

Memoria Practica 2

Jesús carrascosa Carro
(jescarcar5)
jescarcar5@alum.us.es

Noviembre 2022

Índice

1. Introducción	1
2. Ejercicio 1:	2
2.1. Código:	2
2.2. Test:	3
2.3. Imágenes	8
3. Ejercicio 2 Quicksort de Miguel Toro	12
3.1. Código:	12
3.2. Test:	14
3.3. Imágenes	20
4. Ejercicio 2 QuickSort con InsertionSort	21
4.1. Código	21
4.2. Test:	23
4.3. Imágenes:	29
5. Ejercicio 3:	30
5.1. Código	30
5.2. Tests	33
5.3. Imágenes	35
6. Ejercicio 4:	36
6.1. Código:	36
6.2. Test:	39
6.3. Imágenes:	40

1. Introducción

Resumen

Este documento es la memoria de la 2ª práctica individual de la asignatura ADDA. El código que se muestra en este documento, también se encuentra disponible en el repositorio privado de github:

https://github.com/yisuscc/PI2_escargar5

Puesto que dicho repositorio tiene la visibilidad en privado, si precisan acceder a él, ruego que me contacten por email que figura al inicio del documento

2. Ejercicio 1:

2.1. Código:

```
// version double recursiva
public static Double facRdouble(Integer n) {
    Double r = null;
    if (n == 0 || n == 1) {
        r = (double) 1;
    } else {
        r = n * facRdouble(n - 1);
    }
    return r;
}

// version double iterativa

public static Double facIterDouble(Integer n) {
    Double ac = (double) 1;
    for (Double i = (double) 1; i <= n; i++) {
        ac = ac * i;
    }
    return ac;
}

// version bigInt recursiv
public static BigInteger facRBigInteger(Integer n) {
    BigInteger r = null;
    if (n == 1) {
        r = BigInteger.ONE;
    } else {
        r = facRBigInteger(n - 1).multiply(BigInteger.valueOf(n));
    }
    return r;
}
```

```

public static BigInteger facIterBigInteger(Integer N) {
    BigInteger f = new BigInteger("1");
    for (int i = 2; i <= N; i++)
        f = f.multiply(BigInteger.valueOf(i));
    return f;
}

public static void main(String[] args) {
    System.out.println("Recursivo double" + facRdouble(14));
    System.out.println("Iterativo double" + facIterDouble(14));
    System.out.println("Iterativo Big Integer" + facIterBigInteger(14));
    System.out.println("Recursivo Big Integer" + facRBigInteger(14));
}

```

2.2. Test:

```

package tests;

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

import ejercicios.Ejercicio1;
import tests.TestEjemplo2.Problema;
import tests.TestEjemplo2.TResultD;
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 TestEj1 {

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

    private static Integer nMin = 2; // n mínimo
    private static Integer nMaxBigInt = 10000; // n máximo para el fibonacci recursivo sin memoria
    private static Integer nMaxDouble = 10000; // n máximo para el fibonacci no exponencial

```

```

private static Integer numSizes = 30; // número de problemas
private static Integer numMediciones = 10; // 10; // número de mediciones de tiempo de cada
// experimentos)
// para exponencial se puede reducir
private static Integer numIter = 50; // 50; // número de iteraciones para cada medición de t
// para exponencial se puede reducir
private static Integer numIterWarmup = 1000; // número de iteraciones para warmup
// cuasi boilerplate
private static List<Trio<Function<Integer, Number>, TipoAjuste, String>> metodosDeDouble = L
Trio.of(Ejercicio1::facIterDouble, TipoAjuste.POWERANB, "Iterativo double (lineal)"),
Trio.of(Ejercicio1::facRdouble, TipoAjuste.POWERANB, "Recursivo double (lineal)"));
private static List<Trio<Function<Integer, Number>, TipoAjuste, String>> metodosDeBigInteger
Trio.of(Ejercicio1::facIterBigInteger, TipoAjuste.EXP2_0, "Iterativo Big Integer(exponencial
Trio.of(Ejercicio1::facRBigInteger, TipoAjuste.EXP2_0, "Recursivo Big Integer(exponencial)");

private static Boolean esExponencial(TipoAjuste ta) {
Boolean r = false;
if (ta.equals(TipoAjuste.EXP) || ta.equals(TipoAjuste.EXP2) || ta.equals(TipoAjuste.EXP2_0))
r = true;
return r;
}

private static <E> void generaFicherosTiempoEjecucionMetodos(
List<Trio<Function<Integer, Number>, TipoAjuste, String>> metodos) {
// cuasi boilerplate

for (int i = 0; i < metodos.size(); i++) {
Trio<Function<Integer, Number>, TipoAjuste, String> tm = metodos.get(i); // TODO cambiarlo a
int numMax = esExponencial(tm.second()) ? nMaxBigInt : nMaxDouble;
Boolean flagExp = esExponencial(tm.second()) ? true : false; // TODO Acortar la función

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(metodosDeDouble);
generaFicherosTiempoEjecucionMetodos(metodosDeBigInteger);
}

public static <E> void muestraGraficasMetodos(List<Trio<Function<E, Number>, TipoAjuste, St

