MEMORIA

PRACTICA INDIVIDUAL 2

Jaime Linares Barrera

2º Ingeniería Informática - Ingeniería del Software, Grupo 4

1. EJERCICIO 1

1.1. CÓDIGO EJERCICIO

```java
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

```java
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


    // Trios 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);
    }
```

```java
@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);
        }

    }

    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 <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%s.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 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);
    }

}


record Problema(Integer tam) {

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

}
```

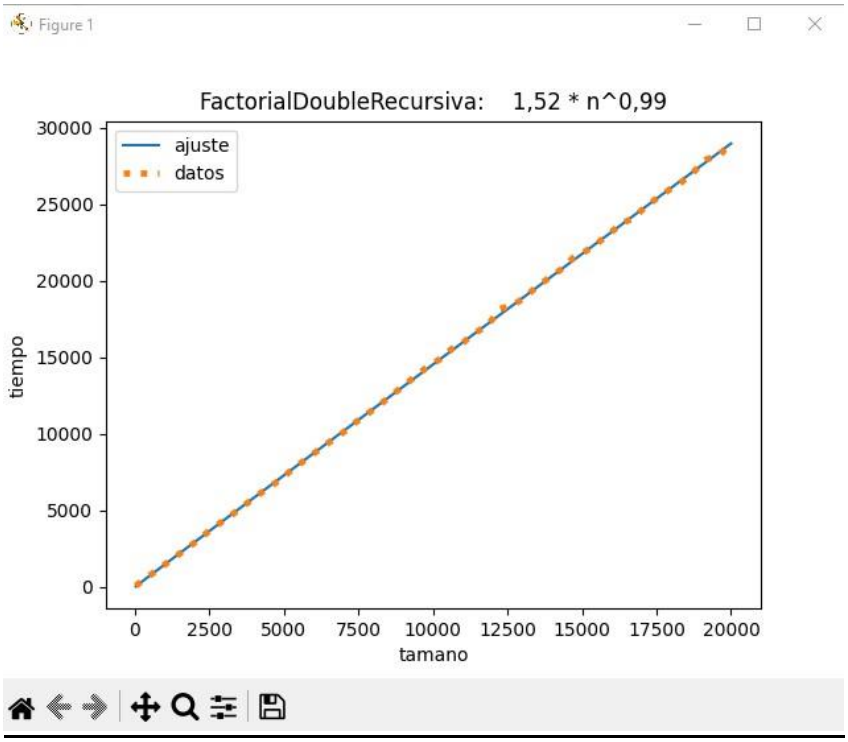
1.3. VOLCADO DE PANTALLA Y GRÁFICAS

