**PI2 Juan Orellana Carretero**

**Ejercicio 1**

**import** java.math.BigInteger;

**public** **class** Ejercicio1 {

/\*

\* PI2 EJERCICIO 1

\*

\* Analizar los tiempos de ejecucion de las versiones recursivas

\* e iterativa para el cálculo del factorial. Comparar segun los resultados sean

\* de tipo Double o BigInteger

\*

\*/

**public** **static** Double factorialDouble(Integer n) {

Double r;

**if** (n == 0) {

r = 1.;

} **else** {

r = factorialDouble(n - 1) \* n;

}

**return** r;

}

**public** **static** Double factorialDoubleIt(Integer n) {

Double r = 1.;

Integer i = 1;

**while** (i<=n) {

r = r \* i;

i = i+1;

}

**return** r;

}

**public** **static** BigInteger factorialBigInteger(Integer n) {

BigInteger r;

**if** (n == 0) {

r = BigInteger.valueOf(1);

} **else** {

r = BigInteger.valueOf(n).multiply(factorialBigInteger(n-1));

}

**return** r;

}

**public** **static** BigInteger factorialBigIntegerIt(Integer n) {

BigInteger r = BigInteger.valueOf(1);

Integer i = 1;

**while** (i<=n) {

r = r.multiply(BigInteger.valueOf(i));

i = i+1;

}

**return** r;

}

}

**Ejercicio 2**

**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** <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);

}

**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);

}

}

}

}

}

**Ejercicio 3**

import java.util.ArrayList;

import java.util.List;

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 {

public static List<String> solucion\_recursivaBinario(BinaryTree<Character> tree,

Character c) {

return *recursivoBinario*(tree, c, new ArrayList<>(), "");

}

public static List<String> recursivoBinario(BinaryTree<Character> tree, Character c, List<String> res, String cadena){

return switch(tree) {

case BEmpty<Character> t -> res;

case BLeaf<Character> t ->{

String s = cadena + t.label();

if(!s.contains(String.*valueOf*(c))) {

res.add(s);

}

yield res;

}

case BTree<Character> t -> {

String s = cadena + t.label();

if(!s.contains(String.*valueOf*(c))) {

res = *recursivoBinario*(t.left(), c, res, s);

res = *recursivoBinario*(t.right(), c, res, s);

}

yield res;

}

};

}

public static List<String> solucion\_recursivaNario(Tree<Character> tree,

Character c) {

return *recursivoNario*(tree, c,"", new ArrayList<>());

}

public static List<String> recursivoNario(Tree<Character> tree, Character c,String cadena, List<String> res){

return switch(tree) {

case TEmpty<Character> t-> res;

case TLeaf<Character> t->{

String ac = cadena + t.label();

if(!ac.contains(String.*valueOf*(c))) {

res.add(ac);

}

yield res;

}

case TNary<Character> t->{

String ac = cadena + t.label();

if(!ac.contains(String.*valueOf*(c))) {

for(Tree<Character> child:t.elements()) {

res = *recursivoNario*(child, c, ac, res);

}

}

yield res;

}

};

}

}

**EJERCICIO 4**

import java.util.List;

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 record Tupla(Boolean b, Integer n) {

public static Tupla of(Boolean b, Integer n) {

return new Tupla(b,n);

}

}

public static Boolean solucion\_recursivaBinaria(BinaryTree<String> tree) {

return *recursivoBinario*(tree).b();

}

private static Boolean esVocal(Character letra) {

return "aeiou".contains(String.*valueOf*(letra).toLowerCase());

}

private static Integer cuentaVocales(String s) {

int vocales = 0;

for (int i=0; i < s.length(); i++) {

char letraActual = s.charAt(i);

if (*esVocal*(letraActual)) vocales++;

}

return vocales;

}

private static Tupla recursivoBinario(BinaryTree<String> tree) {

return switch(tree) {

case BEmpty<String> t -> Tupla.*of*(true, 0);

case BLeaf<String> t -> {

Boolean res = true;