```

```

metodos,
List<String> ficherosSalida, List<String> labels) {
// boilerplate
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));
}
}

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

muestraGraficasMetodos(metodosDeBigInteger, ficherosSalida, labels);
muestraGraficasMetodos(metodosDeDouble, ficherosSalida, labels);

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

@SuppressWarnings("unchecked")
public static <E> void testTiemposEjecucion(Integer nMin, Integer nMax,
// cuasi boilerplate
Function<E, Number> funcion, 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(funcion, 10);
Integer nIter = flagExp ? 5 : numIter;
Number[] res = new Number[nIter];
Long t0 = System.nanoTime();
for (int z = 0; z < nIter; z++) {
    res[z] = funcion.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) {
// boilerplate
if (!tiempos.containsKey(p)) {
    tiempos.put(p, d);
} else if (tiempos.get(p) > d) {
    tiempos.put(p, d);
}
}

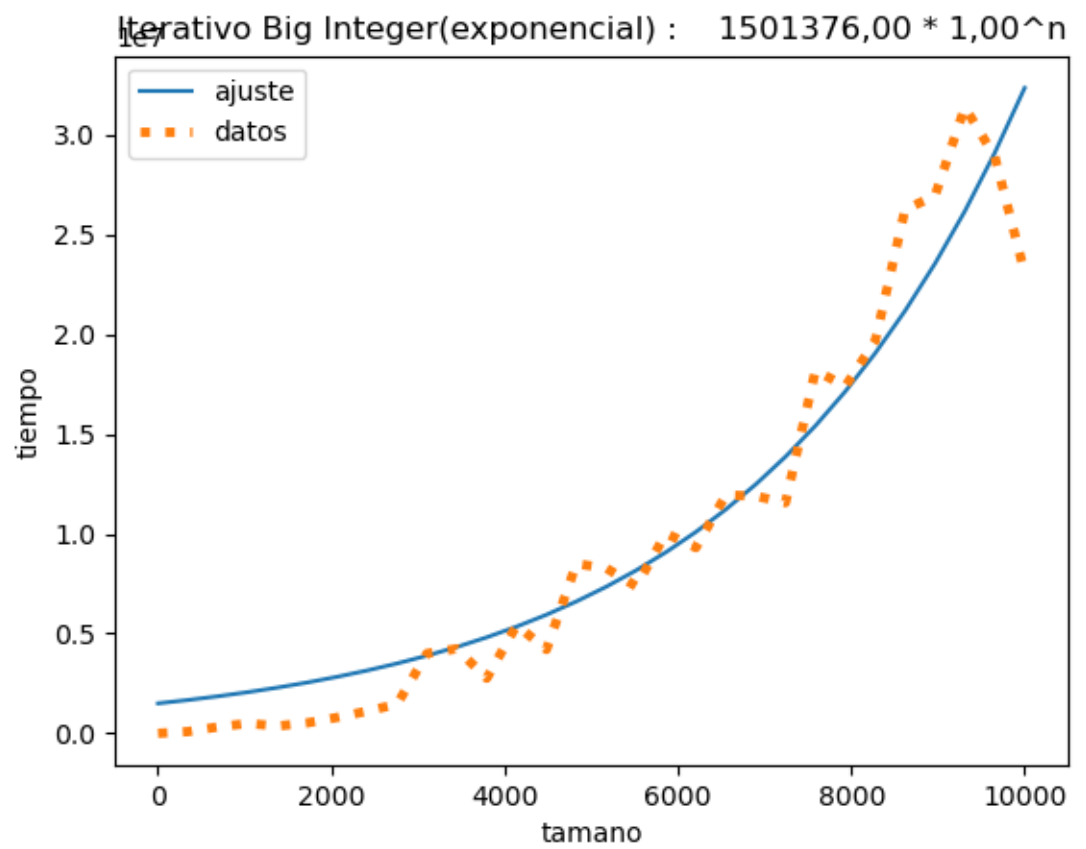
private static <E> BigInteger warmup(Function<E, Number> fun, Integer n) {
// boilerplate
BigInteger res = BigInteger.ZERO;
BigInteger z = BigInteger.ZERO;
for (int i = 0; i < numIterWarmup; i++) {
    if (fun.apply((E) n).equals(z))
        z.add(BigInteger.ONE);
}
res = z.equals(BigInteger.ONE) ? z.add(BigInteger.ONE) : z;
return res;
}

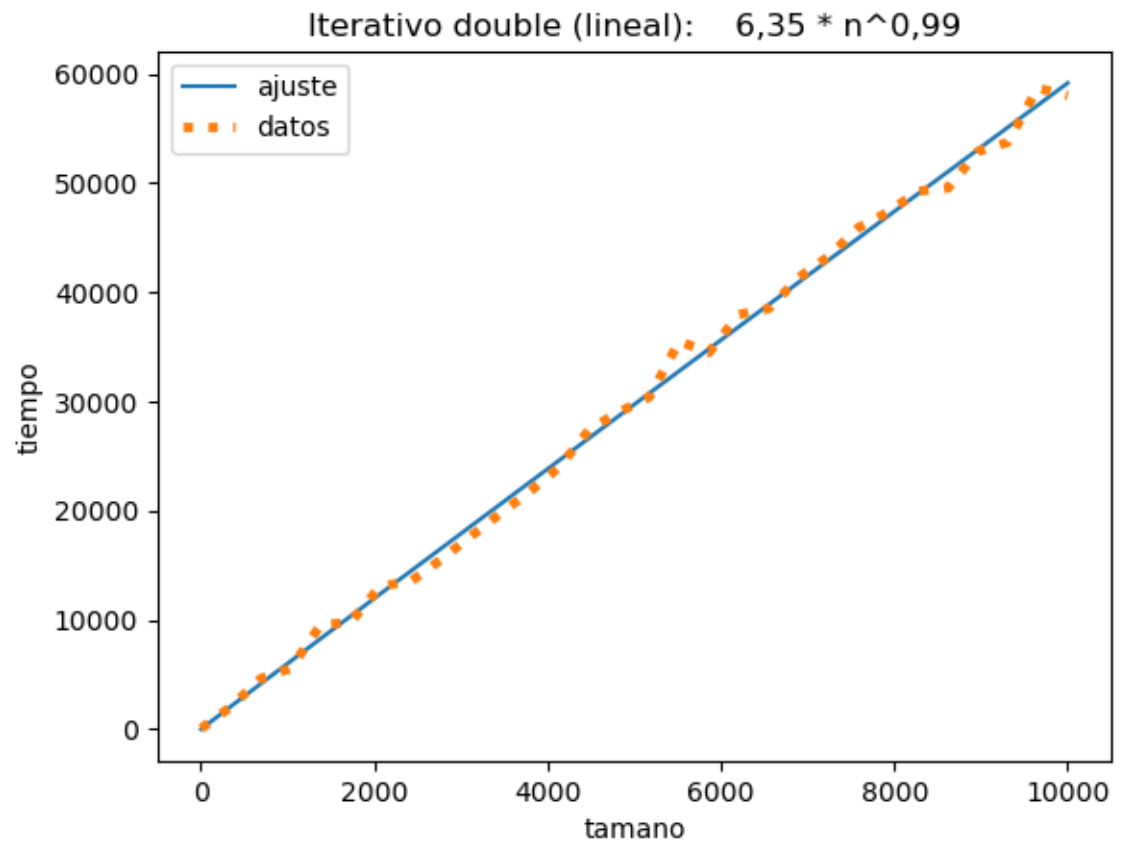
record TResultD(Integer tam, Double t) {
// boilerplate
public static TResultD of(Integer tam, Double t) {
return new TResultD(tam, t);
}
}

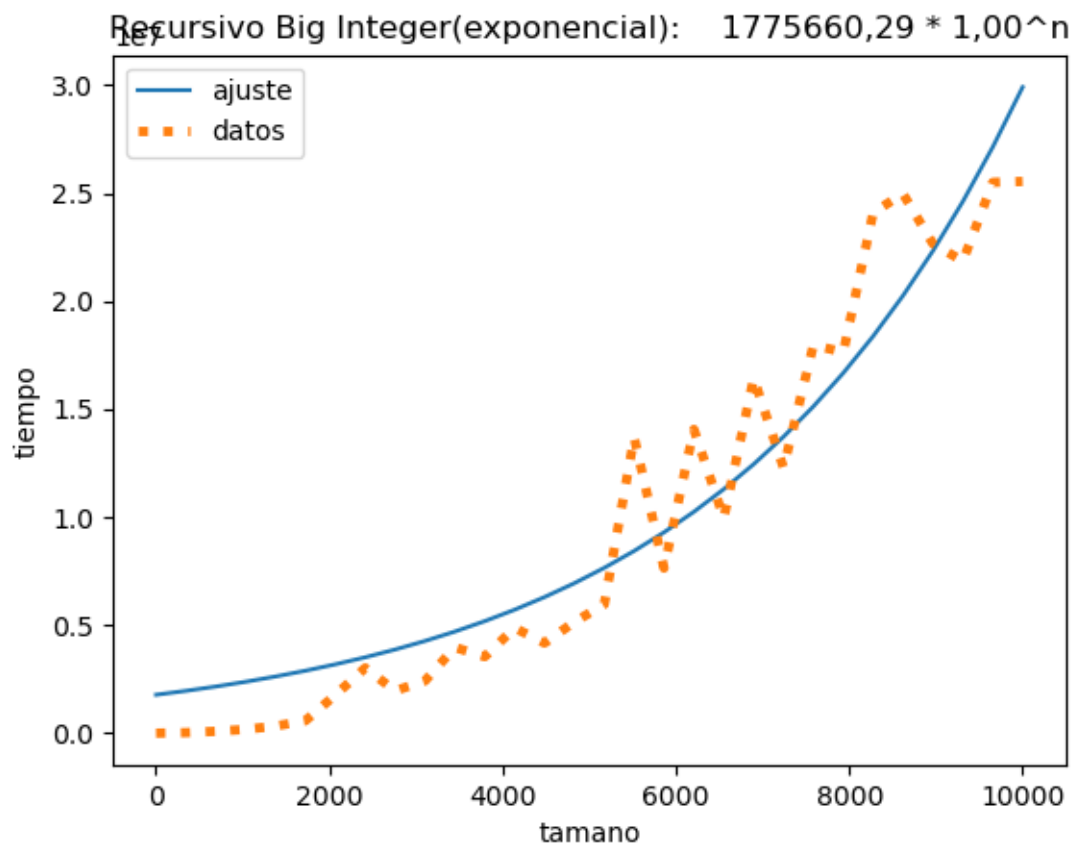
```

