PRACTICA INDIVIDUAL 2

MEMORIA 1: Jose Luis Navarro (josnavlop4)

EJERCICIO 1:

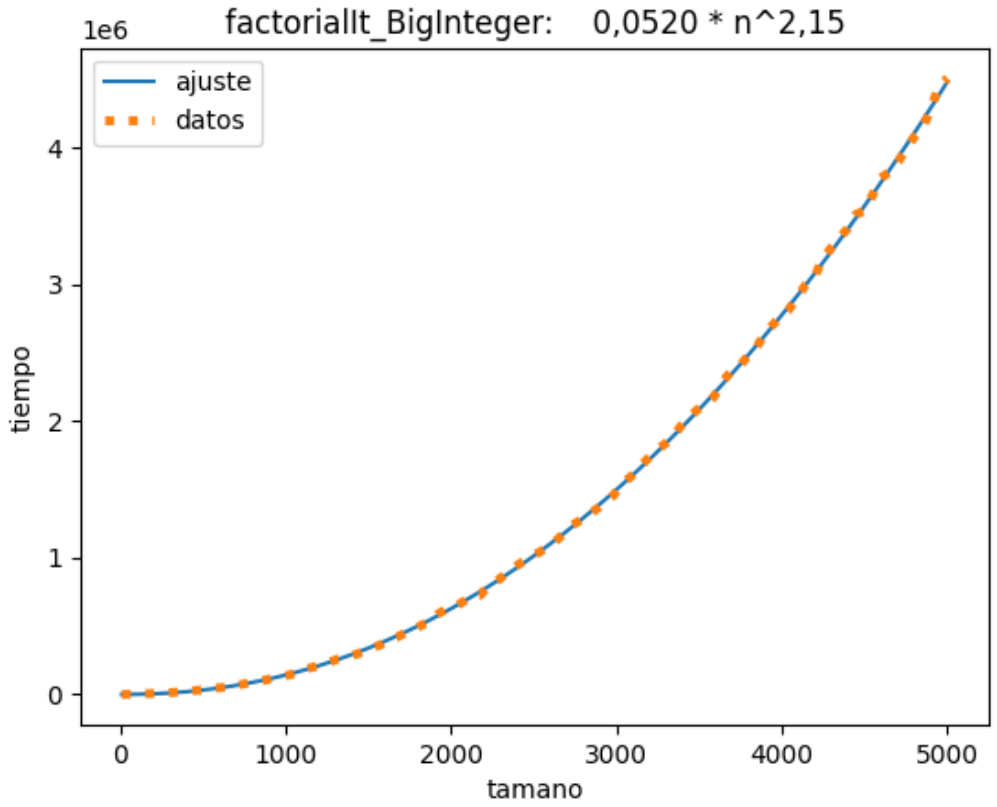
- **DOUBLE**

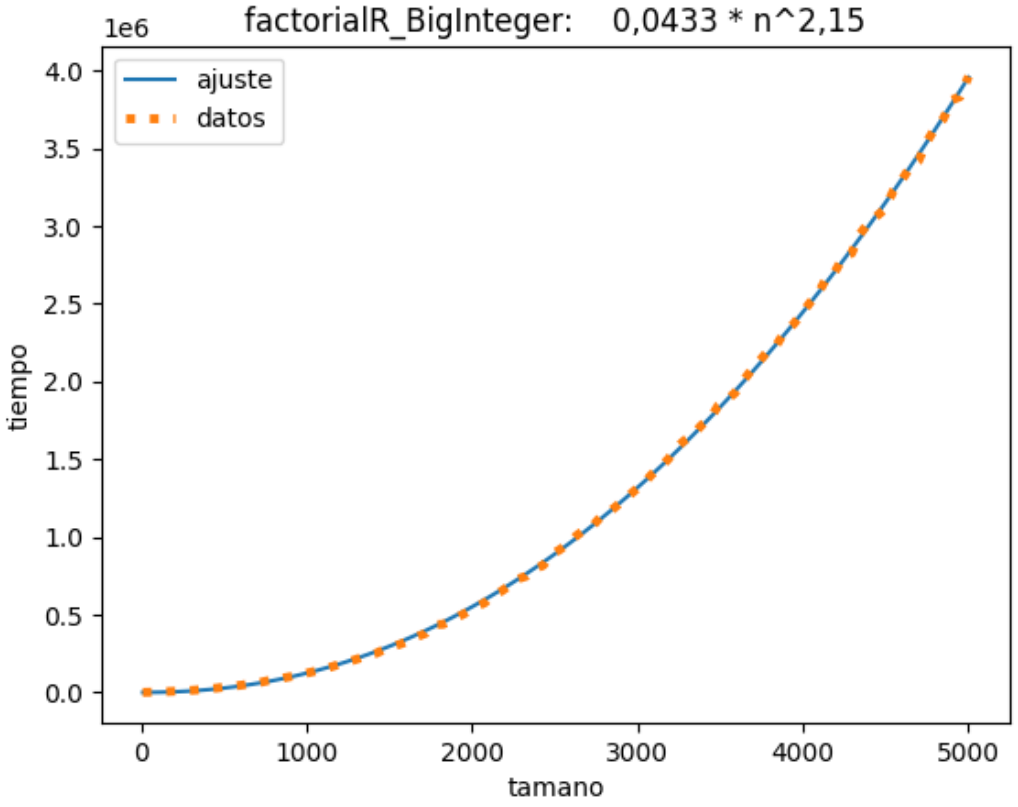
```java
public static Double factorialR_Double(int n) {
    Double r;
    if (n == 0) {
        r = 1.0;
    } else {
        r = factorialR_Double(n - 1) * n;
    }
    return r;
}

public static Double factorialIt_Double(int n) {
    Double a = 1.0;
    while (n != 0) {
        a = a * n;
        n = n - 1;
    }
    return a;
}
```