```
* 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_b2d91182_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, 5
DEBUG - .plot command: ret_01864d41_c983_439f_b924_5b4beeda1628 = 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, 5
DEBUG - .plot command: plt.title("FactorialDoubleRecursiva:    1,52 * n^0,99")
DEBUG - .plot command: ret_5c816865_32a7_453a_8fa7_fac33ebe9c4a = plt.legend()
DEBUG - .plot command: ret_10e836b7_f583_4cc2_a841_34190502ef1e = plt.xlabel("tamano")
DEBUG - .plot command: ret_34889239_b1b0_4935_bace_af8defc052cc = plt.ylabel("tiempo")
DEBUG - Commands... : [python, C:\Users\pc\AppData\Local\Temp\1668985062748-0\exec.py]
WARN -
FactorialDoubleRecursiva
Solutions = a = 1,52,b = 0,99
FactorialDoubleIterativa
Solutions = a = 4,76,b = 1,02
DEBUG - .plot command: ret_44144a21_c1ab_41f6_9172_2355e9a3f960 = 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, 5
DEBUG - .plot command: ret_56c136eb_1f3b_40e1_bdd6_2a89435f5a1e = 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, 5
DEBUG - .plot command: plt.title("FactorialDoubleIterativa:    4,76 * n^1,02")
DEBUG - .plot command: ret_242508ab_edc5_437f_a051_93513c5960d9 = plt.legend()
DEBUG - .plot command: ret_521382cc_3e0e_40d2_abda_2b0a016288c8 = plt.xlabel("tamano")
DEBUG - .plot command: ret_eb38dfae_eac9_4129_8585_1a03820ee02c = plt.ylabel("tiempo")
DEBUG - Commands... : [python, C:\Users\pc\AppData\Local\Temp\1668985116952-0\exec.py]
WARN -
FactorialDoubleIterativa
FactorialBigIntegerRecursiva
Solutions = a = 0,12,b = 2,10