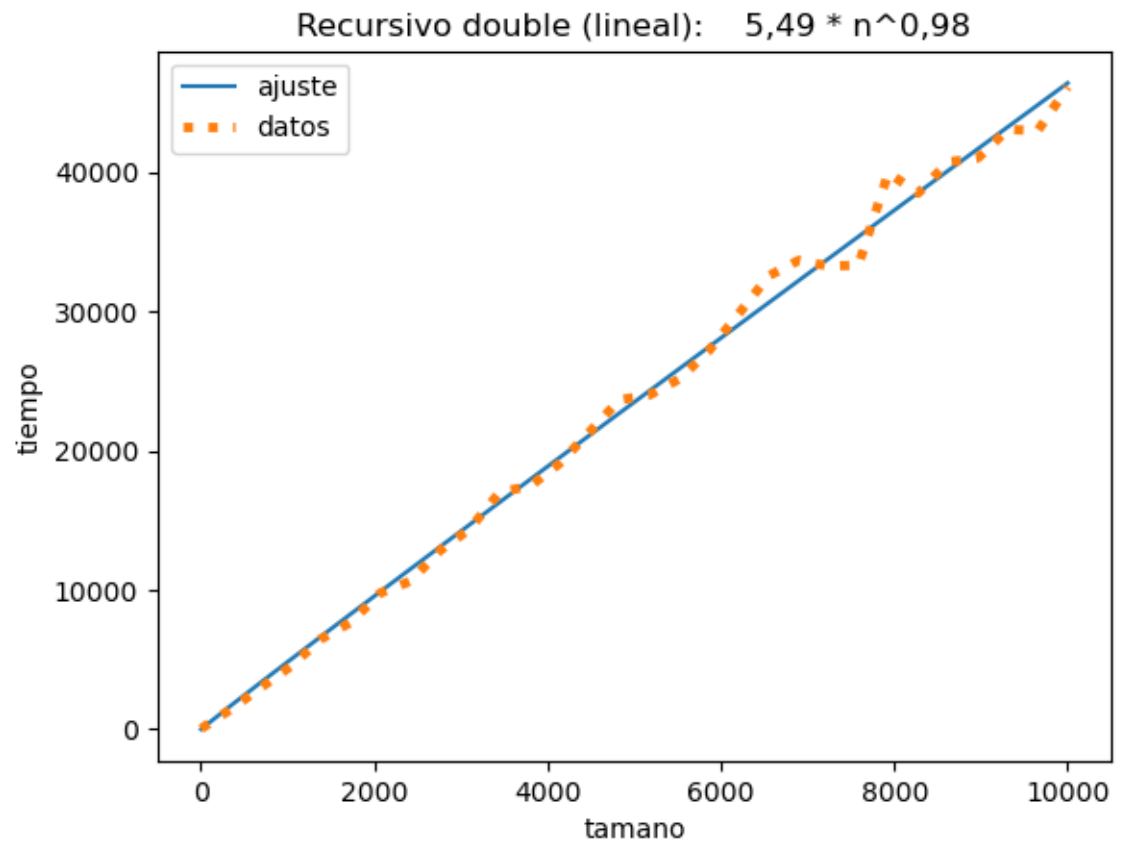
```
public String toString() {  
    // boilerplate  
    return String.format("%d,%.0f", tam, t);  
}  
}  
  
record Problema(Integer tam) {  
    // boilerplate  
    public static Problema of(Integer tam) {  
        return new Problema(tam);  
    }  
}  
}
```

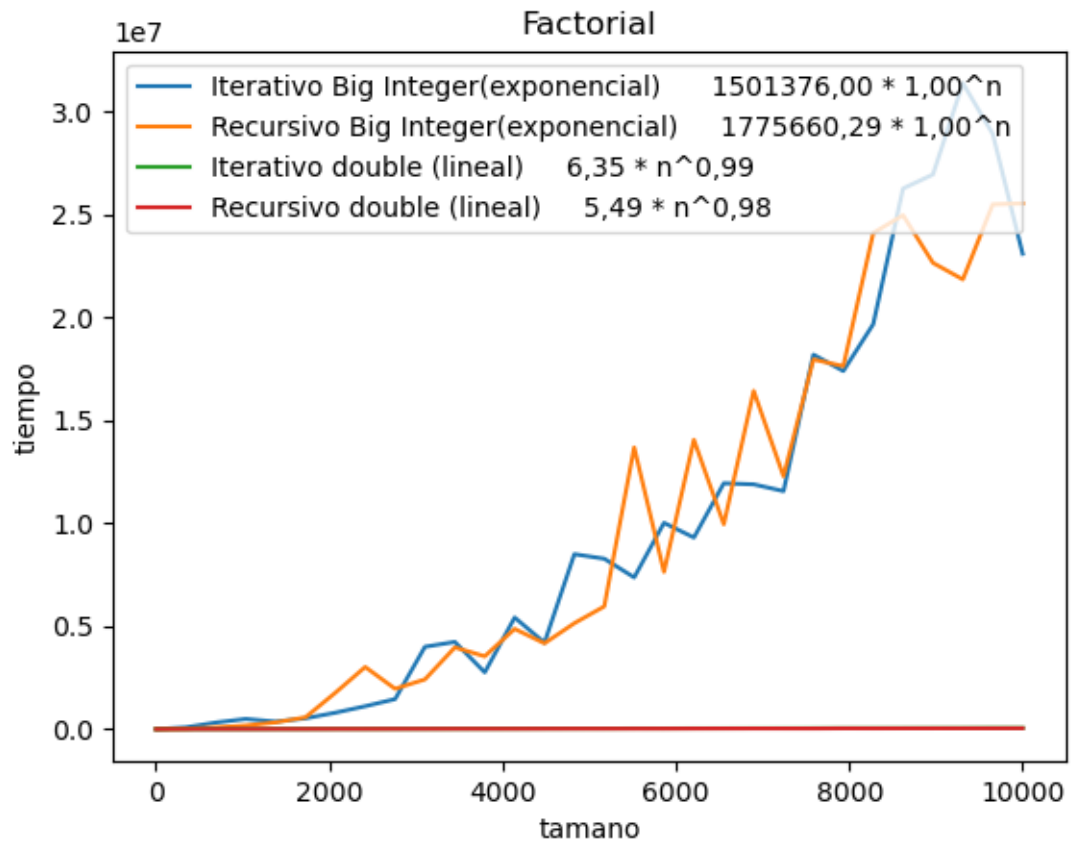
2.3. Imágenes











3. Ejercicio 2 Quicksort de Miguel Toro

3.1. Código:

```
package ejercicios;

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

import us.lsi.common.Files2;
import us.lsi.common.IntPair;
import us.lsi.common.List2;
import us.lsi.common.Preconditions;
import us.lsi.math.Math2;
```

```

public class Ejercicio2MiguelToro {
    public static void quickSortMT(List<Integer> ls, Integer um) {
        Comparator<Integer> ord = Comparator.naturalOrder();
        Integer i = 0;
        Integer j = ls.size();
        quickSort(ls, i, j, um, ord);
    }

    private static <E> void quickSort(List<E> lista, int i, int j, Integer um,
        Comparator<? super E> ord) {
        Preconditions.checkArgument(j >= i);
        if (j - i <= um) {
            ordenaBase(lista, i, j, ord);
        } else {
            E pivote = lista.get(Math2.getEnteroAleatorio(i, j));
            IntPair p = banderaHolandesa(lista, pivote, i, j, ord);
            quickSort(lista, i, p.first(), um, ord);
            quickSort(lista, p.second(), j, um, ord);
        }
    }

    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);
                }
            }
        }
    }

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

```

```

    b++;
  }
}
return IntPair.of(a, b);
}

public static void test() {
Files2.linesFromFile("ficheros/Listasej3.txt").forEach(l -> {
String[] e = l.split(",");
List<Integer> ls = new ArrayList<>();
for (int i = 0; i < e.length; i++)
ls.add(Integer.parseInt(e[i]));
System.out.println("Lista. " + ls);
System.out.println("¿Esta ordenada?:" + List2.isOrdered(ls));
quickSortMT(ls, 2);
System.out.println("¿Y ahora?:" + List2.isOrdered(ls));
System.out.println(ls);
});
}

public static void main(String[] args) {
test();
// List<Integer > ls = new ArrayList<>();
// for(int i = 0; i<15; i++)
// ls.add(Math2.getEnteroAleatorio(0, 999));
// System.out.println("Sin ordenar:" + ls);
// quickSortMT(ls);
// System.out.println("Ordenada. "+ ls);
// System.out.println(List2.isOrdered(ls, Comparator.naturalOrder()));
}
}