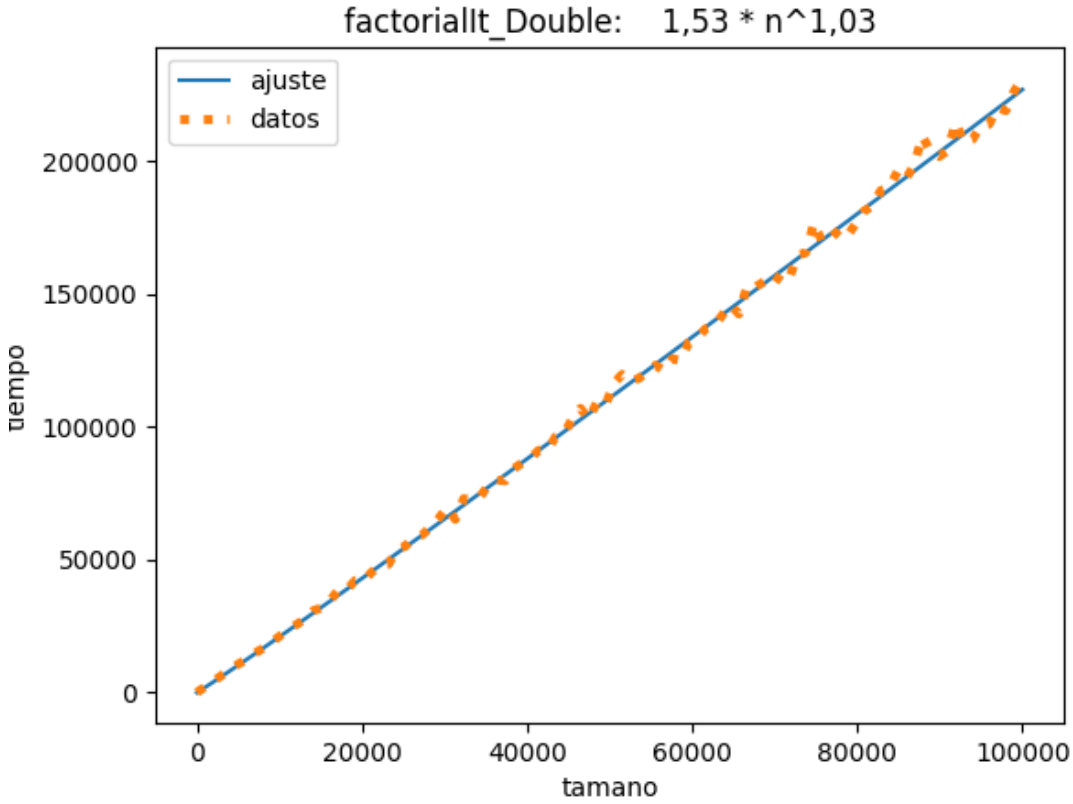
- **BIGINTEGER**

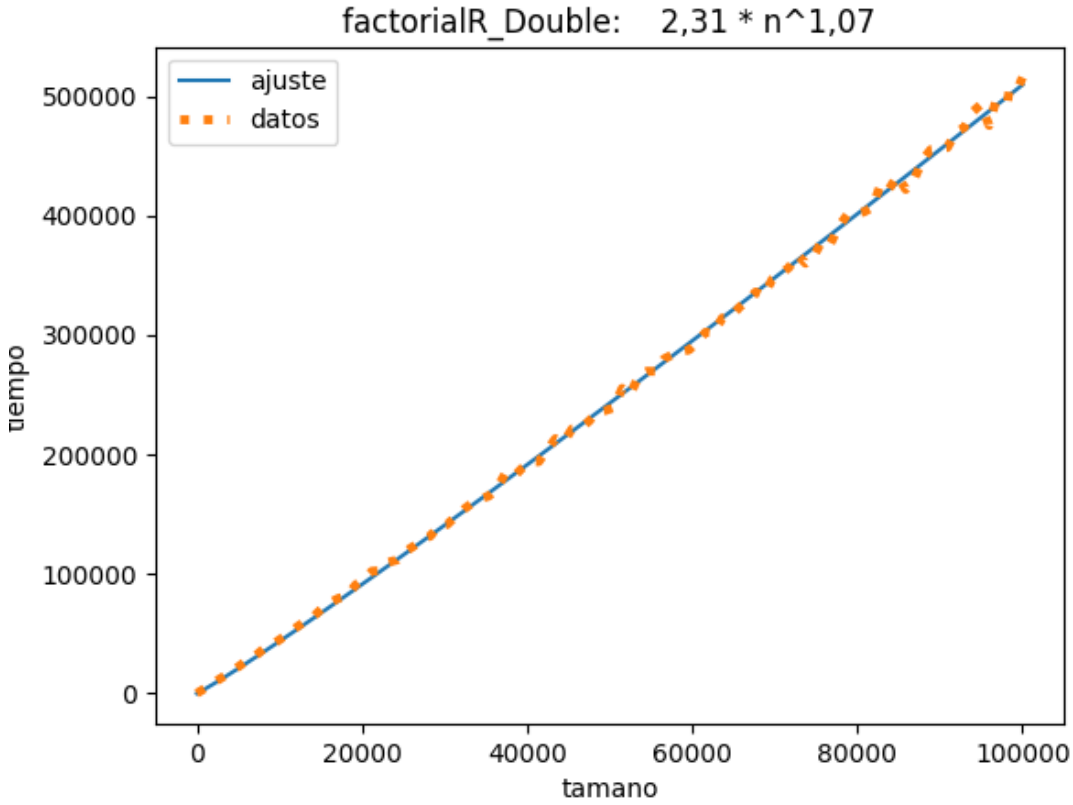
```java
public static BigInteger factorialR_BigInteger(int n) {
    BigInteger r;
    if (n == 0) {
        r = BigInteger.ONE;
    } else {
        r = factorialR_BigInteger(n - 1).multiply(BigInteger.valueOf(n));
    }
    return r;
}

