 315034.0, 315441.0, 315848.0, 316255.0, 316662.0, 317069.0, 317476.0, 317883.0, 318290.0, 318697.0, 319104.0, 319511.0, 319918.0, 320325.0, 320732.0, 321139.0, 321546.0, 321953.0, 322360.0, 322767.0, 323174.0, 323581.0, 323988.0, 324395.0, 324802.0, 325209.0, 325616.0, 326023.0, 326430.0, 326837.0, 327244.0, 327651.0, 328058.0, 328465.0, 328872.0, 329279.0, 329686.0, 330093.0, 330499.0, 330906.0, 331313.0, 331720.0, 332127.0, 332534.0, 332941.0, 333348.0, 333755.0, 334162.0, 334569.0, 334976.0, 335383.0, 335790.0, 336197.0, 336604.0, 337011.0, 337418.0, 337825.0, 338232.0, 338639.0, 339046.0, 339453.0, 339860.0, 340267.0, 340674.0, 341081.0, 341488.0, 341895.0, 342302.0, 342709.0, 343116.0, 343523.0, 343930.0, 344337.0, 344744.0, 345151.0, 345558.0, 345965.0, 346372.0, 346779.0, 347186.0, 347593.0, 347999.0, 348406.0, 348813.0, 349220.0, 349627.0, 350034.0, 350441.0, 350848.0, 351255.0, 351662.0, 352069.0, 352476.0, 352883.0, 353290.0, 353697.0, 354104.0, 354511.0, 354918.0, 355325.0, 355732.0, 356139.0, 356546.0, 356953.0, 357360.0, 357767.0, 358174.0, 358581.0, 358988.0, 359395.0, 359802.0, 360209.0, 360616.0, 361023.0, 361430.0, 361837.0, 362244.0, 362651.0, 363058.0, 363465.0, 363872.0, 364279.0, 364686.0, 365093.0, 365499.0, 365906.0, 366313.0, 366720.0, 367127.0, 367534.0, 367941.0, 368348.0, 368755.0, 369162.0, 369569.0, 369976.0, 370383.0, 370790.0, 371197.0, 371604.0, 372011.0, 372418.0, 372825.0, 373232.0, 373639.0, 374046.0, 374453.0, 374860.0, 375267.0, 375674.0, 376081.0, 376488.0, 376895.0, 377302.0, 377709.0, 378116.0, 378523.0, 378930.0, 379337.0, 379744.0, 380151.0, 380558.0, 380965.0, 381372.0, 381779.0, 382186.0, 382593.0, 382999.0, 383406.0, 383813.0, 384220.0, 384627.0, 385034.0, 385441.0, 385848.0, 386255.0, 386662.0, 387069.0, 387476.0, 387883.0, 388290.0, 388697.0, 389104.0, 389511.0, 389918.0, 390325.0, 390732.0, 391139.0, 391546.0, 391953.0, 392360.0, 392767.0, 393174.0, 393581.0, 393988.0, 394395.0, 394802.0, 395209.0, 395616.0, 396023.0, 396430.0, 396837.0, 397244.0, 397651.0, 398058.0, 398465.0, 398872.0, 399279.0, 399686.0, 400093.0, 400499.0, 400906.0, 401313.0, 401720.0, 402127.0, 402534.0, 402941.0, 403348.0, 403755.0, 404162.0, 404569.0, 404976.0, 405383.0, 405790.0, 406197.0, 406604.0, 407011.0, 407418.0, 407825.0, 408232.0, 408639.0, 409046.0, 409453.0, 409860.0, 410267.0, 410674.0, 411081.0, 411488.0, 411895.0, 412302.0, 412709.0, 413116.0, 413523.0, 413930.0, 414337.0, 