```

3.2. Test:

```

package tests;

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

```

```

import java.util.function.BiFunction;
import java.util.function.Consumer;
import java.util.function.Function;
import java.util.stream.Collectors;

import ejercicios.Ejercicio2MiguelToro;
import tests.TestEjemplo3.Problema;
import tests.TestEjemplo3.TResult;
import tests.TestEjemplo3.TResultD;
import tests.TestEjemplo3.TResultMedD;
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 us.lsi.math.Math2;
import us.lsi.tiposrecursivos.BinaryTree;
import utils.FicherosListas;
import utils.GraficosAjuste;
import utils.Resultados;
import utils.TipoAjuste;
import utils.FicherosListas.PropiedadesListas;

public class TestEj2 {

    public static void main(String[] args) {

        // generamos los datos a usar
        PropiedadesListas props = PropiedadesListas.of(sizeMin, sizeMax, numSizes,
        minValue, maxValue, numListPerSize,
        numCasesPerList, rr);
        generaFicherosDatos(props);
        generaFicherosTiempoEjecucion();
        muestraGraficas();
    }

    // boilerplate
    private static Integer sizeMin = 100; // tamaño mínimo de lista
    private static Integer sizeMax = 1000; // tamaño máximo de lista
    private static Integer numSizes = 50; // número de tamaños de listas
    private static Integer minValue = 0;
    private static Integer maxValue = 100000;
    private static Integer numListPerSize = 1; // número de listas por cada tamaño (UTIL???)
    private static Integer numCasesPerList = 1; // número de casos (elementos a buscar) por cada
    private static Random rr = new Random(System.nanoTime()); // para inicializarlo una sola vez
    // métodos que lo requieran

```

```

private static Integer numMediciones = 3; // número de mediciones de tiempo de cada caso (n)
private static Integer numIter = 50; // número de iteraciones para cada medición de tiempo
private static Integer numIterWarmup = 1000; // número de iteraciones para warmup
// otras random
private static Integer rMin = 5;
private static Integer rMax = 500;
private static Integer rInt = Math2.getEnteroAleatorio(rMin, rMax);

// metodos a testear
private static List<BiConsumer<List<Integer>, Integer>> metodos = List.of(
// TODO
Ejercicio2MiguelToro::quickSortMT // versión normal
);
private static List<String> etiquetasMetodos = List.of("QSMTUmbra14", "QSMTumbral2",
"QSMtUmbra18", "QSMTUmbra1Random");
private static List<String> ficheroListaEntrada = List.of("ficheros/Listasej2.txt");
private static List<Integer> umbrales = List.of(4,2,8,rInt);

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

for (Integer i=0; i<etiquetasMetodos.size(); i++) {
String ficheroMediosSalida =
String.format("ficheros/Tiempos%s.csv",etiquetasMetodos.get(i));
String label = etiquetasMetodos.get(i);
//TipoAjuste tipoAjuste = TipoAjuste.NLOGN_0;

// cuanto mas grande sea el umbral mas exponencia será
TipoAjuste tipoAjuste = i==3?TipoAjuste.EXP2_0:TipoAjuste.NLOGN_0;
GraficosAjuste.show(ficheroMediosSalida, tipoAjuste, label);
Pair<Function<Double, Double>, String> parCurve = GraficosAjuste.fitCurve(
DataCurveFitting.points(ficheroMediosSalida), tipoAjuste);
String ajusteString = parCurve.second();
if(true) { //TODO Modificar para restringir los valores
ficherosSalida.add(ficheroMediosSalida);
labels.add(String.format("%s      %s", label, ajusteString));
}
}
GraficosAjuste.showCombined("Quicksorts", ficherosSalida, labels);

}

```



```

public static void generaFicherosTiempoEjecucion() {
    for(Integer i= 0 ; i<etiquetasMetodos.size(); i++) {
        String ficheroSalida = String.format("ficheros/Tiempos%s.csv",
            etiquetasMetodos.get(i));
        System.out.println(ficheroSalida);
        String ficheroMediosSalida =
            String.format("ficheros/Tiempos%s.csv",etiquetasMetodos.get(i));
        testTiemposEjecucionQuickSort(ficheroListaEntrada.get(0),
            umbrales.get(i),
            metodos.get(0), //TODO Modificar para que corresponda con la i
            ficheroSalida,
            ficheroMediosSalida);
    }
}

// creacion de los ficheros de datos

private static void testTiemposEjecucionQuickSort(String ficheroListas, Integer umbral,

    BiConsumer<List<Integer>, Integer> quicksort, String ficheroSalida, String
    ficheroMediosSalida) {
    Map<Problema, Double> tiempos = new HashMap<Problema,Double>();
    Map<Integer, Double> tiemposMedios; // tiempos medios por cada tamaño
    List<String> lineasListas = Files2.linesFromFile(ficheroListas);
    // aki iria el fichero lista de umbrales,
    for(int iter=0; iter<numMediciones; iter++) {
        System.out.println(iter);
        // iterador por cada linea
        for(int i=0; i<lineasListas.size(); i++) {
            String []lineaLista = lineasListas.get(i).split(",");
            List<Integer> ls = List2.parse(lineaLista, t-> Integer.parseInt(t));
            Integer tam = ls.size();

            // String lineaElem = lineasElem.get(i*numCasesPerList+j);
            // List<String> lse = List2.parse(lineaElem, "#", Function.identity());
            Problema p = Problema.of(tam, i%1, 1);
            warmup(quicksort, ls, umbral);

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

```

```

    }
    tiemposMedios = tiempos.entrySet().stream()
        .collect(Collectors.groupingBy(x->x.getKey().tam(),
            Collectors.averagingDouble(x->x.getValue())
        )
    );
    ResultadosToFile(tiemposMedios.entrySet().stream()
        .map(x->TResultMedD.of(x.getKey(),x.getValue()))
        .map(TResultMedD::toString),
        ficheroMediosSalida,
        true);
}

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

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

}

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

}

public static void generaFicherosDatos(PropiedadesListas p) {
    Resultados.cleanFile("ficheros/Listasej2.txt"); // crea un file txt que se llama así
    // crea un version alternativa del crea listas enteros
    generaFicheroListasEnterosV1("ficheros/Listasej2.txt", p);
}

public static void generaFicheroListasEnterosV1(String fichero, PropiedadesListas p) {
    for (int i = 0; i < p.numSizes(); i++) {
        int tam = p.sizeMin() + i * (p.sizeMax() - p.sizeMin()) / (p.numSizes() - 1);
        for (int j = 0; j < p.numListPerSize(); j++) {
            List<Integer> ls = FicherosListas.generaListaEnteros(tam, p);
            String sls = ls.stream().map(x -> x.toString()).collect(Collectors.joining(", "));
            ResultadosToFile(String.format("%s", sls), fichero, false);
        }
    }
}