DEBUG - .plot command: ret_c9738f36_7b0b_4901_bedc_daf199adc390 = 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, 5
DEBUG - .plot command: ret_d949741b_7847_4947_b75b_3e2e848aff9f = 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, 5
DEBUG - .plot command: plt.title("FactorialBigIntegerRecursiva:    0,124 * n^2,10")
DEBUG - .plot command: ret_c79ccaf8_7621_4b3a_902e_6a01c0e718f5 = plt.legend()
DEBUG - .plot command: ret_985f9a5e_d128_4677_a085_f6f32be5362f = plt.xlabel("tamano")
DEBUG - .plot command: ret_aee4cf42_6848_47a8_90a5_ba585678ec2b = plt.ylabel("tiempo")
DEBUG - Commands... : [python, C:\Users\pc\AppData\Local\Temp\1668985140313-0\exec.py]
WARN -
FactorialBigIntegerRecursiva
Solutions = a = 0,12,b = 2,10
FactorialBigIntegerIterativa
Solutions = a = 0,15,b = 2,09
DEBUG - .plot command: ret_9a30adfe_48c6_4e46_88c5_5081ef3aa417 = 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, 5
DEBUG - .plot command: ret_a3556ff1_40b7_4473_af9c_6213f7c6ae10 = 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, 5
DEBUG - .plot command: plt.title("FactorialBigIntegerIterativa:    0,146 * n^2,09")
DEBUG - .plot command: ret_627df4fc_9064_4359_80ea_4318d71a393f = plt.legend()
DEBUG - .plot command: ret_6e2c909b_1421_4fbb_9982_63d3b92bad19 = plt.xlabel("tamano")
DEBUG - .plot command: ret_4e8f5c57_9c74_489a_8fdd_c1ca8815b7e6 = plt.ylabel("tiempo")
DEBUG - Commands... : [python, C:\Users\pc\AppData\Local\Temp\1668985162492-0\exec.py]
WARN -
FactorialBigIntegerIterativa
Solutions = a = 0,15,b = 2,09
DEBUG - .plot command: plt.title("Calculo Factorial")