Integer sumaP = *cuentaVocales*(t.label());

yield Tupla.*of*(res, sumaP);

}

case BTree<String> t->{

Boolean res;

Tupla l = *recursivoBinario*(t.left());

Tupla r = *recursivoBinario*(t.right());

Integer sumaL = l.n();

Integer sumaR = r.n();

Integer sumaP = *cuentaVocales*(t.label());

Integer sumaTotal = sumaL + sumaR + sumaP;

if(sumaL==sumaR && l.b()==true && r.b()==true) {

res = true;

}else {

res = false;

}

yield Tupla.*of*(res, sumaTotal);

}

};

}

/\*

\*

\* Guardar en una lista de tuplas el resultado de todas las llamadas recursivas

\*

\* comprobar si los primeros elementos son true y sumar los segundos

\*

\*/

public static Boolean solucion\_recursivaNaria(Tree<String> tree) {

return *recursivoNario*(tree).b();

}

public static Tupla recursivoNario(Tree<String> tree) {

return switch(tree) {

case TEmpty<String> t -> Tupla.*of*(true, 0);

case TLeaf<String> t -> {

Boolean res = true;

Integer sP = *cuentaVocales*(t.label());

yield Tupla.*of*(res, sP);

}

case TNary<String> t -> {

Boolean res = true;

List<Tupla> ls = t.elements().stream().map(x->*recursivoNario*(x)).toList();

Integer s = *cuentaVocales*(t.label());

Integer numeroVocales = ls.get(0).n();

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

s = ls.get(i).n() + s;

if(ls.get(i).b() == true && numeroVocales.equals(ls.get(i).n())) {

res = true;

}else {

res = false;

}

}

yield Tupla.*of*(res, s);

}

};

}

**TESTS de los ejercicios**

**TEST EJERCICIO 1**

**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 us.lsi.math.Math2;

**import** utils.GraficosAjuste;

**import** utils.Resultados;

**import** utils.TipoAjuste;

**public** **class** TestEjercicio1 {

**public** **static** **void** main(String[] args) {

generaFicherosTiempoEjecucion();

muestraGraficas();

}

**private** **static** Integer nMin = 2; // n mínimo

**private** **static** Integer nMax = 10000; //

**private** **static** Integer numSizes = 30; // número de problemas

**private** **static** Integer numMediciones = 3; //10; // número de mediciones de tiempo de cada caso (número de experimentos)

// para exponencial se puede reducir

**private** **static** Integer numIter = 50; //50; // número de iteraciones para cada medición de tiempo !!

// para exponencial se puede reducir

**private** **static** Integer numIterWarmup = 100; // número de iteraciones para warmup

// Trios de métodos a probar con su tipo de ajuste y etiqueta para el nombre de los ficheros

**private** **static** List<Trio<Function<Integer, Number>, TipoAjuste, String>> metodosBigInteger =

List.of(

Trio.of(Ejercicio1::factorialBigInteger, TipoAjuste.POWERANB, "FactorialRecursivo\_BigInteger"),

Trio.of(Ejercicio1::factorialBigIntegerIt, TipoAjuste.POWERANB, "FactorialIterativo\_BigInteger")

);

**private** **static** List<Trio<Function<Integer, Number>, TipoAjuste, String>> metodosDouble =

List.of(

Trio.of(Ejercicio1::factorialDouble, TipoAjuste.POWERANB, "FactorialRecursivoDouble"),

Trio.of(Ejercicio1::factorialDoubleIt, TipoAjuste.POWERANB, "FactorialIterativoDouble")

);

**private** **static** <E> **void** generaFicherosTiempoEjecucionMetodos(List<Trio<Function<E, Number>, TipoAjuste, String>> metodos) {