```

```

}
}

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

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

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

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

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

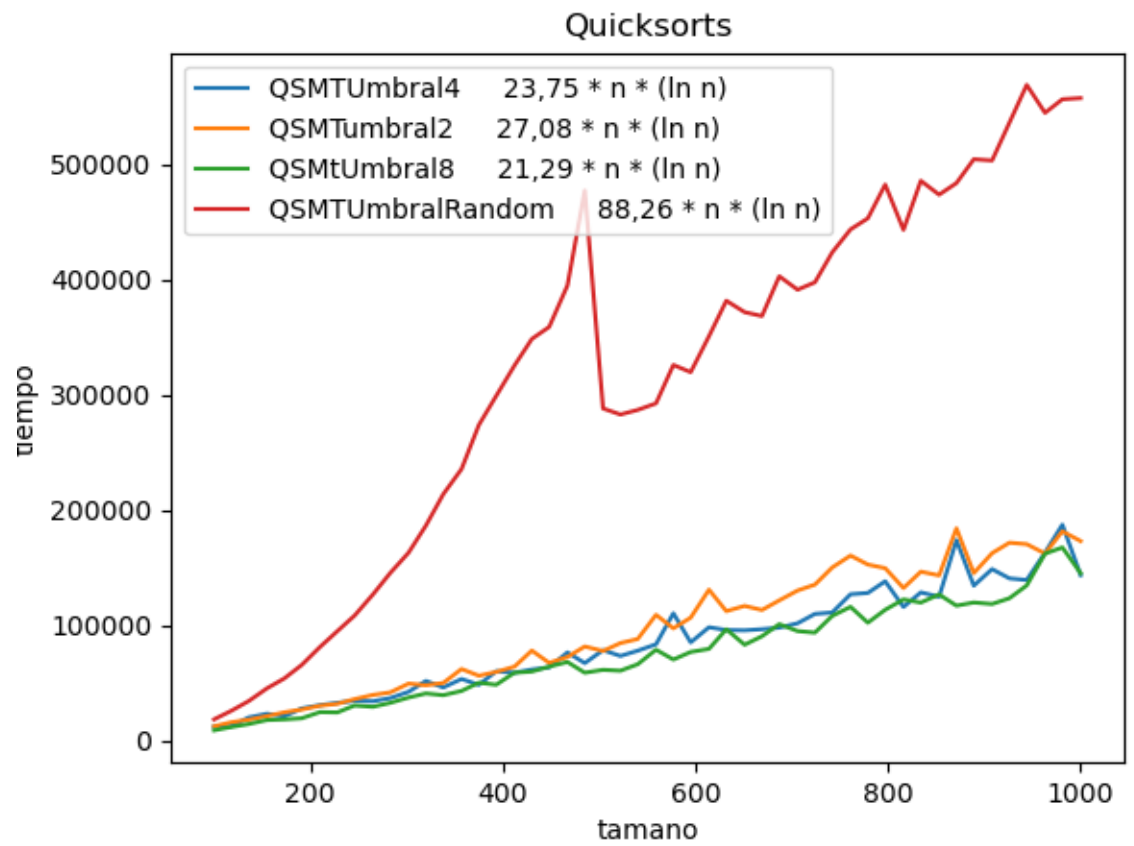
public String toString() {
return String.format("%d,%.0f", tam, t);
}
}

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

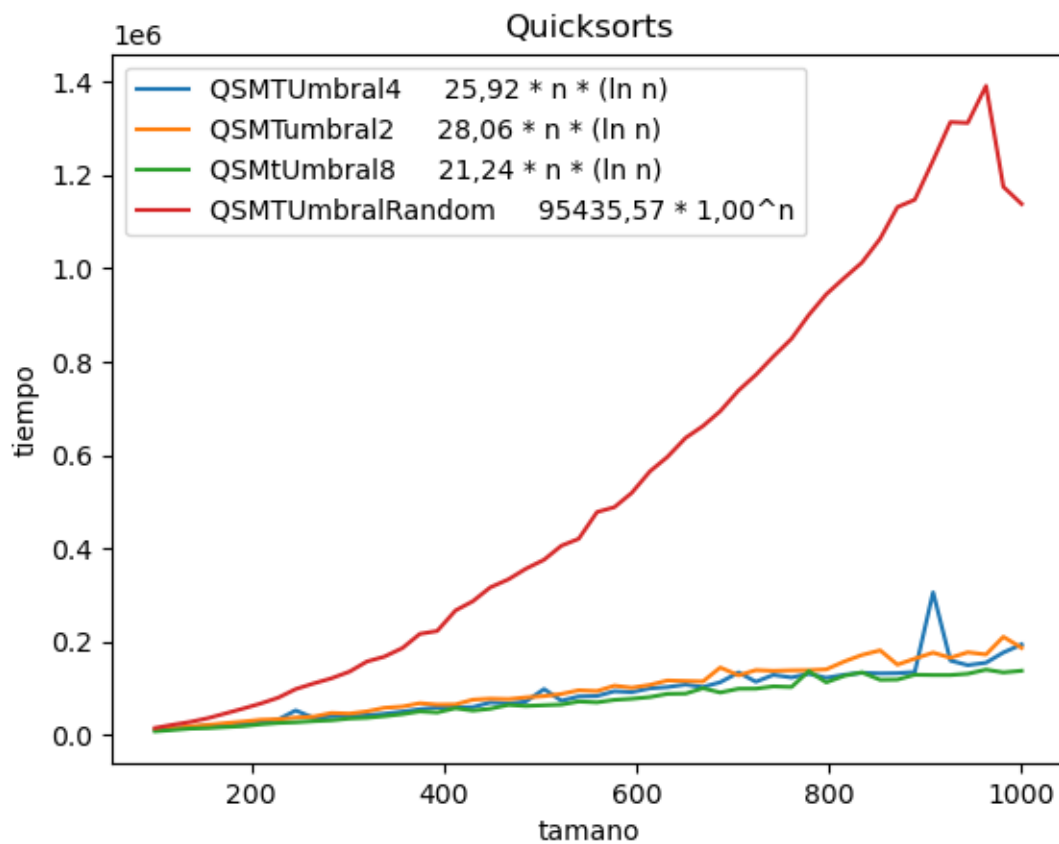
}

```

3.3. Imágenes



Versión con ajuste exponencial para el umbral random:



4. Ejercicio 2 QuickSort con InsertionSort

4.1. Código

```
package ejercicios;

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

import us.lsi.common.Files2;
import us.lsi.common.IntPair;
import us.lsi.common.List2;
import us.lsi.common.Preconditions;
```

```

import us.lsi.math.Math2;

public class Ejercicio2InsertionSort {
    public static void quickSortInsSort(List<Integer> ls, Integer um) {
        Comparator<Integer> ord = Comparator.naturalOrder();
        Integer i = 0;
        Integer j = ls.size();
        quickSort(ls, i, j, um, ord);
    }

    private static <E> void quickSort(List<E> lista, int i, int j, Integer um,
        Comparator<? super E> ord) {
        Preconditions.checkArgument(j >= i);
        if (j - i <= um) {
            insertionSort(lista, i, j, ord);
        } else {
            E pivote = lista.get(Math2.getEnteroAleatorio(i, j));
            IntPair p = banderaHolandesa(lista, pivote, i, j, ord);
            quickSort(lista, i, p.first(), um, ord);
            quickSort(lista, p.second(), j, um, ord);
        }
    }

    public static <T> void insertionSort(List<T> lista, Integer inf, Integer sup,
        Comparator<? super T> ord) {
        // https://www.youtube.com/watch?v=JU767SDMDvA
        for (int i = inf; i < sup; i++) {
            int j = i;
            while (j > 0 && ord.compare(lista.get(j - 1), lista.get(j)) > 0) {
                List2.intercambia(lista, j, j - 1);
                j--;
            }
        }
    }

    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);
}

public static void test() {
Files2.linesFromFile("ficheros/Listasej2.txt").forEach(l -> {
String[] e = l.split(",");
List<Integer> ls = new ArrayList<>();
for (int i = 0; i < e.length; i++)
ls.add(Integer.parseInt(e[i]));
System.out.println("Lista. " + ls);
System.out.println(";Esta ordenada?:" + List2.isOrdered(ls));
quickSortInsSort(ls, 2);
System.out.println(";Y ahora?:" + List2.isOrdered(ls));
System.out.println(ls);
});
}

public static void main(String[] args) {
test();
// List<Integer > ls = new ArrayList<>();
// for(int i = 0; i<15; i++)
// ls.add(Math2.getEnteroAleatorio(0, 999));
// System.out.println("Sin ordenar:" + ls);
// quickSortMT(ls);
// System.out.println("Ordenada. "+ ls);
// System.out.println(List2.isOrdered(ls, Comparator.naturalOrder()));
}
}

```

4.2. Test:

```

package tests;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

```

```

import java.util.Random;
import java.util.function.BiConsumer;
import java.util.function.BiFunction;
import java.util.function.Consumer;
import java.util.function.Function;
import java.util.stream.Collectors;

import ejercicios.Ejercicio2InsertionSort;
import ejercicios.Ejercicio2MiguelToro;
import tests.TestEjemplo3.Problema;
import tests.TestEjemplo3.TResult;
import tests.TestEjemplo3.TResultD;
import tests.TestEjemplo3.TResultMedD;
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 us.lsi.math.Math2;
import us.lsi.tiposrecursivos.BinaryTree;
import utils.FicherosListas;
import utils.GraficosAjuste;
import utils.Resultados;
import utils.TipoAjuste;
import utils.FicherosListas.PropiedadesListas;

public class TestEj2INSersionSort {

    public static void main(String[] args) {

        // generamos los datos a usar
        PropiedadesListas props = PropiedadesListas.of(sizeMin, sizeMax, numSizes, minValue,
        maxValue, numListPerSize,
        numCasesPerList, rr);
        generaFicherosDatos(props);
        generaFicherosTiempoEjecucion();
        muestraGraficas();
    }

    // boilerplate
    private static Integer sizeMin = 100; // tamaño mínimo de lista
    private static Integer sizeMax = 1000; // tamaño máximo de lista
    private static Integer numSizes = 50; // número de tamaños de listas
    private static Integer minValue = 0;
    private static Integer maxValue = 100000;
    private static Integer numListPerSize = 1; // número de listas por cada tamaño (UTIL???)