414744.0, 415151.0, 415558.0, 415965.0, 416372.0, 416779.0, 417186.0, 417593.0, 417999.0, 418406.0, 418813.0, 419220.0, 419627.0, 420034.0, 420441.0, 420848.0, 421255.0, 421662.0, 422069.0, 422476.0, 422883.0, 423290.0, 423697.0, 424104.0, 424511.0, 424918.0, 425325.0, 425732.0, 426139.0, 426546.0, 426953.0, 427360.0, 427767.0, 428174.0, 428581.0, 428988.0, 429395.0, 429802.0, 430209.0, 430616.0, 431023.0, 431430.0, 431837.0, 432244.0, 432651.0, 433058.0, 433465.0, 433872.0, 434279.0, 434686.0, 435093.0, 435499.0, 435906.0, 436313.0, 436720.0, 437127.0, 437534.0, 437941.0, 438348.0, 438755.0, 439162.0, 439569.0, 439976.0, 440383.0, 440790.0, 441197.0, 441604.0, 442011.0, 442418.0, 442825.0, 443232.0, 443639.0, 444046.0, 444453.0, 444860.0, 445267.0, 445674.0, 446081.0, 446488.0, 446895.0, 447302.0, 447709.0, 448116.0, 448523.0, 448930.0, 449337.0, 449744.0, 450151.0, 450558.0, 450965.0, 451372.0, 451779.0, 452186.0, 452593.0, 452999.0, 453406.0, 453813.0, 454220.0, 454627.0, 455034.0, 455441.0, 455848.0, 456255.0, 456662.0, 457069.0, 457476.0, 457883.0, 458290.0, 458697.0, 459104.0, 459511.0, 459918.0, 460325.0, 460732.0, 461139.0, 461546.0, 461953.0, 462360.0, 462767.0, 463174.0, 463581.0, 463988.0, 464395.0, 464802.0, 465209.0, 465616.0, 466023.0, 466430.0, 466837.0, 467244.0, 467651.0, 468058.0, 468465.0, 468872.0, 469279.0, 469686.0, 470093.0, 470499.0, 470906.0, 471313.0, 471720.0, 472127.0, 472534.0, 472941.0, 473348.0, 473755.0, 474162.0, 474569.0, 474976.0, 475383.0, 475790.0, 476197.0, 476604.0, 477011.0, 477418.0, 477825.0, 478232.0, 478639.0, 479046.0, 479453.0, 479860.0, 480267.0, 480674.0, 481081.0, 481488.0, 481895.0, 482302.0, 482709.0, 483116.0, 483523.0, 483930.0, 484337.0, 484744.0, 485151.0, 485558.0, 485965.0, 486372.0, 486779.0, 487186.0, 487593.0, 487999.0, 488406.0, 488813.0, 489220.0, 489627.0, 490034.0, 490441.0, 490848.0, 491255.0, 491662.0, 492069.0, 492476.0, 492883.0, 493290.0, 493697.0, 494104.0, 494511.0, 494918.0, 495325.0, 495732.0, 496139.0, 496546.0, 496953.0, 497360.0, 497767.0, 498174.0, 498581.0, 498988.0, 499395.0, 499802.0, 500209.0, 500616.0, 501023.0, 501430.0, 501837.0, 502244.0, 502651.0, 503058.0, 503465.0, 503872.0, 504279.0, 504686.0, 5050
```



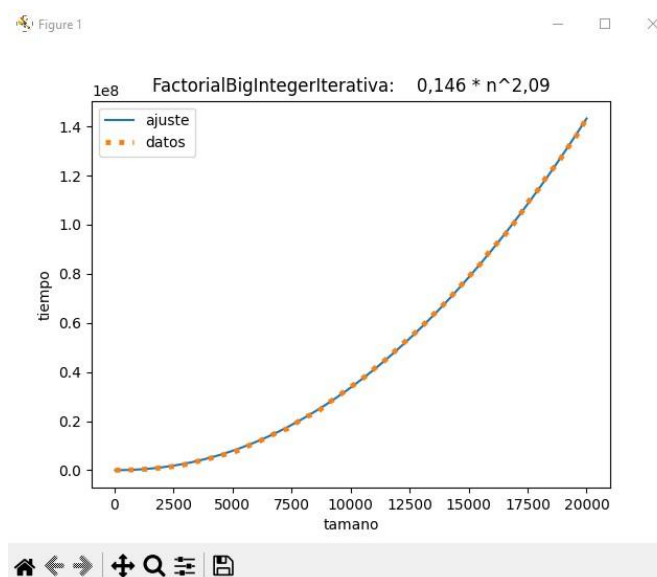
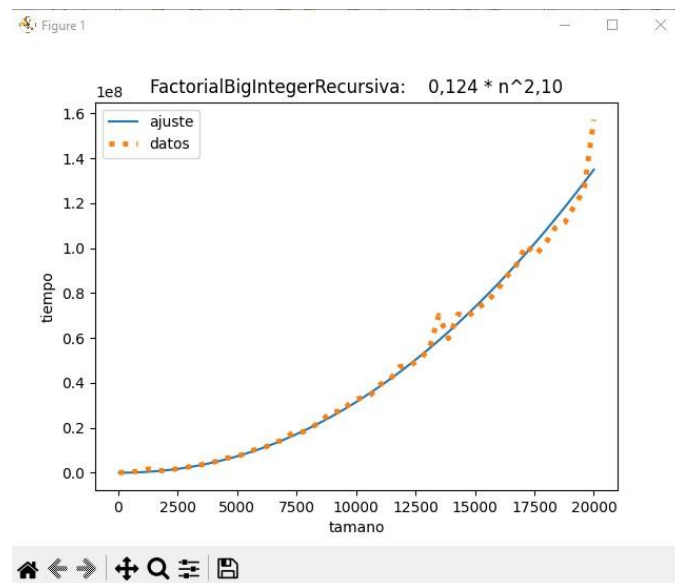
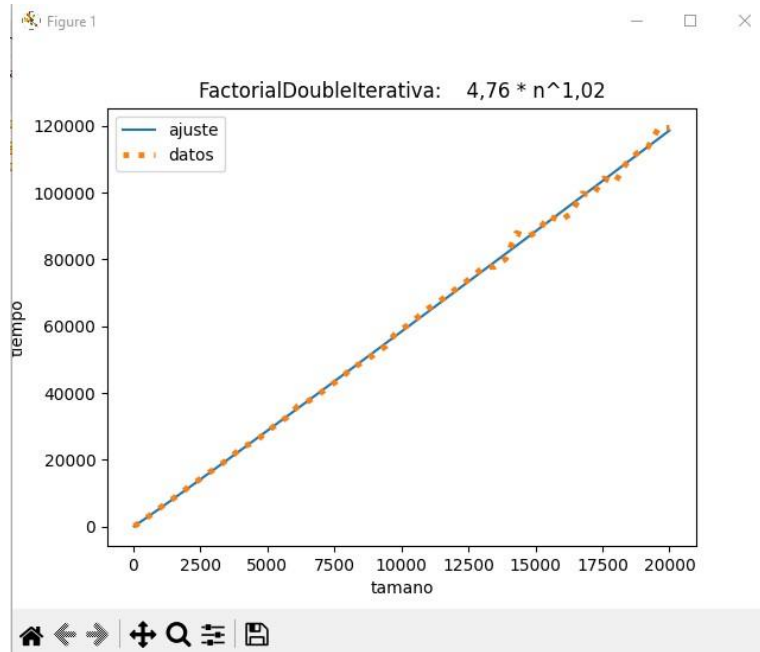
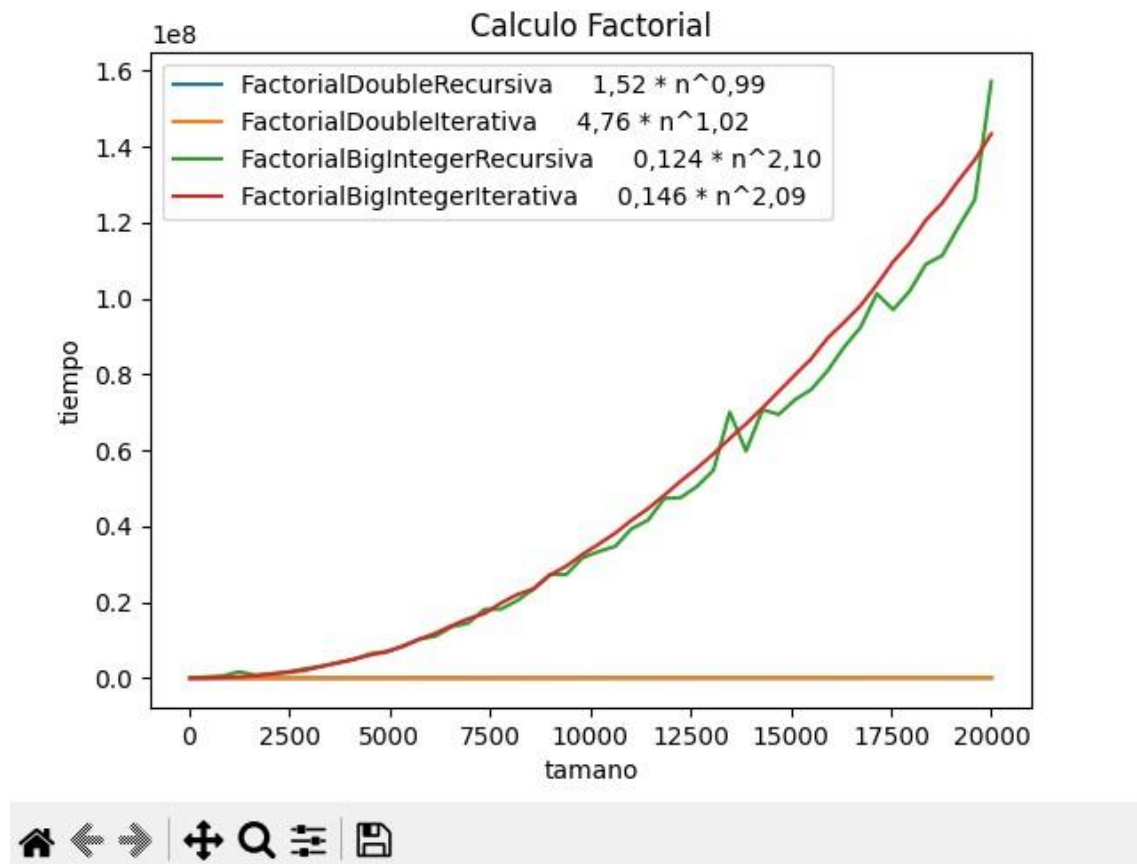


Figure 1



## 2. EJERCICIO 2

### 2.1. CÓDIGO EJERCICIO