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

**int** numMax = nMax ;

Boolean flagExp = **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(metodosBigInteger);

generaFicherosTiempoEjecucionMetodos(metodosDouble);

}

**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 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() {

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

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

muestraGraficasMetodos(metodosBigInteger, ficherosSalida, labels);

muestraGraficasMetodos(metodosDouble, ficherosSalida, labels);

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

}

@SuppressWarnings("unchecked")

**public** **static** <E> **void** testTiemposEjecucion(Integer nMin, Integer nMax,

Function<E, Number> funcionFac,

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(funcionFac, 10);

Integer nIter = flagExp ? 3 : numIter;

Number[] res = **new** Number[nIter];

Long t0 = System.nanoTime();

**for** (**int** z=0; z<nIter; z++) {

res[z] = funcionFac.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> fac, Integer n) {

BigInteger res=BigInteger.ZERO;

BigInteger z = BigInteger.ZERO;

**for** (**int** i=0; i<numIterWarmup; i++) {

**if** (fac.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) {

**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);

}

}

}

**TEST EJERCICIO 2**

**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** 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.FicherosListas;

**import** utils.GraficosAjuste;

**import** utils.Resultados;

**import** utils.TipoAjuste;

**import** utils.FicherosListas.PropiedadesListas;

**public** **class** TestEjercicio2 {

// Parámetros de generación de las listas

**private** **static** Integer sizeMin = 100; // tamaño mínimo de lista

**private** **static** Integer sizeMax = 10000; // tamaño máximo de lista

**private** **static** Integer numSizes = 50; // número de tamaños de listas

**private** **static** Integer minValue = 0;

**private** **static** Integer maxValue = 1000000;

**private** **static** Integer numListPerSize = 1; // número de listas por cada tamaño (UTIL???)

**private** **static** Integer numCasesPerList = 30; // número de casos (elementos a buscar) por cada lista

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

// Parámetros de medición

**private** **static** Integer numMediciones = 5; // 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 = 100000; // número de iteraciones para warmup

**public** **static** **void** main(String[] args) {

// Generación de Listas

PropiedadesListas props = PropiedadesListas.of(sizeMin, sizeMax, numSizes, minValue, maxValue,numListPerSize, numCasesPerList, rr);

generaFicherosDatos(props);

generaFicherosTiempoEjecucion(metodos);

muestraGraficas();

}

**private** **static** List<Trio<BiConsumer<List<Integer>, Integer>, TipoAjuste, String>> metodos =

List.of(

Trio.of(Ejercicio2::sort, TipoAjuste.NLOGN, "Sort")

);

**public** **static** **void** muestraGraficas() {

System.out.println("a\*n^b\*(ln n)^c + d");

List<Integer> umbrales = List.of(4, 20, 100, 500);

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

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

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

String ficheroSalida = String.format("ficheros/Tiempos%s\_%d.csv",

metodos.get(0).third(),umbrales.get(i));

ficherosSalida.add(ficheroSalida);

String label = metodos.get(0).third() + umbrales.get(i);

System.out.println(label);

TipoAjuste tipoAjuste = metodos.get(0).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("QuickSort", ficherosSalida, labels);

}

**public** **static** **void** generaFicherosDatos(PropiedadesListas p) {

Resultados.cleanFiles(List.of("ficheros/Listas.txt",

"ficheros/ListasOrdenadas.txt",

"ficheros/ElementosSI.txt",

"ficheros/ElementosNO.txt",

"ficheros/ElementosProb\_0.txt"));

FicherosListas.generaFicherosDatos(p,"ficheros/Listas.txt",

"ficheros/ListasOrdenadas.txt",

"ficheros/ElementosSI.txt",

"ficheros/ElementosNO.txt",

"ficheros/ElementosProb.txt",

0.5

);

}

**private** **static** <E> **void** generaFicherosTiempoEjecucion(List<Trio<BiConsumer<List<Integer>, Integer>, TipoAjuste, String>> metodos) {