```



```

private static Integer numCasesPerList = 1; // número de casos (elementos a buscar) por cada
private static Random rr = new Random(System.nanoTime()); // para inicializarlo una sola vez
// métodos que lo requieran

private static Integer numMediciones = 3; // número de mediciones de tiempo de cada caso (n
private static Integer numIter = 50; // número de iteraciones para cada medición de tiempo
private static Integer numIterWarmup = 1000; // número de iteraciones para warmup
// otras random
private static Integer rMin = 200;
private static Integer rMax = 1000;
private static Integer rInt = Math2.getEnteroAleatorio(rMin, rMax);
private static String ubiFichero = "ficheros/Listasej2IS.txt";

// metodos a testear
private static BiConsumer<List<Integer>, Integer> metodo =
// TODO
Ejercicio2InsertionSort::quickSortInsSort ;// versión normal

private static List<String> etiquetasMetodos = List.of("QSIsumbral4", "QSIsumbral2",
"QSIsumbral8", "QSIsumbralRandom");
private static List<String> ficheroListaEntrada = List.of(ubiFichero);
private static List<Integer> umbrales = List.of(4,2,8,rInt);

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

for (Integer i=0; i<etiquetasMetodos.size(); i++) {
String ficheroMediosSalida =
String.format("ficheros/Tiempos%s.csv",etiquetasMetodos.get(i));
String label = etiquetasMetodos.get(i);
// TipoAjuste tipoAjuste = TipoAjuste.NLOGN_0;
TipoAjuste tipoAjuste = i==3?TipoAjuste.EXP2_0:TipoAjuste.NLOGN_0;
GraficosAjuste.show(ficheroMediosSalida, tipoAjuste, label);
Pair<Function<Double, Double>, String> parCurve = GraficosAjuste.fitCurve(
DataCurveFitting.points(ficheroMediosSalida), tipoAjuste);
String ajusteString = parCurve.second();
if(true) { //TODO Modificar para restringir los valores
ficherosSalida.add(ficheroMediosSalida);
labels.add(String.format("%s      %s", label, ajusteString));
}
}
GraficosAjuste.showCombined("Quicksorts", ficherosSalida, labels);

}

```

```

public static void generaFicherosTiempoEjecucion() {
    for(Integer i= 0 ; i<etiquetasMetodos.size(); i++) {
        String ficheroSalida = String.format("ficheros/Tiempos%s.csv",etiquetasMetodos.get(i));
        System.out.println(ficheroSalida);
        String ficheroMediosSalida =
            String.format("ficheros/Tiempos%s.csv",etiquetasMetodos.get(i));
        testTiemposEjecucionQuickSort(ubiFichero,
            umbrales.get(i),
            metodo, //TODO Modificar para que corresponda con la i
            ficheroSalida,
            ficheroMediosSalida);
    }
}

// creacion de los ficheros de datos

private static void testTiemposEjecucionQuickSort(String ficheroListas, Integer umbral,
    BiConsumer<List<Integer>, Integer> quicksort, String ficheroSalida, String
    ficheroMediosSalida) {
    Map<Problema, Double> tiempos = new HashMap<Problema,Double>();
    Map<Integer, Double> tiemposMedios; // tiempos medios por cada tamaño
    List<String> lineasListas = Files2.linesFromFile(ficheroListas);
    // aki iria el fichero lista de umbrales,
    for(int iter=0; iter<numMediciones; iter++) {
        System.out.println(iter);
        // iterador por cada linea
        for(int i=0; i<lineasListas.size(); i++) {
            String []lineaLista = lineasListas.get(i).split(",");
            List<Integer> ls = List2.parse(lineaLista, t-> Integer.parseInt(t));
            Integer tam = ls.size();

            // String lineaElem = lineasElem.get(i*numCasesPerList+j);
            // List<String> lse = List2.parse(lineaElem, "#", Function.identity());
            Problema p = Problema.of(tam, i%1, 1);
            warmup(quicksort, ls, umbral);

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

```

```

    tiemposMedios = tiempos.entrySet().stream()
        .collect(Collectors.groupingBy(x->x.getKey().tam(),
            Collectors.averagingDouble(x->x.getValue())
        )
    );
    ResultadosToFile(tiemposMedios.entrySet().stream()
        .map(x->TResultMedD.of(x.getKey(),x.getValue()))
        .map(TResultMedD::toString),
        ficheroMediosSalida,
        true);
}

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

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

}

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

}

public static void generaFicherosDatos(PropiedadesListas p) {
    Resultados.cleanFile(ubiFichero); // crea un file txt que se llama así
    // crea un version alternativa del crea listas enteros
    generaFicheroListasEnterosV1(ubiFichero, p);
}

public static void generaFicheroListasEnterosV1(String fichero, PropiedadesListas p) {
    for (int i = 0; i < p.numSizes(); i++) {
        int tam = p.sizeMin() + i * (p.sizeMax() - p.sizeMin()) / (p.numSizes() - 1);
        for (int j = 0; j < p.numListPerSize(); j++) {
            List<Integer> ls = FicherosListas.generaListaEnteros(tam, p);
            String sls = ls.stream().map(x -> x.toString()).collect(Collectors.joining(","));
            ResultadosToFile(String.format("%s", sls), fichero, false);
        }
    }
}