```
package ejercicios;

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);
        }
    }
}
```

```

private 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;
}

private 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);
}
}

```

---

## 2.2. CÓDIGO TEST

```

package ejerciciosTests;

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 ejercicios.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 {

    public static void main(String[] args) {
        System.out.println("* TEST EJERCICIO 1 *");

        // TEST (correcto funcionamiento de las funciones)
        System.out.println("\n*Analizamos correcto funcionamiento*");
        List<Integer> ls1 = List2.of(23,5,56,67,57,32,21,8,24,11,1,44,89,2);
        List<Integer> ls2 = List2.ofCollection(ls1);
        System.out.println("- Lista original: " + ls1);
        Ejercicio2.sort(ls1, 4);
        System.out.println("- Lista ordenada con umbral pequeño (4): " + ls1);
        Ejercicio2.sort(ls2, 25);
        System.out.println("- Lista ordenada con umbral grande (20): " + ls2);
    }
}

```



```

// TEST (analizamos tiempo de ejecucion con graficas)
System.out.println("\n*Analizamos tiempo de ejecucion con graficas*\n");
generaFicheroListasEnteros("ficheros/2Lista1AlgoritmoSort.txt");
generaFicherosTiempoEjecucion();
muestraGraficas();
}

// Parametros de generacion de las listas
private static Integer sizeMin = 50; // tamaño mínimo de lista
private static Integer sizeMax = 20000; // tamaño máximo de lista
private static Integer numListas = 40; // número de de listas
// para inicializarlo una sola vez y compartirlo con los metodos que lo requieran
private static Random rr = new Random(System.nanoTime());

// Parametros de medicion
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 medicion de tiempo
private static Integer numIterWarmup = 100; // número de iteraciones para warmup

// Umbral fijo para analisis de complejidad
private static Integer umbralFijo = 4;

// Lista umbrales que vamos a tomar
private static List<Integer> umbrales = List2.of(4, 25, 100, 500);

// Fichero entrada
private static String ficheroListaEntrada = "ficheros/2Lista1AlgoritmoSort.txt";

// Para el analisis de la complejidad con umbral fijo y listas de distinto tamaño
private static List<Trio<BiConsumer<List<Integer>, Integer>, TipoAjuste, String>> metodosComplejidadUmbralFijoTamVariable =
    List.of(
        Trio.of(Ejercicio2::sort, TipoAjuste.NLOGN_0, "QuickSort(Complejidad)")
    );

// Para el analisis de la complejidad con distintos umbral
private static List<Trio<BiConsumer<List<Integer>, Integer>, Integer, String>> metodosUmbralVariable =
    List.of(
        Trio.of(Ejercicio2::sort, umbrales.get(0), "QuickSort(umbral4)"),
        Trio.of(Ejercicio2::sort, umbrales.get(1), "QuickSort(umbral25)"),
        Trio.of(Ejercicio2::sort, umbrales.get(2), "QuickSort(umbral100)"),
        Trio.of(Ejercicio2::sort, umbrales.get(3), "QuickSort(umbral500)")
    );

// Genera fichero de listas de enteros con tamaño variable
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);
    }
}

private static List<Integer> generalistaEnteros(Integer tamLista) {
    List<Integer> res = List2.of();
    for(int j=0; j<tamLista; j++) {
        res.add(0+rr.nextInt(1000000-0));
    }
    return res;
}

// Generando ficheros con tiempo de ejecucion
private static void generaFicherosTiempoEjecucionComplejidad(
    List<Trio<BiConsumer<List<Integer>, Integer>, TipoAjuste, String>> metodos) {
    for (int i=0; i<metodos.size(); i++) {
        String ficheroSalida = String.format("ficheros/2Tiempos1%s.csv",
            metodos.get(i).third());
        System.out.println(ficheroSalida);
        testTiemposEjecucionComplejidad(metodos.get(i).first(), ficheroSalida);
    }
}

private static void generaFicherosTiempoEjecucionUmbral(
    List<Trio<BiConsumer<List<Integer>, Integer>, Integer, String>> metodos) {
    for (int i=0; i<metodos.size(); i++) {
        String ficheroSalida = String.format("ficheros/2Tiempos1%s.csv",
            metodos.get(i).third());
        System.out.println(ficheroSalida);
        testTiemposEjecucionUmbral(metodos.get(i).first(), ficheroSalida, metodos.get(i).second());
    }
}

public static void generaFicherosTiempoEjecucion() {
    generaFicherosTiempoEjecucionComplejidad(metodosComplejidadUmbralFijoTamVariable);
    generaFicherosTiempoEjecucionUmbral(metodosUmbralVariable);
}

```