DEBUG - .plot command: ret_8e4f3271_db24_48ff_83c3_4ed1a3f3d67b = 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, 5
DEBUG - .plot command: ret_559a552b_c62b_4306_8c3a_b0b472197302 = 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, 5
DEBUG - .plot command: ret_0a712c1f_3661_4473_8b49_9a0227e25a37 = 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, 5
DEBUG - .plot command: ret_e9fb4902_346b_4df5_8049_a31a34994680 = 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, 5
DEBUG - .plot command: ret_13a62e8d_c734_41e0_a98b_f610bb0b11ce = plt.legend()
DEBUG - .plot command: ret_30238aca_a798_4962_af11_0bc79fe6f3d5 = plt.xlabel("tamano")
DEBUG - .plot command: ret_2340e8b5_ad90_4105_8c00_934eb54f6eed = plt.ylabel("tiempo")
DEBUG - Commands... : [python, C:\Users\pc\AppData\Local\Temp\1668985185927-0\exec.py]
WARN -
```



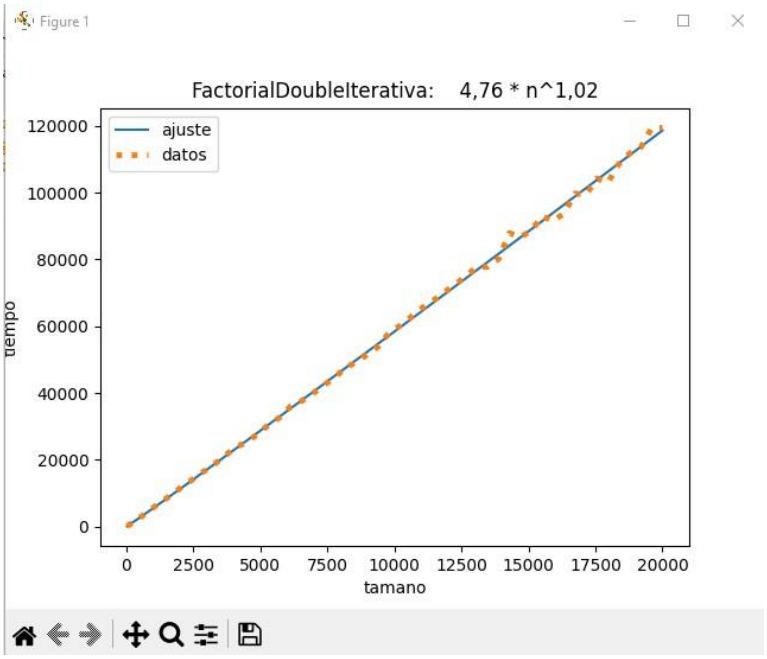


Figure 1
FactorialDoubleIterativa: 4,76 * n^1,02
ajuste
datos
tiempo
tamano
0
20000
40000
60000
80000
100000
120000
0
2500
5000
7500
10000
12500
15000
17500
20000

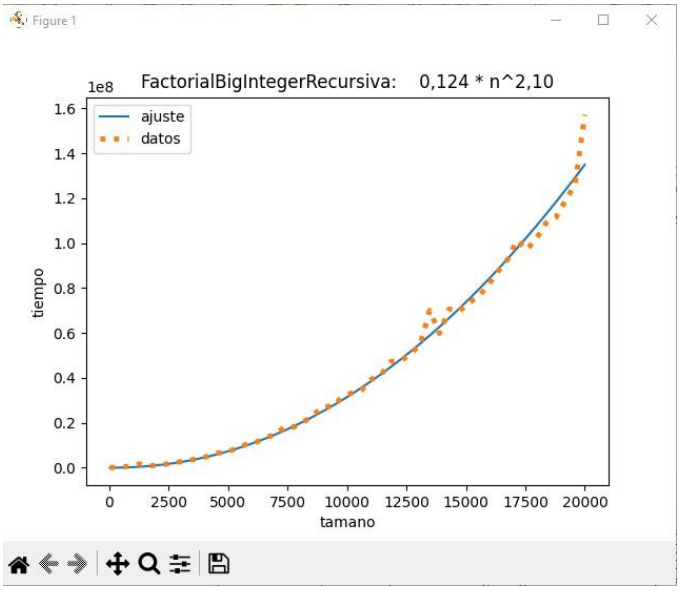


Figure 1
1e8
FactorialBigIntegerRecursiva: 0,124 * n^2,10
ajuste
datos
tiempo
tamano
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
0
2500
5000
7500
10000
12500
15000
17500
20000

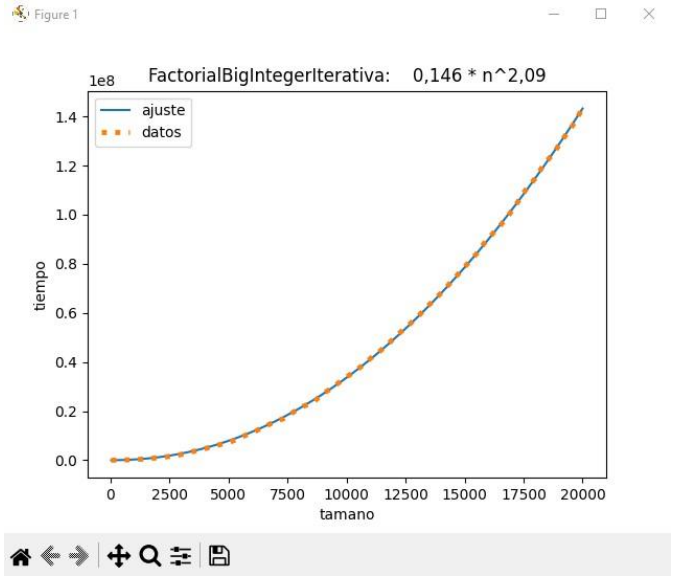
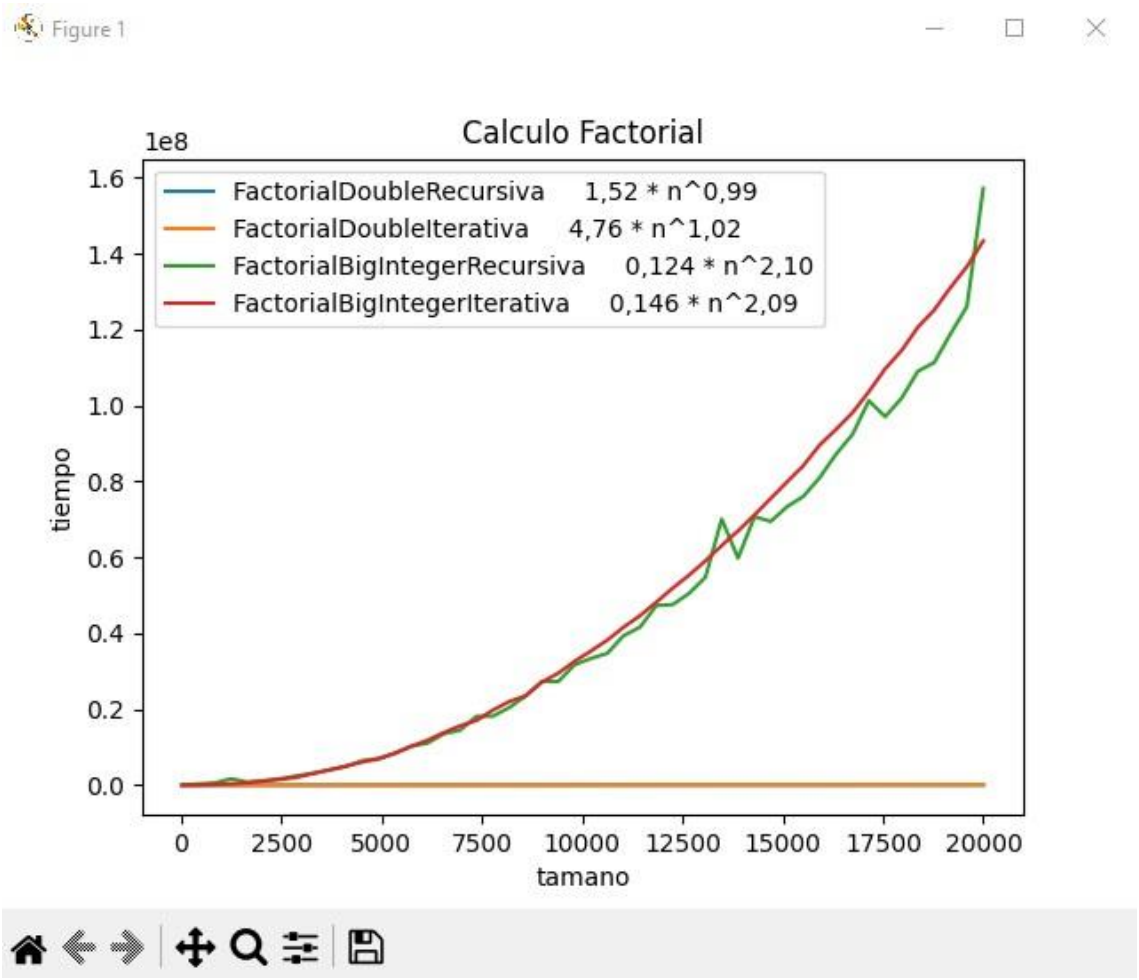


Figure 1
1e8
FactorialBigIntegerIterativa: 0,146 * n^2,09
ajuste
datos
tiempo
tamano
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0
2500
5000
7500
10000
12500
15000
17500
20000



2. EJERCICIO 2

2.1. CÓDIGO EJERCICIO