List<Integer> umbrales = List.of(4, 20, 100, 500);

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

String ficheroSalida = String.format("ficheros/Tiempos%s\_%d.csv",

metodos.get(0).third(),umbrales.get(i));

testTiemposEjecucionBusqueda("ficheros/Listas.txt",

metodos.get(0).first(),

ficheroSalida,

umbrales.get(i));

}

}

**private** **static** **void** testTiemposEjecucionBusqueda(String ficheroListas,

BiConsumer<List<Integer>, Integer> busca,

String ficheroTiempos,

Integer umbral) {

Map<Problema, Double> tiempos = **new** HashMap<Problema,Double>();

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

**for** (**int** iter=0; iter<numMediciones; iter++) {

System.out.println(iter);

**for** (**int** i=0; i<lineasListas.size(); i++) { // numSizes\*numListPerSize

String lineaLista = lineasListas.get(i);

List<String> ls = List2.parse(lineaLista, "#", Function.identity());

Integer tam = Integer.parseInt(ls.get(0));

List<Integer> le = List2.parse(ls.get(1), ",", Integer::parseInt);

Problema p = Problema.of(tam);

warmup(busca, lineasListas.get(0));

Integer[] res = **new** Integer[numIter];

Long t0 = System.nanoTime();

**for** (**int** z=0; z<numIter; z++) {

busca.accept(le, umbral);

}

Long t1 = System.nanoTime();

actualizaTiempos(tiempos, p, Double.valueOf(t1-t0)/numIter);

}

}

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** **int** warmup(BiConsumer<List<Integer>, Integer> busca, String lineaLista) {

**int** res=0;

List<String> ls = List2.parse(lineaLista, "#", Function.identity());

Integer tam = Integer.parseInt(ls.get(0));

List<Integer> le = List2.parse(ls.get(1), ",", Integer::parseInt);

Integer umbral = 4;

Integer z = 0;

**for** (**int** i=0; i<numIterWarmup; i++) {

busca.accept(le, umbral);

}

res = z>0?z+tam:tam;

**return** res;

}

**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, 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** 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) {

**public** **static** Problema of(Integer tam){

**return** **new** Problema(tam);

}

}

}

**TEST EJERCICIO 3**

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) {

//testEjericio3Binario();

*testEjericio3Nario*();

}

public static void testEjericio3Binario() {

String file = "ficheros/Ejercicio3DatosEntradaBinario.txt";

List<Pair<BinaryTree<Character>, Character>> inputsBinario =

Files2.*streamFromFile*(file)

.map(linea ->{

String[] aux = linea.split("#");

BinaryTree<Character> t =

BinaryTree.*parse*(aux[0], s->s.charAt(0));

Character c = aux[1].charAt(0);

return Pair.*of*(t, c);

}

).toList();

for(Pair<BinaryTree<Character>, Character> par:inputsBinario) {

BinaryTree<Character> t = par.first();

Character c = par.second();

System.***out***.println("Arbol:" + t + "\tCaracter:" + c +

"\tResultado: " + Ejercicio3.*solucion\_recursivaBinario*(t, c));

}

}

public static void testEjericio3Nario() {

String file = "ficheros/Ejercicio3DatosEntradaNario.txt";

List<Pair<Tree<Character>, Character>> inputsNario =

Files2.*streamFromFile*(file)

.map(linea ->{

String[] aux = linea.split("#");

Tree<Character> t =

Tree.*parse*(aux[0], s->s.charAt(0));

Character c = aux[1].charAt(0);

return Pair.*of*(t, c);

}

).toList();

for(Pair<Tree<Character>, Character> par:inputsNario) {

Tree<Character> t = par.first();

Character c = par.second();

System.***out***.println("Arbol:" + t + "\tCaracter:" + c +

"\tResultado: " + Ejercicio3.*solucion\_recursivaNario*(t, c));

}

}

}

**TEST EJERCICIO 4**

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) {

//testEjercicio4Binario();