```

public static void testTiemposEjecucionComplejidad(BiConsumer<List<Integer>, Integer> funcion,
String ficheroTiempo) {
    Map<Problema, Double> tiempos = new HashMap<>();
    Map<Integer, Double> tiemposMedios; // tiempos medios por 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++) {
            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, umbralFijo); // 4
            }
            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())));

    ResultadosToFile(tiemposMedios.entrySet().stream()
        .map(x -> TResultD.of(x.getKey(), x.getValue()).toString()),
        ficheroTiempo, true);
}

```

```

public static void testTiemposEjecucionUmbral(BiConsumer<List<Integer>, Integer> funcion,
String ficheroTiempos, Integer umbral) {
    Map<Problema, Double> tiempos = new HashMap<>();
    Map<Integer, Double> tiemposMedios;

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

    Integer nMed = numMediciones;

    for (int iter=0; iter<nMed; iter++) {
        for (int j=0; j<lineasListas.size(); j++) {
            String linealista = lineasListas.get(j);
            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, j, tamLista);
            warmup(funcion, le, umbral);

            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()))));

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

```

```

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

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);
    }
}

// Generando graficas
public static void muestraGraficas() {
    muestraGraficasMetodos(metodosComplejidadUmbralFijoTamVariable);
    muestraGraficasUmbral(metodosUmbralVariable);
}

public static void muestraGraficasMetodos(List<Trio<BiConsumer<List<Integer>, Integer>,
    TipoAjuste, String>> metodosComplejidad) {
    List<String> ficherosSalida = new ArrayList<>();
    List<String> labels = new ArrayList<>();

    for (int i=0; i<metodosComplejidadUmbralFijoTamVariable.size(); i++) {
        String ficheroSalida = String.format("ficheros/2Tiempos1%s.csv",
            metodosComplejidad.get(i).third());
        ficherosSalida.add(ficheroSalida);
        String label = metodosComplejidad.get(i).third();
        System.out.println(label);

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

        // Obtener ajusteString para mostrarlo en grafica 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));
    }
}

public static void muestraGraficasUmbral(List<Trio<BiConsumer<List<Integer>, Integer>,
    Integer, String>> metodosUmbral) {
    List<String> ficherosSalida = new ArrayList<>();
    List<String> labels = new ArrayList<>();

    for(int i=0; i<metodosUmbral.size(); i++) {
        System.out.println(metodosUmbral.size());
        String ficheroSalida = String.format("ficheros/2Tiempos1%s.csv", metodosUmbral.get(i).third());
        ficherosSalida.add(ficheroSalida);
        String label = metodosUmbral.get(i).third();
        System.out.println(label);
        TipoAjuste tipoAjuste = TipoAjuste.NLOGN_0;
        GraficosAjuste.show(ficheroSalida, tipoAjuste, label);