public static BigInteger factorialIt_BigInteger(int n) {
    BigInteger a = BigInteger.ONE;
    while (n != 0) {
        a= a.multiply(BigInteger.valueOf(n));
        n = n - 1;
    }
    return a;
}
```

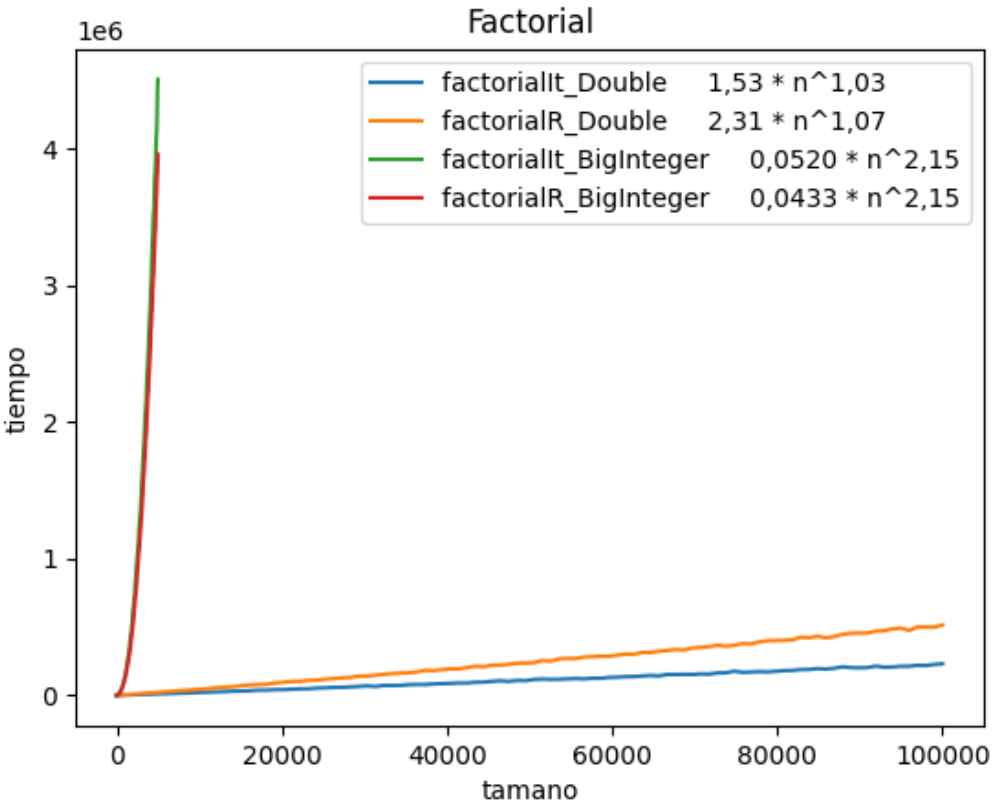
RESULTADOS


factorialIt_BigInteger: 0,0520 * n^2,15
1e6
ajuste
datos
tiempo
tamano
0
1
2
3
4
0
1000
2000
3000
4000
5000