```java
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; // tamano minimo de lista
    private static Integer sizeMax = 20000; // tamano maximo de lista
    private static Integer numListas = 40; // numero 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; // numero de mediciones de tiempo de cada caso (numero de experimentos)
    private static Integer numIter = 5; // numero de iteraciones para cada medicion de tiempo
    private static Integer numIterWarmup = 100; // numero 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 tamano
    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 tamano 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(","));
            Resultados.toFile(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);
}
```

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

    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())));

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

```java
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()))));

    Resultados.toFile(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 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);
        }

    }

    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_b66ed1bfb1ac = 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_fdea3adfcbfb = 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_fd83e68ecddf = 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_871e_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_01252c233443 = 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 -
```

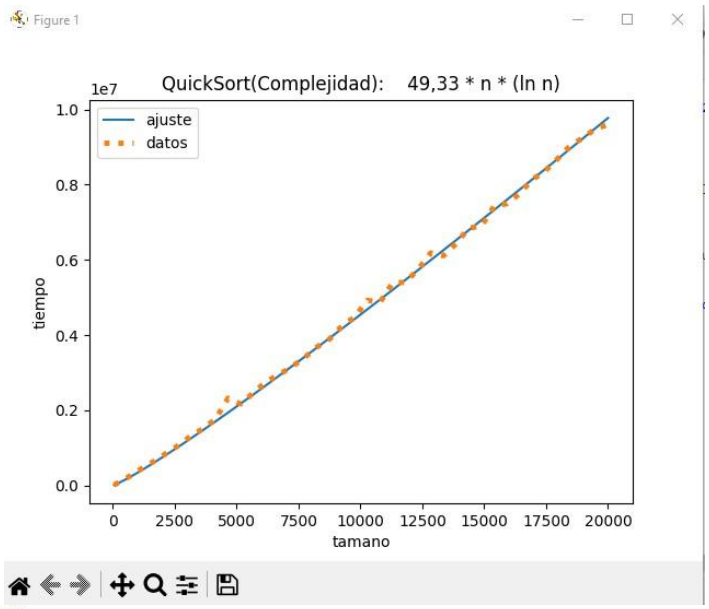


Figure 1
QuickSort(Complejidad): 49,33 * n * (ln n)
1e7
ajuste
datos
tiempo
tamano

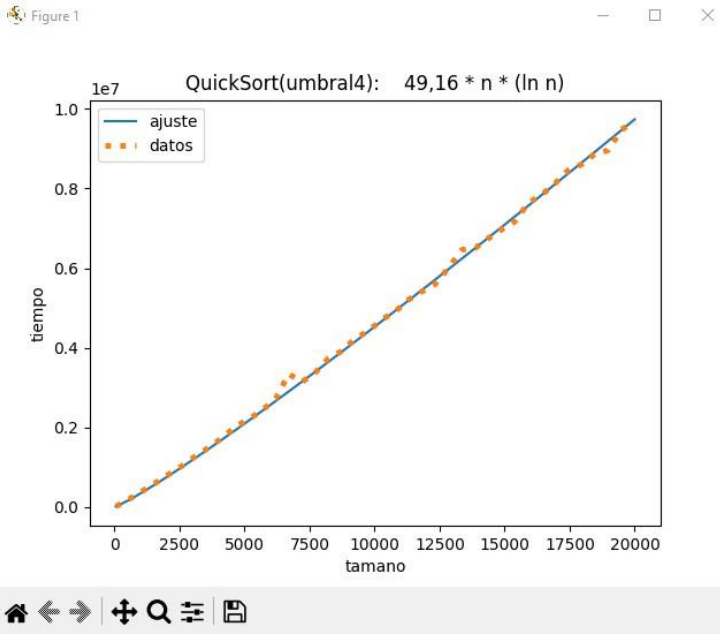


Figure 1
QuickSort(umbral4): 49,16 * n * (ln n)
1e7
ajuste
datos
tiempo
tamano

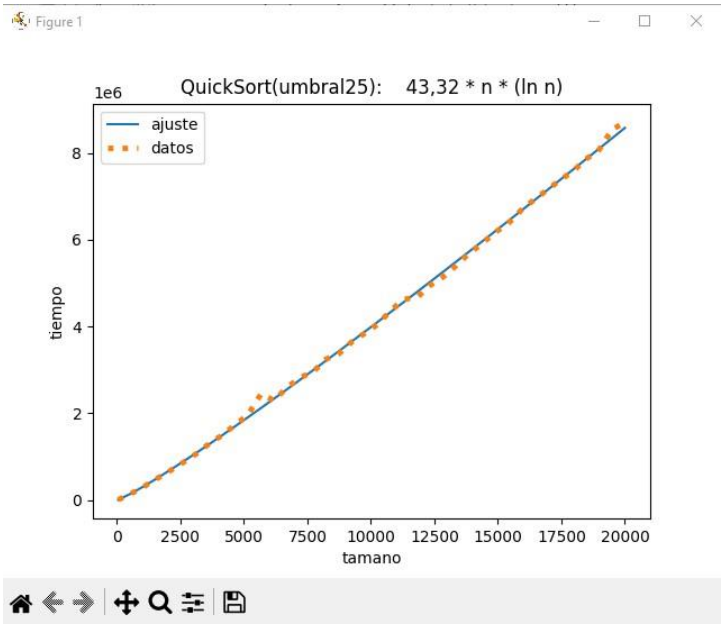


Figure 1
QuickSort(umbral25): 43,32 * n * (ln n)
1e6
ajuste
datos
tiempo
tamano

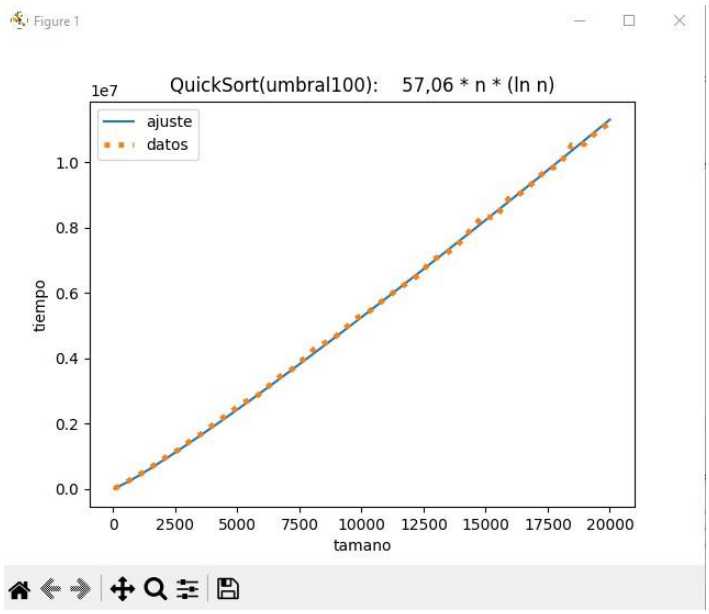


Figure 1
QuickSort(umbral100): 57,06 * n * (ln n)
1e7
ajuste
datos
tiempo
tamano
0.0
0.2
0.4
0.6
0.8
1.0
0
2500
5000
7500
10000
12500
15000
17500
20000

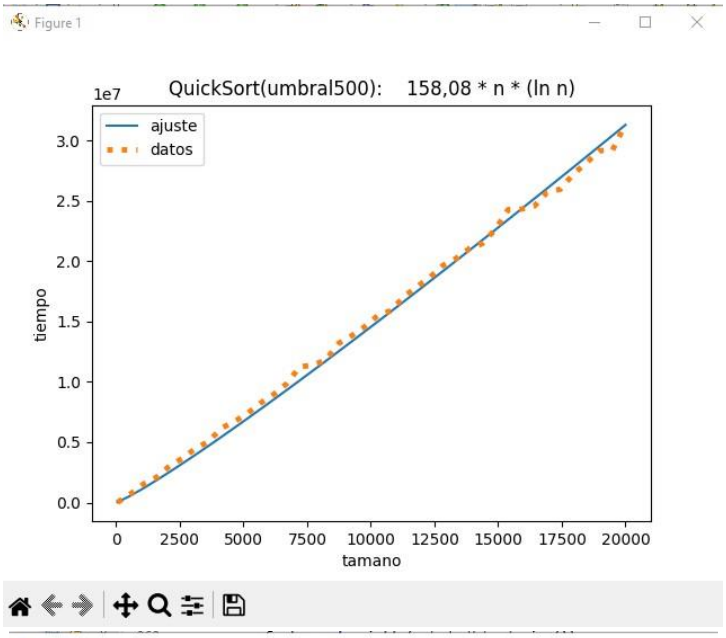


Figure 1
QuickSort(umbral500): 158,08 * n * (ln n)
1e7
ajuste
datos
tiempo
tamano
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
2500
5000
7500
10000
12500
15000
17500
20000

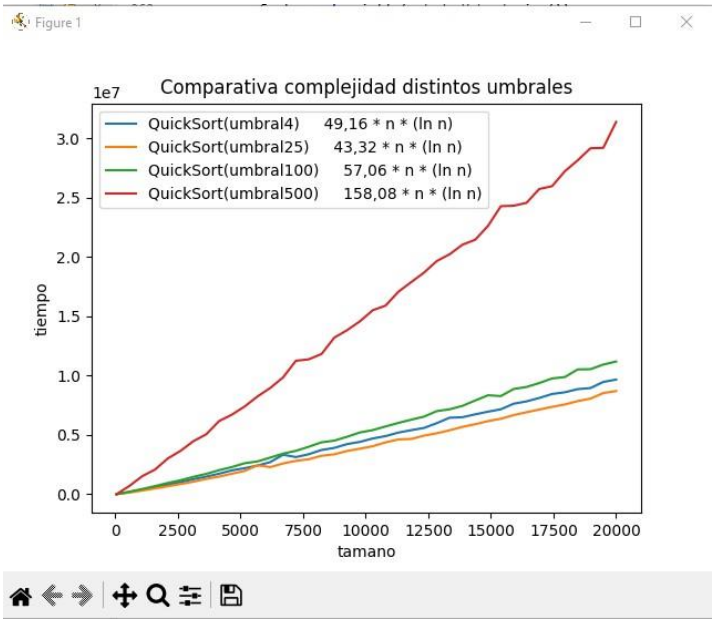


Figure 1
Comparativa complejidad distintos umbrales
1e7
QuickSort(umbral4) 49,16 * n * (ln n)
QuickSort(umbral25) 43,32 * n * (ln n)
QuickSort(umbral100) 57,06 * n * (ln n)
QuickSort(umbral500) 158,08 * n * (ln n)
tiempo
tamano
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
2500
5000
7500
10000
12500
15000
17500
20000

3. EJERCICIO 3

3.1. CÓDIGO EJERCICIO