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

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

// Tipos (records) auxiliares
record TResultID(Integer tam, Double t) {

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

```



## 2.3. VOLCADO DE PANTALLA Y GRÁFICAS

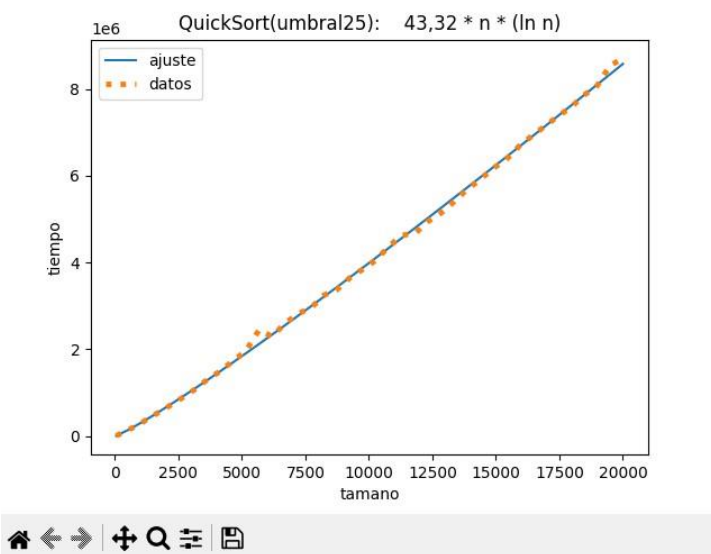
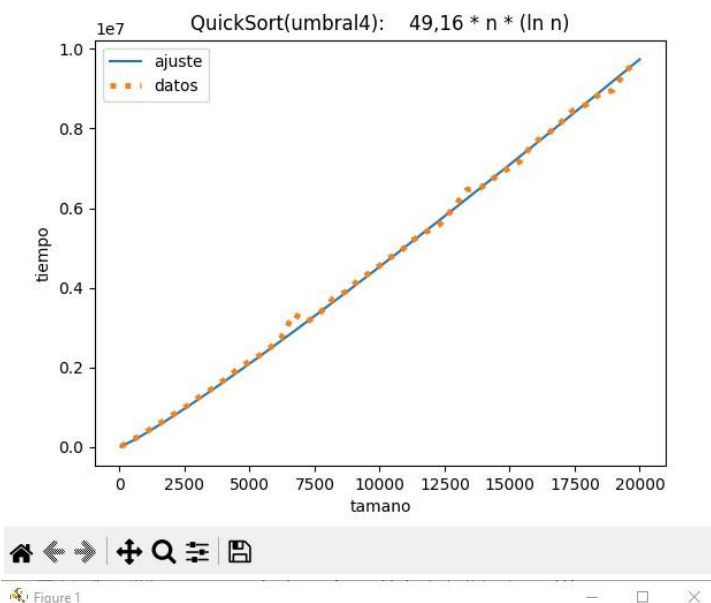
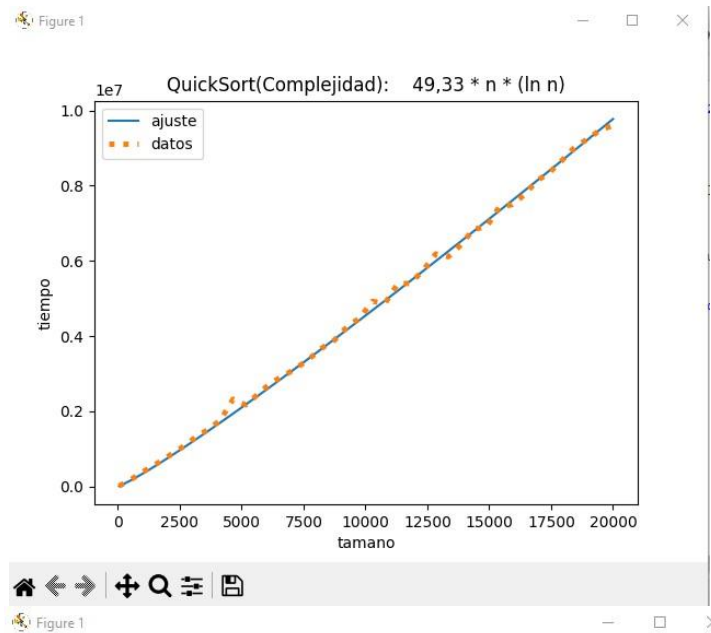
\* TEST EJERCICIO 1 \*

\*Analizamos correcto funcionamiento\*

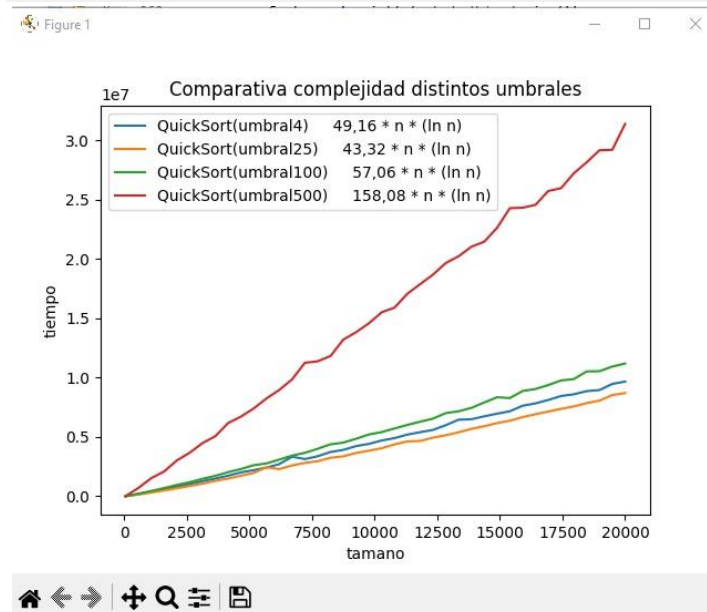
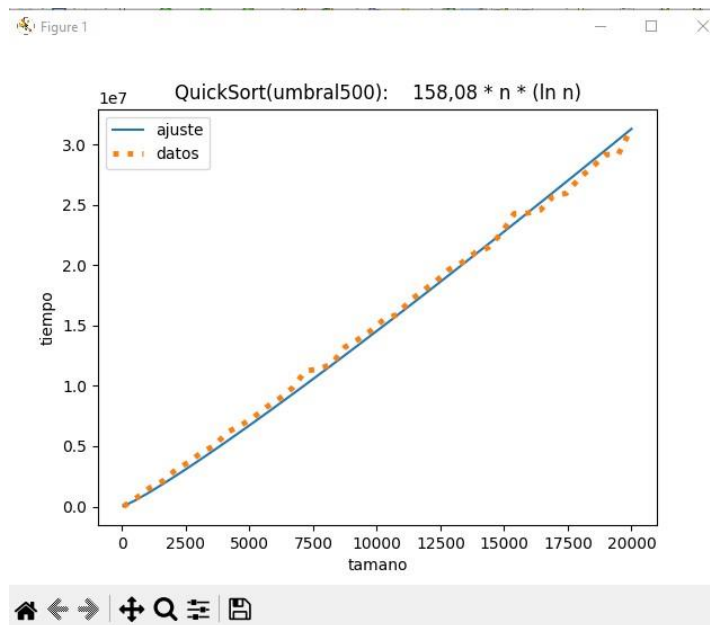
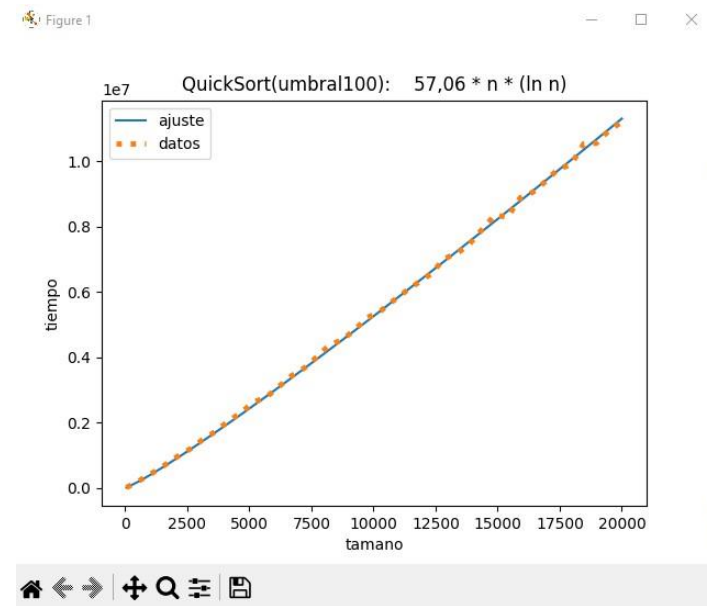
- Lista original: [23, 5, 56, 67, 57, 32, 21, 8, 24, 11, 1, 44, 89, 2]  
- Lista ordenada con umbral pequeño (4): [1, 2, 5, 8, 11, 21, 23, 24, 32, 44, 56, 57, 67, 89]  
- Lista ordenada con umbral grande (20): [1, 2, 5, 8, 11, 21, 23, 24, 32, 44, 56, 57, 67, 89]

\*Analizamos tiempo de ejecucion con graficas\*