factorialR_BigInteger: 0,0433 * n^2,15
1e6
ajuste
datos
tiempo
tamano
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
1000
2000
3000
4000
5000


factorialIt_Double: 1,53 * n^1,03
ajuste
datos
200000
150000
100000
50000
0
tiempo
0
20000
40000
60000
80000
100000
tamano


factorialR_Double: 2,31 * n^1,07
ajuste
datos
500000
400000
300000
200000
100000
0
tiempo
0
20000
40000
60000
80000
100000
tamano


Factorial
1e6
factorialIt_Double 1,53 * n^1,03
factorialR_Double 2,31 * n^1,07
factorialIt_BigInteger 0,0520 * n^2,15
factorialR_BigInteger 0,0433 * n^2,15
tiempo
tamano
0
1
2
3
4
0
20000
40000
60000
80000
100000

TEST

```
public static void main(String[] args) {
    generaFicherosTiempoEjecucion();
    generaFicherosTiempoEjecucion2();
    muestraGraficas();

}

private static Integer nMin = 10; // n mínimo para el cálculo de potencia
private static Integer nMax = 100000; // n máximo para el cálculo de potencia
private static Integer nMaxBigInt = 5000;
private static Integer numSizes = 100; // número de problemas (número de potencias distintas a calcular)
private static Integer numMediciones = 10; // número de mediciones de tiempo de cada caso (número de experimentos)
private static Integer numIter = 50; // número de iteraciones para cada medición de tiempo
private static Integer numIterWarmup = 1000; // número de iteraciones para warmup

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

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

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

public static void generaFicherosTiempoEjecucion2() {
    for (int i = 0; i < metodosBigInteger.size(); i++) {
        String ficheroSalida = String.format("ficheros/Tiempos%s.csv", metodosBigInteger.get(i).third());
        testTiemposEjecucion2(nMin, nMaxBigInt, metodosBigInteger.get(i).first(), ficheroSalida);
    }
}

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);

        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<>();

    for (int i = 0; i < metodosDouble.size(); i++) {
        String ficheroSalida = String.format("ficheros/Tiempos%s.csv", metodosDouble.get(i).third());
        ficherosSalida.add(ficheroSalida);
        String label = metodosDouble.get(i).third();
        System.out.println(label);
        TipoAjuste tipoAjuste = metodosDouble.get(i).second();
        GraficosAjuste.show(ficheroSalida, tipoAjuste, label);
        Pair<Function<Double, Double>, String> parCurve = GraficosAjuste.fitCurve(DataCurveFitting.points(ficheroSalida), tipoAjuste);
        String ajusteString = parCurve.second();
        labels.add(String.format("%s     %s", label, ajusteString));
    }
    GraficosAjuste.showCombined("Factorial", ficherosSalida, labels);

    for (int i = 0; i < metodosBigInteger.size(); i++) {
        String ficheroSalida = String.format("ficheros/Tiempos%s.csv", metodosBigInteger.get(i).third());
        ficherosSalida.add(ficheroSalida);
        String label = metodosBigInteger.get(i).third();
        System.out.println(label);