*testEjercicio4Nario*();

}

/\*

\* TEST EJERCICIO 4 BINARIO

\*/

public static void testEjercicio4Binario() {

String file = "ficheros/Ejercicio4DatosEntradaBinario.txt";

List<BinaryTree<String>> inputs = Files2.*streamFromFile*(file)

.map(linea -> BinaryTree.*parse*(linea))

.toList();

for(BinaryTree<String> tree:inputs) {

System.***out***.println("Arbol:" + tree +

"\tResultado: " + Ejercicio4.*solucion\_recursivaBinaria*(tree));

}

}

/\*

\* TEST EJERCICIO 4 NARIO

\*/

public static void testEjercicio4Nario() {

String file = "ficheros/Ejercicio4DatosEntradaNario.txt";

List<Tree<String>> inputs = Files2.*streamFromFile*(file)

.map(linea -> Tree.*parse*(linea))

.toList();

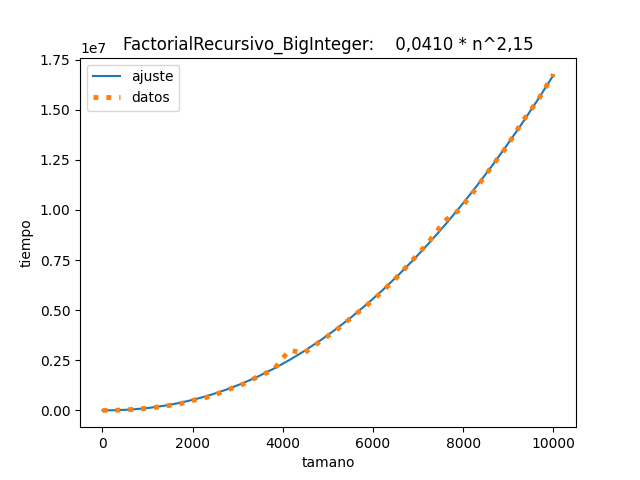
for(Tree<String> tree:inputs) {

System.***out***.println("Arbol:" + tree +

"\tResultado: " + Ejercicio4.*solucion\_recursivaNaria*(tree));