```
QuickSort(Complejidad)
Solutions = a = 49,33
DEBUG - .plot command: ret_7f52788f_c1b8_4d01_af86_3dcf0edd4932 = plt.plot(np.array([50.0, 561.0, 1073.0, 1584.0, 2096.0, 2607.0, 3119.0, 3630.0, 4142.0, 4653.0, 5165.0, 5676.0, 6188.0,
DEBUG - .plot command: ret_b36bfea9_6e24_4a74_a561_b66ed1bfblac = plt.plot(np.array([50.0, 561.0, 1073.0, 1584.0, 2096.0, 2607.0, 3119.0, 3630.0, 4142.0, 4653.0, 5165.0, 5676.0, 6188.0,
DEBUG - .plot command: plt.title("QuickSort(Complejidad): 49,33 * n * (ln n)")
DEBUG - .plot command: ret_e9e99c0d_585b_47a8_ab18_7006c19d2a50 = plt.legend()
DEBUG - .plot command: ret_573d8c7f_56db_467e_bb06_12a0ef88a1da = plt.xlabel("tamano")
DEBUG - .plot command: ret_c227ee6b_d9d1_45c7_9722_404db41127ba = plt.ylabel("tiempo")
DEBUG - Commands... : [python, C:\Users\pc\AppData\Local\Temp\1669062247790-0\exec.py]
WARN -
QuickSort(Complejidad)
Solutions = a = 49,33
4
QuickSort(umbral4)
Solutions = a = 49,16
DEBUG - .plot command: ret_284946b2_d24f_4136_b853_fdea3adfcfbf = plt.plot(np.array([50.0, 561.0, 1073.0, 1584.0, 2096.0, 2607.0, 3119.0, 3630.0, 4142.0, 4653.0, 5165.0, 5676.0, 6188.0,
DEBUG - .plot command: ret_cc82f6b9_a37a_494c_9fc8_2001b649530b = plt.plot(np.array([50.0, 561.0, 1073.0, 1584.0, 2096.0, 2607.0, 3119.0, 3630.0, 4142.0, 4653.0, 5165.0, 5676.0, 6188.0,
DEBUG - .plot command: plt.title("QuickSort(umbral4): 49,16 * n * (ln n)")
DEBUG - .plot command: ret_84b9c19b_545d_425d_9133_6390fb3de801 = plt.legend()
DEBUG - .plot command: ret_47fbbbcf_d01e_47bb_9130_f75958701723 = plt.xlabel("tamano")
DEBUG - .plot command: ret_c3cf09ac_9ae1_469f_9e1b_62f913e14f16 = plt.ylabel("tiempo")
DEBUG - Commands... : [python, C:\Users\pc\AppData\Local\Temp\1669062280714-0\exec.py]
WARN -
QuickSort(umbral4)
Solutions = a = 49,16
4
QuickSort(umbral25)
Solutions = a = 43,32
DEBUG - .plot command: ret_da9f6ca7_9980_499e_99b6_85987cc2ee9f = plt.plot(np.array([50.0, 561.0, 1073.0, 1584.0, 2096.0, 2607.0, 3119.0, 3630.0, 4142.0, 4653.0, 5165.0, 5676.0, 6188.0,
DEBUG - .plot command: ret_4b5f5312_8617_4f21_a564_4c064734d04b = plt.plot(np.array([50.0, 561.0, 1073.0, 1584.0, 2096.0, 2607.0, 3119.0, 3630.0, 4142.0, 4653.0, 5165.0, 5676.0, 6188.0,
DEBUG - .plot command: plt.title("QuickSort(umbral25): 43,32 * n * (ln n)")
DEBUG - .plot command: ret_fef36637_2497_4cab_be74_8703e541c031 = plt.legend()
DEBUG - .plot command: ret_1733d798_0920_4154_b5a2_3afd6ab0afc7 = plt.xlabel("tamano")
DEBUG - .plot command: ret_e4949f99_7b31_4189_8854_fd83e68ecdff = plt.ylabel("tiempo")
DEBUG - Commands... : [python, C:\Users\pc\AppData\Local\Temp\1669062303826-0\exec.py]
WARN -
QuickSort(umbral25)
Solutions = a = 43,32
4
QuickSort(umbral100)
Solutions = a = 57,06
DEBUG - .plot command: ret_dee41ce8_3a22_4f26_9c04_a9f00c5a3fe4 = plt.plot(np.array([50.0, 561.0, 1073.0, 1584.0, 2096.0, 2607.0, 3119.0, 3630.0, 4142.0, 4653.0, 5165.0, 5676.0, 6188.0,
DEBUG - .plot command: ret_e59e3bd5_7dc7_4018_9258_53bd8848233e = plt.plot(np.array([50.0, 561.0, 1073.0, 1584.0, 2096.0, 2607.0, 3119.0, 3630.0, 4142.0, 4653.0, 5165.0, 5676.0, 6188.0,
DEBUG - .plot command: plt.title("QuickSort(umbral100): 57,06 * n * (ln n)")
DEBUG - .plot command: ret_5e5980a7_84e0_46b3_88ee_80d9bcfdb605 = plt.legend()
DEBUG - .plot command: ret_380a9810_204e_4a1c_b93d_28ea2fd3dc78 = plt.xlabel("tamano")
DEBUG - .plot command: ret_4f512a18_ac79_434e_9b52_ac2154335517 = plt.ylabel("tiempo")
DEBUG - Commands... : [python, C:\Users\pc\AppData\Local\Temp\1669062330163-0\exec.py]
WARN -
QuickSort(umbral100)
Solutions = a = 57,06
4
QuickSort(umbral500)
Solutions = a = 158,08
DEBUG - .plot command: ret_c01ad1e7_ffd0_438e_8bce_106075a752e7 = plt.plot(np.array([50.0, 561.0, 1073.0, 1584.0, 2096.0, 2607.0, 3119.0, 3630.0, 4142.0, 4653.0, 5165.0, 5676.0, 6188.0,
DEBUG - .plot command: ret_f0219c3a_8bd1_4af5_071e_e4a7df71d952 = plt.plot(np.array([50.0, 561.0, 1073.0, 1584.0, 2096.0, 2607.0, 3119.0, 3630.0, 4142.0, 4653.0, 5165.0, 5676.0, 6188.0,
DEBUG - .plot command: plt.title("QuickSort(umbral500): 158,08 * n * (ln n)")
DEBUG - .plot command: ret_08a3b72d_0d19_40db_b420_6edda46bdc8d = plt.legend()
DEBUG - .plot command: ret_70152188_9a2a_42ab_b190_fff4e9f6d157 = plt.xlabel("tamano")
DEBUG - .plot command: ret_1fb8039d_9276_46b4_8dd8_3ffcd016659f = plt.ylabel("tiempo")
DEBUG - Commands... : [python, C:\Users\pc\AppData\Local\Temp\1669062356729-0\exec.py]
WARN -
QuickSort(umbral500)
Solutions = a = 158,08
DEBUG - .plot command: plt.title("Comparativa complejidad distintos umbrales")
DEBUG - .plot command: ret_2b766d76_00d2_47af_9703_0125c233443 = plt.plot(np.array([50.0, 561.0, 1073.0, 1584.0, 2096.0, 2607.0, 3119.0, 3630.0, 4142.0, 4653.0, 5165.0, 5676.0, 6188.0,
DEBUG - .plot command: ret_5c871545_2edb_4304_8c12_66e201c26070 = plt.plot(np.array([50.0, 561.0, 1073.0, 1584.0, 2096.0, 2607.0, 3119.0, 3630.0, 4142.0, 4653.0, 5165.0, 5676.0, 6188.0,
DEBUG - .plot command: ret_79d33a2e_1442_486c_875c_9759c86932dd = plt.plot(np.array([50.0, 561.0, 1073.0, 1584.0, 2096.0, 2607.0, 3119.0, 3630.0, 4142.0, 4653.0, 5165.0, 5676.0, 6188.0,
DEBUG - .plot command: ret_019c9ef4_efd4_435f_be6b_9e86d1c66d71 = plt.plot(np.array([50.0, 561.0, 1073.0, 1584.0, 2096.0, 2607.0, 3119.0, 3630.0, 4142.0, 4653.0, 5165.0, 5676.0, 6188.0,
DEBUG - .plot command: ret_4adbd524_d828_4a01_83ae_7dfe60d43d14 = plt.legend()
DEBUG - .plot command: ret_885b1bf2_7c6b_4883_8f23_4f8a437bd3d5 = plt.xlabel("tamano")
DEBUG - .plot command: ret_9247b482_ed60_4929_a078_f7890bf7db5d = plt.ylabel("tiempo")
DEBUG - Commands... : [python, C:\Users\pc\AppData\Local\Temp\1669062388166-0\exec.py]
WARN -
```







## 3. EJERCICIO 3

### 3.1. CÓDIGO EJERCICIO