        TipoAjuste tipoAjuste = metodosBigInteger.get(i).second();
        GraficosAjuste.show(ficheroSalida, tipoAjuste, label);
        Pair<Function<Double, Double>, String> parCurve = GraficosAjuste.fitCurve(DataCurveFitting.points(ficheroSalida), tipoAjuste);
        String ajusteString = parCurve.second();
        labels.add(String.format("%s     %s", label, ajusteString));
    }
    GraficosAjuste.showCombined("Factorial", ficherosSalida, labels);

}
```

```java
@SuppressWarnings("unchecked")
public static void testTiemposEjecucion(Integer nMin, Integer nMax, Function<Integer, Double> funcion,
        String ficheroTiempos) {
        Map<Problema, Double> tiempos = new HashMap<Problema, Double>();
        Integer nMed = numMediciones;
        for (int iter = 0; iter < nMed; iter++) {
        for (int i = 0; i < numSizes; i++) {
        Double r = Double.valueOf(nMax-nMin) / (numSizes-1);
        Integer tam = (Integer.MAX_VALUE/nMax > i) ?
        nMin + i*(nMax - nMin) / (numSizes - 1):
        nMin + (int) (r * i);
        Problema p = Problema.of(tam);
        warmup(funcion, 10);
        Integer nIter = numIter;
        Double[] res = new Double[nIter];
        Long t0 = System.nanoTime();
        for (int z = 0; z < nIter; z++) {
        res[z] = (Double) funcion.apply(tam);
        }
        Long t1 = System.nanoTime();
        actualizaTiempos(tiempos, p, Double.valueOf(t1 - t0) / nIter);
        }
        }

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

        }

public static void testTiemposEjecucion2(Integer nMin, Integer nMax, Function<Integer, BigInteger> funcion,
        String ficheroTiempos) {
    Map<Problema, Double> tiempos = new HashMap<Problema, Double>();
    Integer nMed = numMediciones;
    for (int iter = 0; iter < nMed; iter++) {
        for (int i = 0; i < numSizes; i++) {
            Double r = Double.valueOf(nMaxBigInt-nMin) / (numSizes-1);
            Integer tam = (Integer.MAX_VALUE/nMaxBigInt > i) ?
                        nMin + i*(nMaxBigInt - nMin) / (numSizes - 1):
                        nMin + (int) (r * i);
            Problema p = Problema.of(tam);
            warmup2(funcion, 10);
            Integer nIter = numIter;
            BigInteger[] res = new BigInteger[nIter];
            Long t0 = System.nanoTime();
            for (int z = 0; z < nIter; z++) {
                res[z] = funcion.apply(tam);
            }
            Long t1 = System.nanoTime();
            actualizaTiempos(tiempos, p, Double.valueOf(t1 - t0) / nIter);
        }
    }

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

}

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

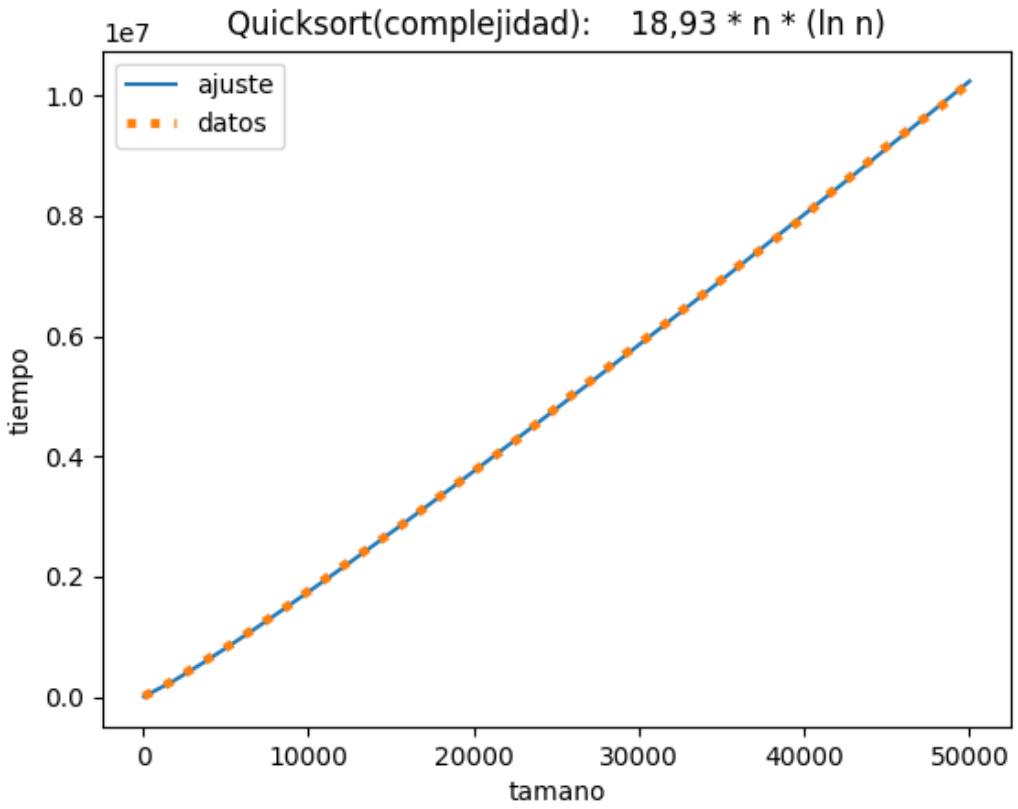
```java
private static void warmup(Function<Integer, Double> fact, Integer n) {
    for (int i = 0; i < numIterWarmup; i++) {
        fact.apply(n);
    }
}

private static void warmup2(Function<Integer, BigInteger> fact, Integer n) {
    for (int i = 0; i < numIterWarmup; i++) {
        fact.apply(n);
    }
}