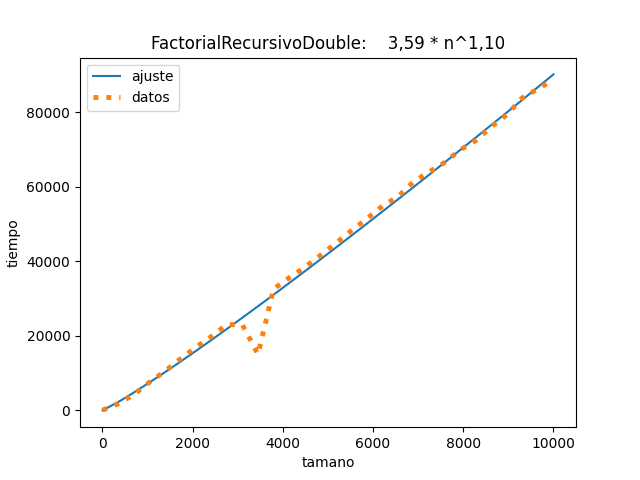
**Volcados de pantalla**

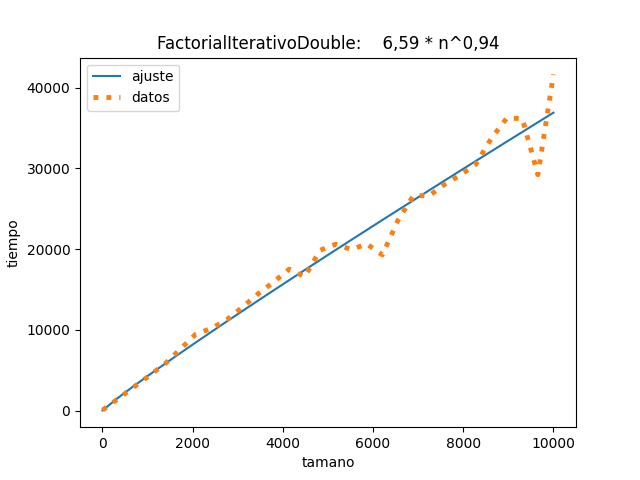
**Ejercicio 1**

****

**Gráfico, Gráfico de líneas

Descripción generada automáticamente**

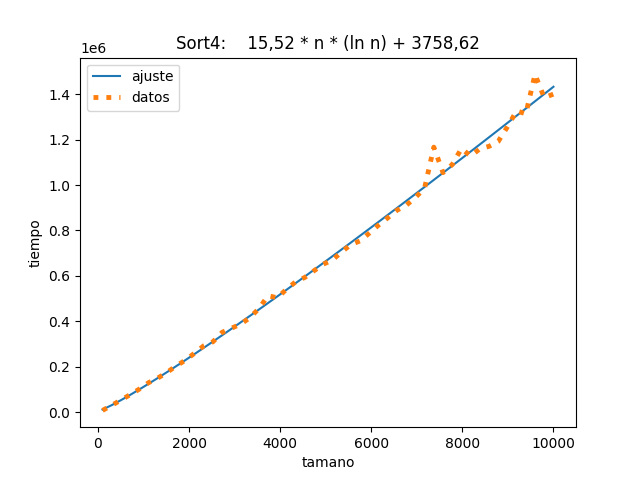
****

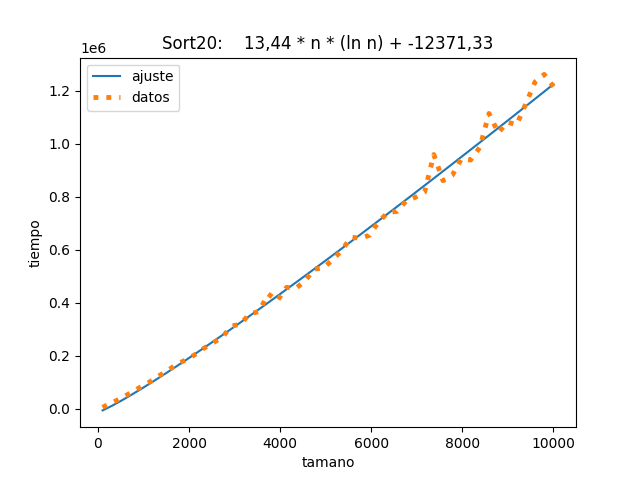
****

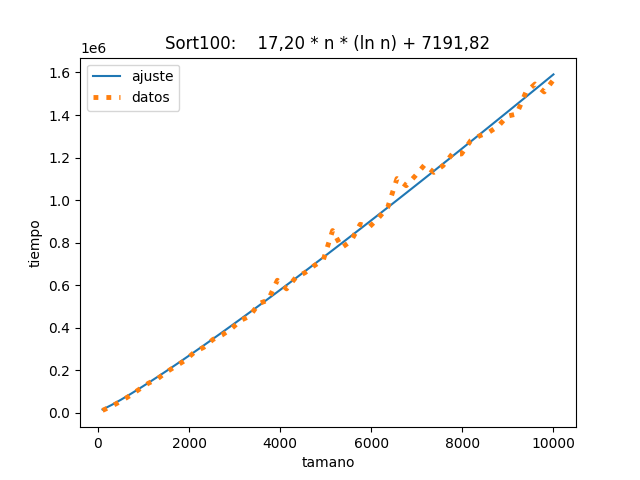
**Gráfico, Gráfico de líneas

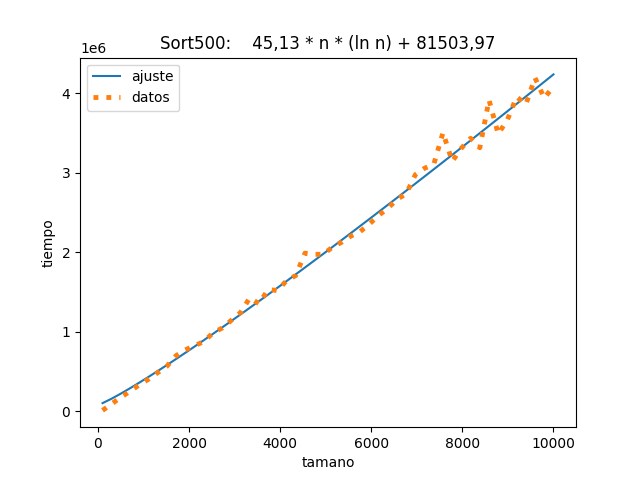
Descripción generada automáticamente**

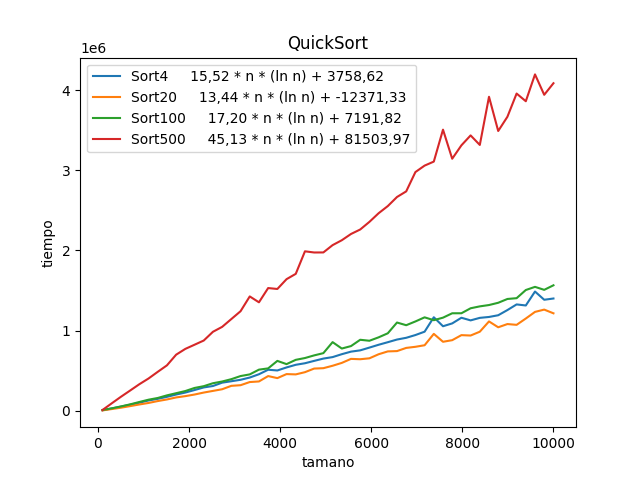
**Ejercicio 2**

****

****

****

****

****

**Ejercicio 3**

**BINARIO**

**Texto

Descripción generada automáticamente**

Arbol:A(B,C) Caracter:D Resultado: [AB, AC]

Arbol:A(B,C) Caracter:C Resultado: [AB]

Arbol:A(B,C) Caracter:A Resultado: []

Arbol:A(B(C,D),E(F,\_)) Caracter:H Resultado: [ABC, ABD, AEF]

Arbol:A(B(C,D),E(F,\_)) Caracter:D Resultado: [ABC, AEF]