```java
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()==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

```java
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<Pair<BinaryTree<Character>, Character>> datosBinarios =
                Files2.streamFromFile(rutaFicheroBinarios).map(linea -> {
                    String[] partes = linea.split("#");
                    BinaryTree<Character> tree = BinaryTree.parse(partes[0], s -> s.charAt(0));
                    Character ch = partes[1].charAt(0);
                    return Pair.of(tree, ch);
                    }
                ).toList();

        List<Pair<Tree<Character>, Character>> datosNarios =
                Files2.streamFromFile(rutaFicheroNarios).map(linea -> {
                    String[] partes = linea.split("#");
                    Tree<Character> tree = Tree.parse(partes[0], s -> s.charAt(0));
                    Character ch = partes[1].charAt(0);
                    return Pair.of(tree, ch);
                    }
                ).toList();
```

```java
        // TEST
        System.out.println("* TEST EJERCICIO 3*");

        System.out.println("\n*Arboles Binarios*");
        for(Pair<BinaryTree<Character>, Character> par: datosBinarios) {
            BinaryTree<Character> tree = par.first();
            Character ch = par.second();
            System.out.println("Arbol: " + tree + "\tCaracter: "+ ch +
                    "\t[" + Ejercicio3.fEjercicio3Binary(tree, ch) + "]");
        }

        System.out.println("\n*Arboles n-arios*");
        for(Pair<Tree<Character>, Character> par: datosNarios) {
            Tree<Character> tree = par.first();
            Character ch = par.second();
            System.out.println("Arbol: " + tree + "\tCaracter: "+ ch +
                    "\t[" + Ejercicio3.fEjercicio3Nary(tree, ch) + "]");
        }

    }
```

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, ABDFG, AIJ, AIK]]
Arbol: A(B(C,D(E,F(G,H))),I(J,K))       Caracter: C     [[ABDE, ABDFG, ABDFH, 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, ABDFG, ABDFH, ABDJ, ABK]]
```

4. EJERCICIO 4

4.1. CÓDIGO EJERCICIO

```java
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

```java
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
```