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);
    }
}
```

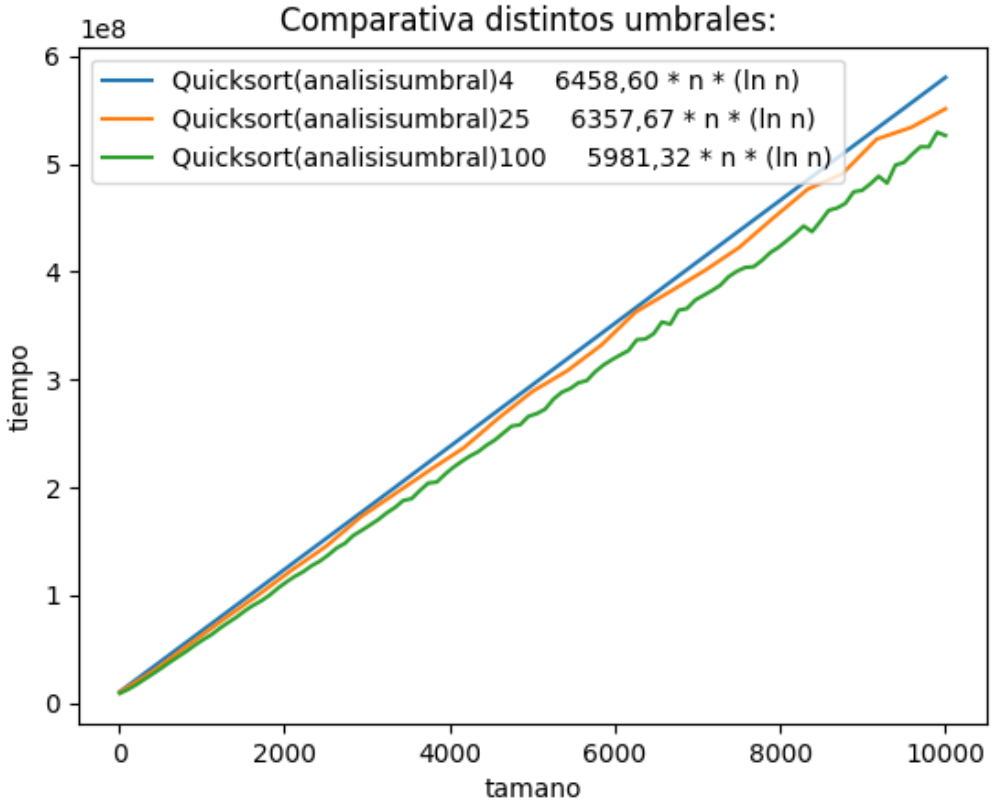
EJERCICIO 2

QUICKSORT

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

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

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

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

public static <E> IntPair banderaHolandesa(List<E> ls, E pivote, Integer i, Integer j,  Comparator<? super E> cmp){
    Integer a=i, b=i, c=j;
    while (c-b>0) {
        E elem =  ls.get(b);
        if (cmp.compare(elem, pivote)<0) {
            List2.intercambia(ls,a,b);
            a++;
            b++;
        } else if (cmp.compare(elem, pivote)>0) {
            List2.intercambia(ls,b,c-1);
            c--;
        } else {
            b++;
        }
    }
    return IntPair.of(a,b);
}
```

RESULTADOS


Quicksort(complejidad): 18,93 * n * (ln n)
1e7
ajuste
datos
tiempo
tamano
0.0
0.2
0.4
0.6
0.8
1.0
0
10000
20000
30000
40000
50000


Comparativa distintos umbrales:
1e8
Quicksort(analisisumbral)4 6458,60 * n * (ln n)
Quicksort(analisisumbral)25 6357,67 * n * (ln n)
Quicksort(analisisumbral)100 5981,32 * n * (ln n)
tiempo
tamano
0
1
2
3
4
5
6
0
2000
4000
6000
8000
10000