```
package ejercicios;

import java.util.List;

import us.lsi.common.List2;
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 {

    // Solucion para arboles binarios
    public static List<String> fEjercicio3Binary(BinaryTree<Character> tree, Character e) {
        return fEjercicio3Binary(tree, e, "", List2.of());
    }

    private static List<String> fEjercicio3Binary(BinaryTree<Character> tree, Character e, String camino, List<String> ac) {
        return switch (tree) {
            case BEmpty<Character> t -> ac;
            case BLeaf<Character> t -> {
                String posibleCamino = camino+t.label();
                if(!t.label().equals(e)) {
                    ac.add(posibleCamino);
                }
                yield ac;
            }
            case BTree<Character> t -> {
                String posibleCamino = camino+t.label();
                if(!t.label().equals(e)) {
                    fEjercicio3Binary(t.left(), e, posibleCamino, ac);
                    fEjercicio3Binary(t.right(), e, posibleCamino, ac);
                }
                yield ac;
            }
        };
    }

    // Solucion para arboles n-arios
    public static List<String> fEjercicio3Nary(Tree<Character> tree, Character e) {
        return fEjercicio3Nary(tree, e, "", List2.of());
    }

    private static List<String> fEjercicio3Nary(Tree<Character> tree, Character e, String camino, List<String> ac) {
        return switch (tree) {
            case TEmpty<Character> t -> ac;
            case TLeaf<Character> t -> {
                String posibleCamino = camino+t.label();
                if(!t.label().equals(e)) {
                    ac.add(posibleCamino);
                }
                yield ac;
            }
            case TNary<Character> t -> {
                String posibleCamino = camino+t.label();
                if(!t.label().equals(e)) {
                    for(Tree<Character> tr: t.elements()) {
                        fEjercicio3Nary(tr, e, posibleCamino, ac);
                    }
                }
                yield ac;
            }
        };
    }
}
```

### 3.2. CÓDIGO TEST

```

package ejerciciosTests;

import java.util.List;

import ejercicios.Ejercicio3;
import us.lsi.common.Files2;
import us.lsi.common.Pair;
import us.lsi.tiposrecursivos.BinaryTree;
import us.lsi.tiposrecursivos.Tree;

public class TestEjercicio3 {

    public static void main(String[] args) {

        // LECTURAS DE FICHERO
        String rutaFicheroBinarios = "ficheros/Ejercicio3DatosEntradaBinario.txt";
        String rutaFicheroNarios = "ficheros/Ejercicio3DatosEntradaNario.txt";

        List
```

### 3.3. VOLCADO PANTALLA

```
* TEST EJERCICIO 3*

*Arboles Binarios*
Arbol: A(B,C)   Caracter: D   [[AB, AC]]
Arbol: A(B,C)   Caracter: C   [[AB]]
Arbol: A(B,C)   Caracter: A   [[]]
Arbol: A(B(C,D),E(F,_)) Caracter: H   [[ABC, ABD, AEF]]
Arbol: A(B(C,D),E(F,_)) Caracter: D   [[ABC, AEF]]
Arbol: A(B(C,D(E,F(G,H))),I(J,K)) Caracter: H   [[ABC, ABDE, ABD FG, AIJ, AIK]]
Arbol: A(B(C,D(E,F(G,H))),I(J,K)) Caracter: C   [[ABDE, ABD FG, ABD FH, AIJ, AIK]]