```

```

}

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

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

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

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

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

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

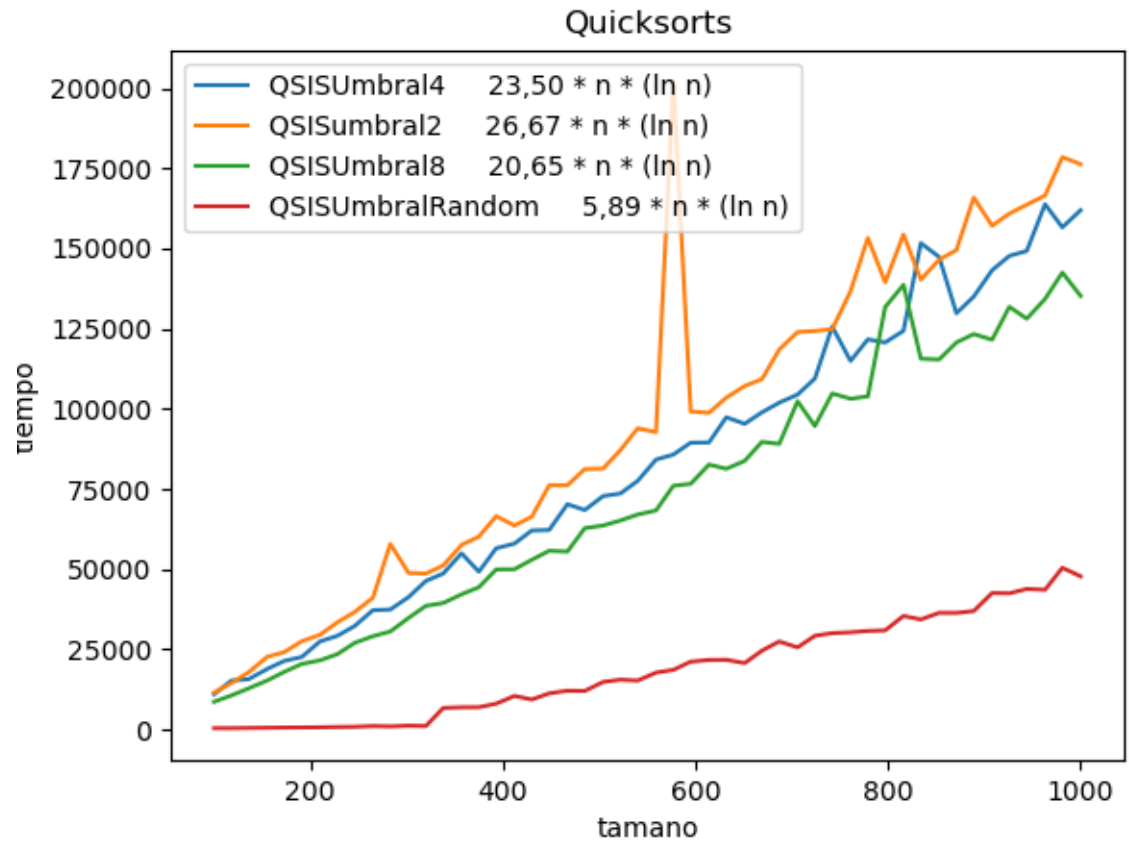
record Problema(Integer tam, Integer numList, Integer numCase) {
public static Problema of(Integer tam, Integer numList, Integer numCase){
return new Problema(tam, numList, numCase);
}
}

}

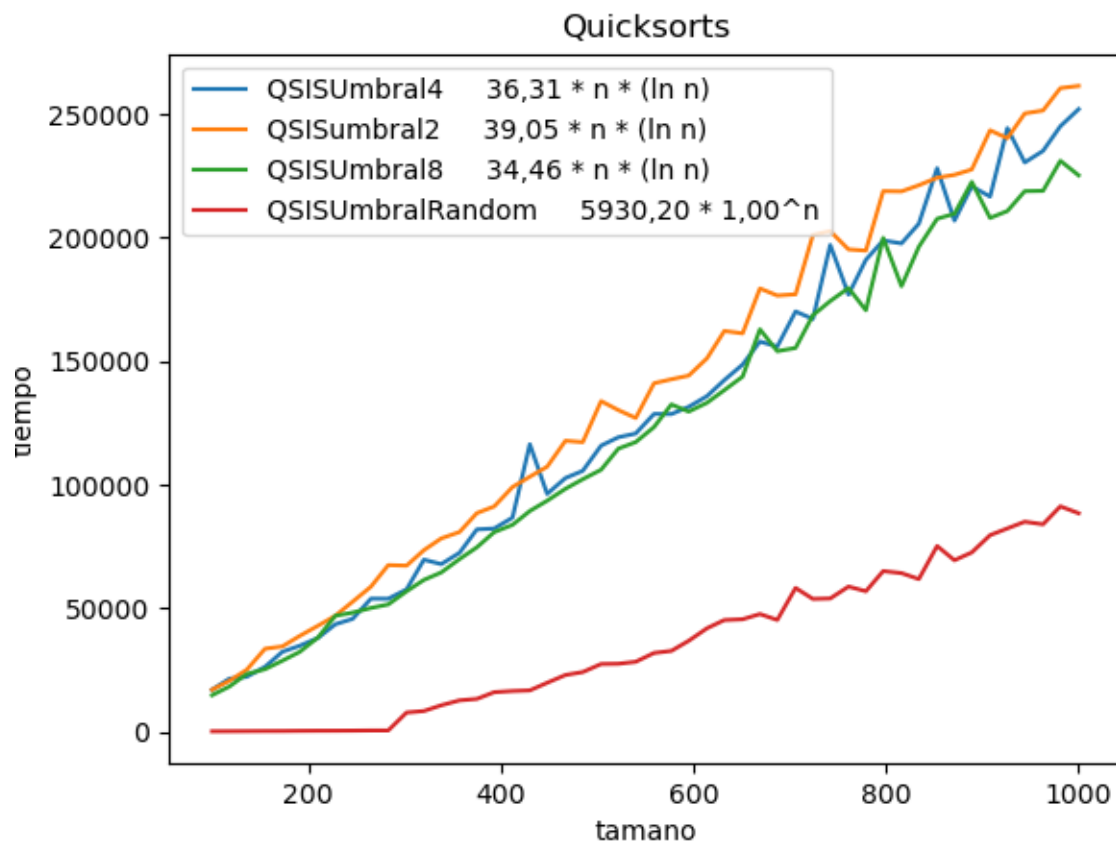
```

4.3. Imágenes:

El umbral random es un numero mayor que 4 escogido al azar, pero no entiendo porque es tan "eficiente", cuando debería ser exponencial conforme mas grande sea el numero random, mas veces entra en el caso base.



Versión alternativa de la gráfica con ajuste exponencial.



5. Ejercicio 3:

5.1. Código

```
package ejercicios;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.function.Consumer;

import us.lsi.common.Files2;
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.BinaryTrees;
import us.lsi.tiposrecursivos.Tree;
import us.lsi.tiposrecursivos.Tree.TEmpty;
import us.lsi.tiposrecursivos.Tree.TLeaf;
import us.lsi.tiposrecursivos.Tree.TNary;
import us.lsi.tiposrecursivos.parsers.BinaryTreeParser;

public class Ejercicio3 {
    public static List<String> caminosSinCaracterNario(Tree<Character> arbol, Character K) {
        List<String> res = new ArrayList<>();
        casoRecursivo(arbol, K, "", res);
        return res;
    }

    private static void casoRecursivo(Tree<Character> arbol, Character K, String ac, List<String> res) {
        // no se si hay que hacer un copy

        // ver la clase palindroma
        // https://blog.adamgamboa.dev/switch-expression-on-java-17/
        switch (arbol) {
            // caso empty, rompemos
            case TEmpty<Character> t:
                break;

            // caso hoja: comprobamos que si etiqueta no sea la letrea y slo mentemos en la cadena y la
            case TLeaf<Character> t:
                if (!t.label().equals(K)) {
                    ac = ac + t.label();
                    res.add(ac);
                }
                ;
                break;

            case TNary<Character> t:
                if (!t.label().equals(K)) {
                    ac = ac + t.label();
                    // con stream no sale :(
                    for (Tree<Character> a : t.elements()) {
                        casoRecursivo(a, K, ac, res);
                    }
                }
                break;
            default:
                break;
        }
    }
}

```

```

    }

    }

    public static List<String> caminosSinCaracterBinario(BinaryTree<Character> arbol, Character K, List<String> res) {
        List<String> res = new ArrayList<>();
        casoRecursoivoBT(arbol, K, "", res);
        return res;
    }

    private static void casoRecursoivoBT(BinaryTree<Character> arbol, Character K, String ac, List<String> res) {
        switch (arbol) {
            case BEmpty<Character> t:
                break;
            case BLeaf<Character> t:
                if (!t.label().equals(K)) {
                    ac = ac + t.label();
                    res.add(ac);
                }
                ;
                break;
            case BTree<Character> t:
                if (!t.label().equals(K)) {
                    ac = ac + t.label();
                    casoRecursoivoBT(t.left(), K, ac, res);
                    casoRecursoivoBT(t.right(), K, ac, res);
                }
                break;
            default:
                break;
        }
    }

    private static void cargaDatosBT() {
        Consumer<String> cnsmr = x -> {
            String[] y = x.split("#");
            BinaryTree<Character> bt = BinaryTree.parse(y[0], s -> s.charAt(0));

            System.out.println("Arbol: " + bt);
            System.out.println("Caracter: " + y[1]);
            System.out.println(caminosSinCaracterBinario(bt, y[1].charAt(0)));
        };
        Files2.streamFromFile("ficheros/Ejercicio3DatosEntradaBinario.txt").forEach(cnsmr);
    }
}

```



```

}

private static void cargaDatosNario() {
    Consumer<String> cnsmr = x -> {
        String[] y = x.split("#");
        Tree<Character> bt = Tree.parse(y[0], s -> s.charAt(0));

        System.out.println("Arbol: " + bt);
        System.out.println("Caracter: " + y[1]);
        System.out.println(caminosSinCaracterNario(bt, y[1].charAt(0)));
    };
    Files2.streamFromFile("ficheros/Ejercicio3DatosEntradaNario.txt").forEach(cnsmr);
}

public static void main(String[] args) {
    cargaDatosBT();
    cargaDatosNario();
}

}