TEST

```
public static void main(String[] args) {
    generaFicheroListasEnteros(ficheroListaEntrada);
    generaFicherosTiempoEjecucionAll();
    muestraGraficasAll();
}

private static Integer numMediciones = 10; // número de mediciones de tiempo de cada caso (número de experimentos)
private static Integer numIter = 50; // número de iteraciones para cada medición de tiempo
private static Integer numIterWarmup = 50; // número de iteraciones para warmup

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

private static Integer numUmbrales = 30; // número de umbrales para análisis de distintos umbrales
private static Integer nMinUmbral = 10; // n mínimo umbral
private static Integer nMaxUmbral = 10000; // n máximo umbral

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

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

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

private static List<Trio<BiConsumer<List<Integer>, Integer>, TipoAjuste, String>> metodosUmbral = List.of(
        Trio.of(Ejercicio2::sort, TipoAjuste.NLOGN_0, "Quicksort(analisisumbral)4"),// Análisis del tamaño umbral con una única lista de tamaño sizeMax
        Trio.of(Ejercicio2::sort, TipoAjuste.NLOGN_0, "Quicksort(analisisumbral)25"),
        Trio.of(Ejercicio2::sort, TipoAjuste.NLOGN_0, "Quicksort(analisisumbral)100")
);

public static void muestraGraficasAll() {
    muestraGraficas(metodosComplejidad);
    muestraGraficas(metodosUmbral);
}

public static void generaFicherosTiempoEjecucionAll() {
    generaFicherosTiempoEjecucion(metodosComplejidad,30);
    generaFicherosTiempoEjecucion(metodosUmbral,4);
    generaFicherosTiempoEjecucion(metodosUmbral,25);
    generaFicherosTiempoEjecucion(metodosUmbral,100);
}

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

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

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

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

    List<String> ficherosSalida = new ArrayList<>();
    List<String> labels = new ArrayList<>();

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

        String ficheroSalida = String.format("ficheros/Tiempos%s.csv",
                metodos.get(i).third());
        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 gráfica combinada
        Pair<Function<Double, Double>, String> parCurve = GraficosAjuste.fitCurve(
                DataCurveFitting.points(ficheroSalida), tipoAjuste);
        String ajusteString = parCurve.second();
        labels.add(String.format("%s     %s", label, ajusteString));


    }

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

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

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

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

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

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

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

        }

    }

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

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

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

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

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

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

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

                Integer nIter = numIter;
                Long t0 = System.nanoTime();
                for (int z=0; z<nIter; z++) {
                    funcion.accept(le, umbral);
                }
                Long t1 = System.nanoTime();
                actualizaTiempos(tiempos, p, Double.valueOf(t1-t0)/nIter);
        }
    }
    tiemposMedios = tiempos.entrySet().stream()
            .collect(Collectors.groupingBy(x->x.getKey().tam(),
                        Collectors.averagingDouble(x->x.getValue())
                        )
                    );

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

```java
private static void warmup(BiConsumer<List<Integer>, Integer> funcion, List<Integer> le, int n) {
    for (int i=0; i<numIterWarmup; i++) {
        funcion.accept(le,n);
    }
}

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

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

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

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

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);
    }
}
```

EJERCICIO 3

```
public static List<String> solucionBinaria(BinaryTree<Character> tree, Character chars) {
    List<String> res = new ArrayList<>();
    return solucionBinaria(tree,chars,res,"");
}

private static List<String> solucionBinaria(BinaryTree<Character> tree, Character chars, List<String> res, String cad) {
    switch(tree) {
    case BEmpty<Character> t->{
        break;
    }
    case BLeaf<Character> t->{
        cad = cad + t.label();
        if (!cad.contains(chars.toString())) {
            res.add(cad);
        }
        break;
    }
    case BTree<Character> t->{
        solucionBinaria(t.left(),chars,res,cad + t.label());
        solucionBinaria(t.right(),chars,res,cad + t.label());
        break;
    }
    };
    return res;
}

public static List<String> solucionNaria(Tree<Character> tree, Character chars) {
    List<String> res = new ArrayList<>();
    return solucionNaria(tree,chars,res,"");
}
private static List<String> solucionNaria(Tree<Character> tree, Character chars,List<String> res, String cad){
    return switch(tree) {
    case TEmpty<Character> t->res;
    case TLeaf<Character> t->{
        cad = cad + t.label();
        if (!cad.contains(chars.toString())) {
            res.add(cad);
        }
        yield res;
    }
    case TNary<Character> t->{
        String cad1 = cad+t.label();
        t.elements().forEach(tc ->solucionNaria(tc,chars,res, cad1));
        yield res;
    }
    };
}
```

RESULTADOS

```
************** EJERCICIO 3 (Binario) ******************
-----------------------------------------------------
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]]
-----------------------------------------------------
************** EJERCICIO 3 (Nario) ******************
-----------------------------------------------------
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]]
-----------------------------------------------------
```

TEST

```
public static void testsEjercicio3() {
    testsEjercicio3Binario();
    testsEjercicio3Nario();
}

public static void testsEjercicio3Binario() {
    String file = "ficheros/Ejercicio3DatosEntradaBinario.txt";

    List<Pair<BinaryTree<Character>, Character>> inputs = Files2.streamFromFile(file).map(linea -> {
        String[] aux = linea.split("#");
        Preconditions.checkArgument(aux.length == 2);
        return Pair.of(BinaryTree.parse(aux[0], s -> s.charAt(0)), aux[1].charAt(0));
    }).toList();