*Arboles n-arios*
Arbol: A(B,C,D) Caracter: A   [[]]
Arbol: A(B,C,D) Caracter: C   [[AB, AD]]
Arbol: A(B,C,D) Caracter: D   [[AB, AC]]
Arbol: A(B(C,D,E),F(G,H,I),J(K,L)) Caracter: F   [[ABC, ABD, ABE, AJK, AJL]]
Arbol: A(B(C,D,E),F(G,H,I),J(K,L)) Caracter: K   [[ABC, ABD, ABE, AFG, AFH, AFI, AJL]]
Arbol: A(B(C,D(E,F(G,H,I),J),K)) Caracter: D   [[ABC, ABK]]
Arbol: A(B(C,D(E,F(G,H,I),J),K)) Caracter: I   [[ABC, ABDE, ABD FG, ABD FH, ABD J, ABK]]
```

## 4. EJERCICIO 4

### 4.1. CÓDIGO EJERCICIO

```
package ejercicios;

import java.util.List;

import us.lsi.common.List2;
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 {

    // Solucion para arboles binarios
    public static Boolean fEjercicio4Binary(BinaryTree<String> tree) {
        return fEjercicio4BinaryAux(tree, TuplaEj4.of(0, true)).b();
    }

    private static TuplaEj4 fEjercicio4BinaryAux(BinaryTree<String> tree, TuplaEj4 ac) {
        return switch (tree) {
            case BEmpty<String> t -> ac;
            case BLeaf<String> t -> TuplaEj4.of(ac.a()+cuentaVocales(t.label()), ac.b());
            case BTree<String> t -> {
                TuplaEj4 resLeft = fEjercicio4BinaryAux(t.left(), TuplaEj4.of(ac.a(), ac.b()));
                TuplaEj4 resRight = fEjercicio4BinaryAux(t.right(), TuplaEj4.of(ac.a(), ac.b()));
                yield TuplaEj4.of(cuentaVocales(t.label())+resLeft.a()+resRight.a(),
                    (resLeft.a()==resRight.a()) && resLeft.b() && resRight.b());
            }
        };
    }

    // Solucion para arboles n-arios
    public static Boolean fEjercicio4Nary(Tree<String> tree) {
        return fEjercicio4NaryAux(tree, TuplaEj4.of(0, true)).b();
    }

    private static TuplaEj4 fEjercicio4NaryAux(Tree<String> tree, TuplaEj4 ac) {
        return switch (tree) {
            case TEmpty<String> t -> ac;
            case TLeaf<String> t -> TuplaEj4.of(ac.a()+cuentaVocales(t.label()), ac.b());
            case TNary<String> t -> {
                List<TuplaEj4> res = List2.of();
                for(int i=0; i<t.elements().size(); i++) {
                    TuplaEj4 resArbol = fEjercicio4NaryAux(t.elements().get(i), TuplaEj4.of(ac.a(), ac.b()));
                    res.add(resArbol);
                }
                yield TuplaEj4.of(cuentaVocales(t.label())+res.stream().mapToInt(x -> x.a()).sum(),
                    (res.stream().mapToInt(x -> x.a()).distinct().count()==1) &&
                    res.stream().map(x -> x.b()).allMatch(x -> x==true));
            }
        };
    }

    // Funcion auxiliar que se encarga de contar las vocales de una cadena
    private static Integer cuentaVocales(String cadena) {
        Integer res = 0;
        for(int i=0; i<cadena.length(); i++) {
            if(cadena.charAt(i)=='a' || cadena.charAt(i)=='e' || cadena.charAt(i)=='i' ||
                cadena.charAt(i)=='o' || cadena.charAt(i)=='u') {
                res++;
            }
        }
        return res;
    }

    // Tupla que hemos usado para que se vayan actualizando los valores
    private static record TuplaEj4(Integer a, Boolean b) {

        public static TuplaEj4 of(Integer a, Boolean b) {
            return new TuplaEj4(a, b);
        }
    }
}
```

## 4.2. CÓDIGO TEST

```
package ejerciciosTests;

import java.util.List;

import ejercicios.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) {

        // LECTURAS DE FICHERO
        String rutaFicheroBinarios = "ficheros/Ejercicio4DatosEntradaBinario.txt";
        String rutaFicheroNarios = "ficheros/Ejercicio4DatosEntradaNario.txt";

        List<BinaryTree<String>> datosBinarios = Files2.streamFromFile(rutaFicheroBinarios)
            .map(linea -> BinaryTree.parse(linea))
            .toList();

        List<Tree<String>> datosNarios = Files2.streamFromFile(rutaFicheroNarios)
            .map(linea -> Tree.parse(linea))
            .toList();

        // TEST
        System.out.println("* TEST EJERCICIO 4 *");

        System.out.println("\n*Arboles binarios*");
        for(BinaryTree<String> tree: datosBinarios) {
            System.out.println(tree + ": " + Ejercicio4.fEjercicio4Binary(tree));
        }

        System.out.println("\n*Arboles n-arios*");
        for(Tree<String> tree: datosNarios) {
            System.out.println(tree + ": " + Ejercicio4.fEjercicio4Nary(tree));
        }

    }

}
```

## 4.3. VOLCADO DE PANTALLA Y GRÁFICAS

---

\* TEST EJERCICIO 4 \*

```
*Arboles binarios*
pepe(pepa,pepe): true
pepe(pepa,pep): false
ada(eda(ola,ale),eda(ele,ale)): true
ada(eda(ola,ale),eda(ele,al)): false
cafe(taza(bote,bolsa),perro(gato,leon)): true
cafe(taza(bote,bolsa),perro(gato,_)): false
cafe(taza(bote,bolsa),perro(gato,tortuga)): false

*Arboles n-arios*
pepe(pepa,pepe,pepo): true
pepe(pepa,pepe,pep): false
ada(eda(ola,ale,elo),eda(ele,ale,alo)): true
ada(eda(ola,ale,elo),eda(ele,ale,al)): false
cafe(taza(bote,bolsa,vaso),perro(gato,leon,tigre)): true
cafe(taza(bote,bolsa,vaso),perro(gato,leon)): false
cafe(taza(bote,bolsa,vaso),perro(gato,tortuga)): false
```