```

5.2. Tests

```

package tests;

import java.util.function.Consumer;

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

public class TestEj3 {
    private static void cargaDatosBT() {
        Consumer<String> cnsmr = x -> {
            String[] y = x.split("#");
            BinaryTree<Character> bt = BinaryTree.parse(y[0], s -> s.charAt(0));

            System.out.println("Arbol: " + bt);
            System.out.println("Caracter: " + y[1]);
            System.out.println(Ejercicio3.caminosSinCaracterBinario(bt, y[1].charAt(0)));
        };
        Files2.streamFromFile("ficheros/Ejercicio3DatosEntradaBinario.txt").forEach(cnsmr);
    }
}

```

```

private static void cargaDatosNario() {
    Consumer<String> cnsmr = x -> {
        String[] y = x.split("#");
        Tree<Character> bt = Tree.parse(y[0], s -> s.charAt(0));

        System.out.println("Arbol: " + bt);
        System.out.println("Caracter: " + y[1]);
        System.out.println(Ejercicio3.caminoSinCaracterNario(bt, y[1].charAt(0)));
    };
    Files2.streamFromFile("ficheros/Ejercicio3DatosEntradaNario.txt").forEach(cnsmr);
}

public static void main(String[] args) {
    System.out.println("Versión Binaria");
    cargaDatosBT();
    System.out.println("Versión N Ario");
    cargaDatosNario();
}
}

```

5.3. Imágenes

```
Problems @ Javadoc Declaration Console x
<terminated> TestEj3 [Java Application] /usr/lib/jvm/jdk-18/bin/java (13
Versión Binaria
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]
Versión N Aria
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]
```

6. Ejercicio 4:

6.1. Código:

```
package ejercicios;

import java.util.function.Consumer;
import java.util.function.Function;

import us.lsi.common.Files2;
import us.lsi.common.String2;
import us.lsi.streams.Stream2;
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 {

    public static Boolean ej4BinarioV1(BinaryTree<String> bt) {
        // version buena
        return casoRecursoivoBtV1(bt, true);
    }

    private static Boolean casoRecursoivoBtV1(BinaryTree<String> bt, Boolean res) {

        switch (bt) {
            case BEmpty<String> t:
                break; // rompemos directamente
            case BLeaf<String> t:
                break; // rompemos directamente
            case BTree<String> t:

                Integer a = t.left().isEmpty() ? 0 : nVocalesBT(t.left());
                Integer b = t.right().isEmpty() ? 0 : nVocalesBT(t.right());
                if (a.equals(b)) {
                    res = casoRecursoivoBtV1(t.left(), res) && casoRecursoivoBtV1(t.right(), res);
                } else
                    res = false;

                break; // solo se comparan los hijos
        }
    }
}
```

```

    }
    return res;
}

public static Boolean ej4NarioV1(Tree<String> nt) {
    return casoRecursoNarioV1(nt);
}

private static Boolean casoRecursoNarioV1(Tree<String> nt) {
    Boolean res = true;
    switch (nt) {
        case TEmpty<String> t:
            break; // rompemos directamente
        case TLeaf<String> t:
            break; // rompemos directamente
        case TNary<String> t:

            if (Stream2.allEquals(t.elements().stream().map(Ejercicio4::nVocales))) {

                res = t.elements().stream().map(Ejercicio4::casoRecursoNarioV1).reduce((r, u) -> r && u).get();

            } else
                res = false;

            break; // solo se comparan los hijos

    }
    return res;
}

private static Integer nVocalesBT(BinaryTree<String> tree) {
    Integer res = 0;
    switch (tree) {
        case BEmpty<String> t:
            res = 0;
            break; // devuelve 0
        case BLeaf<String> t:
            res = cuentavocales(t.label());
            break;
        case BTree<String> t: // comprueba sus vocales y devuelvelas suyas mas la sum de sus hijos

            res = cuentavocales(t.label()) + nVocalesBT(t.left()) + nVocalesBT(t.right());
            break;
    }
}

```

```

    }

    return res;
}

private static Integer nVocales(Tree<String> tree) {
    Integer res = null;
    switch (tree) {
        case TEmpty<String> t:
            res = 0; // devuelve 0
            break;
        case TLeaf<String> t:
            res = cuentavocales(t.label());
            break;
        case TNary<String> t: // comprueba sus vocales y devuelvelas suyas mas la sum de sus hijos
            Integer a = t.elements().stream().mapToInt(k -> nVocales(k)).sum();
            res = cuentavocales(t.label()) + a;
            break;
    }

    return res;
}

private static Integer cuentavocales(String s) {
    // complejidad probablemente n ( tamaño de la cadena)
    // https://stackoverflow.com/questions/19160921/how-do-i-check-if-a-char-is-a-vowel
    Integer res = 0;
    for (Integer i = 0; i < s.length(); i++) {
        if ("AEIOUaeiou".indexOf(s.charAt(i)) != -1) {
            res += 1;
        }
    }

    return res;
}

private static void cargaDatosBT() {
    Consumer<String> cnsmr = x -> {
        // String[] y = x.split("#");
        BinaryTree<String> bt = BinaryTree.parse(x);

        System.out.println("Arbol: " + bt);

        System.out.println(ej4BinarioV1(bt));
    };
    Files2.streamFromFile("ficheros/Ejercicio4DatosEntradaBinario.txt").forEach(cnsmr);
}

```

```

}

private static void cargaDatosNario() {
    Consumer<String> cnsmr = x -> {

        Tree<String> bt = Tree.parse(x);

        System.out.println("Arbol: " + bt);
        System.out.println("N vocales" + nVocales(bt));
        System.out.println(ej4NarioV1(bt));
    };
    Files2.streamFromFile("ficheros/Ejercicio4DatosEntradaNario.txt").forEach(cnsmr);
}

public static void main(String[] args) {
    cargaDatosBT();
    cargaDatosNario();
}

}

```


6.2. Test:

```

package tests;
import java.util.function.Consumer;
import ejercicios.Ejercicio4; import us.lsi.common.Files2; import us.lsi.tiposrecursivos.BinaryTree;
import us.lsi.tiposrecursivos.Tree;
public class TestEj4
private static void cargaDatosBT() Consumer<String> cnsmr = x -
    System.out.println("Arbol: - bt);
    System.out.println(Ejercicio4.ej4BinarioV1(bt)); @i // String[] y = x.split("");
BinaryTree<String> bt = BinaryTree.parse(x);
    System.out.println("Arbol: - bt);
    System.out.println(Ejercicio4.ej4BinarioV1(bt)); ; Files2.streamFromFile("ficheros/Ejercicio4DatosEntradaN
private static void cargaDatosNario() Consumer<String> cnsmr = x -
    Tree<String> bt = Tree.parse(x);
    System.out.println("Arbol: - bt);
    System.out.println(Ejercicio4.ej4NarioV1(bt)); @i
    Tree<String> bt = Tree.parse(x);
    System.out.println("Arbol: - bt);
    System.out.println(Ejercicio4.ej4NarioV1(bt)); ; Files2.streamFromFile("ficheros/Ejercicio4DatosEntradaN
public static void main(String[] args) cargaDatosBT(); cargaDatosNario();

```

6.3. Imágenes:



```
<terminated> TestEj4 [Java Application] /usr/lib/jvm/jdk-18/bin/java (13 nov 2022 16:58:39 -
Arbol: pepe(pepa,pepe)
true
Arbol: pepe(pepa,pep)
false
Arbol: ada(eda(ola,ale),eda(ele,ale))
true
Arbol: ada(eda(ola,ale),eda(ele,al))
false
Arbol: cafe(taza(bote,bolsa),perro(gato,leon))
true
Arbol: cafe(taza(bote,bolsa),perro(gato,_))
false
Arbol: cafe(taza(bote,bolsa),perro(gato,tortuga))
false
Arbol: cafe(SO(AEI,E),ADDA(TC,ISSI))
false
Arbol: pepe(pepa,pepe,pepo)
true
Arbol: pepe(pepa,pepe,pep)
false
Arbol: ada(eda(ola,ale,elo),eda(ele,ale,alo))
true
Arbol: ada(eda(ola,ale,elo),eda(ele,ale,al))
false
Arbol: cafe(taza(bote,bolsa,vaso),perro(gato,leon,tigre))
true
Arbol: cafe(taza(bote,bolsa,vaso),perro(gato,leon))
false
Arbol: cafe(taza(bote,bolsa,vaso),perro(gato,tortuga))
false
```