    System.out.println("************** EJERCICIO 3 (Binario) ******************");
    System.out.println("-------------------------------------------------------");

    inputs.stream().forEach(par->{
        BinaryTree<Character> tree = par.first();
        Character chars = par.second();

        System.out.println("Arbol:"+tree+"\t Caracter:"+ chars + "\t["+
                Ejercicio3.solucionBinaria(tree,chars)+"]");

    System.out.println("-------------------------------------------------------");
    });
}
public static void testsEjercicio3Nario() {
    String file = "ficheros/Ejercicio3DatosEntradaNario.txt";

    List<Pair<Tree<Character>, Character>> inputs = Files2.streamFromFile(file).map(linea -> {
        String[] aux = linea.split("#");
        Preconditions.checkArgument(aux.length == 2);
        return Pair.of(Tree.parse(aux[0], s -> s.charAt(0)), aux[1].charAt(0));
    }).toList();

    System.out.println("************** EJERCICIO 3 (Nario) ******************");
    System.out.println("-------------------------------------------------------");

    inputs.stream().forEach(par->{
        Tree<Character> tree = par.first();
        Character chars = par.second();

        System.out.println("Arbol: "+tree+"\t Caracter: "+ chars + "\t["+
                Ejercicio3.solucionNaria(tree,chars)+"]");

    System.out.println("-------------------------------------------------------");
    });
}
```

EJERCICIO 4

- **SOLUCION BINARIA**

```
private static Long nVocales(String s) {
    Long b = 0L;
    List<Character> ls = new ArrayList<>();
    ls.add('a');ls.add('e');ls.add('i');ls.add('o');ls.add('u');
    for (int i = 0; i<s.length();i++) {
        Character a = s.charAt(i);
        if (ls.contains(a)) {
            b++;
        }
    }
    return b;
}

public static Boolean solucionBinaria(BinaryTree<String> tree) {
    return solucionBinariaRec(tree).second();
}

private static Pair<Long, Boolean> solucionBinariaRec(BinaryTree<String> tree) {
    return switch(tree) {
        case BEmpty<String> t->Pair.of(0L, true);
        case BLeaf<String> t->Pair.of(nVocales(t.label()),true);
        case BTree<String> t->{
            Pair<Long, Boolean> r = solucionBinariaRec(t.right());
            Pair<Long, Boolean> l = solucionBinariaRec(t.left());
            yield Pair.of(l.first() + r.first(), l.second() && r.second() && l.first()==r.first());
        }
    };
}
```

- **SOLUCION N-ARIA**

```
public static Boolean solucionNaria(Tree<String> tree) {
    return solucionNariaRec(tree).second();
}

private static Pair<Long, Boolean> solucionNariaRec(Tree<String> tree){
    return switch(tree) {
        case TEmpty<String> t->Pair.of(0L, true);
        case TLeaf<String> t->Pair.of(nVocales(t.label()),true);
        case TNary<String> t->{
            yield Pair.of(t.elements().stream()
                          .map(x->solucionNariaRec(x))
                          .map(x->x.first())
                          .reduce((x,y)->x+y)
                          .get(),
                         t.elements().stream()
                          .map(x->solucionNariaRec(x))
                          .reduce((x,y)->Pair.of(x.first(),x.first()==y.first()&&x.second()&&y.second()))
                          .get()
                          .second());
    }
    };
}
```

RESULTADO

```
*************************      EJERCICIO 4 (Binario)       **************************
--------------------------------------------------------------------------------------
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
--------------------------------------------------------------------------------------
*************************       EJERCICIO 4 (Nario)       **************************
--------------------------------------------------------------------------------------
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
--------------------------------------------------------------------------------------
```

TEST

```
public static void testsEjercicio4() {
    testsEjercicio4Binario();
    testsEjercicio4Nario();
}

public static void testsEjercicio4Binario() {
    String file = "ficheros/Ejercicio4DatosEntradaBinario.txt";

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

    System.out.println("*************************      EJERCICIO 4 (Binario)       **************************");
    System.out.println("--------------------------------------------------------------------------------------");

    inputs.stream().forEach(x -> System.out.println(x + ":" + Ejercicio4.solucionBinaria(x)));
    System.out.println("--------------------------------------------------------------------------------------");
}

public static void testsEjercicio4Nario() {
    String file = "ficheros/Ejercicio4DatosEntradaNario.txt";

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

    System.out.println("*************************      EJERCICIO 4 (Nario)       **************************");
    System.out.println("--------------------------------------------------------------------------------------");

    inputs.stream().forEach(x -> System.out.println(x+ ":" + Ejercicio4.solucionNaria(x)));
    System.out.println("--------------------------------------------------------------------------------------");
}
```