Arbol:A(B(C,D(E,F(G,H))),I(J,K)) Caracter:H Resultado: [ABC, ABDE, ABDFG, AIJ, AIK]

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

**Texto

Descripción generada automáticamenteNARIO**

Arbol:A(B,C,D) Caracter:A Resultado: []

Arbol:A(B,C,D) Caracter:C Resultado: [AB, AD]

Arbol:A(B,C,D) Caracter:D Resultado: [AB, AC]

Arbol:A(B(C,D,E),F(G,H,I),J(K,L)) Caracter:F Resultado: [ABC, ABD, ABE, AJK, AJL]

Arbol:A(B(C,D,E),F(G,H,I),J(K,L)) Caracter:K Resultado: [ABC, ABD, ABE, AFG, AFH, AFI, AJL]

Arbol:A(B(C,D(E,F(G,H,I),J),K)) Caracter:D Resultado: [ABC, ABK]

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

**Ejercicio 4**

**Texto

Descripción generada automáticamenteBINARIO**

Arbol:pepe(pepa,pepe) Resultado: true

Arbol:pepe(pepa,pep) Resultado: false

Arbol:ada(eda(ola,ale),eda(ele,ale)) Resultado: true

Arbol:ada(eda(ola,ale),eda(ele,al)) Resultado: false

Arbol:cafe(taza(bote,bolsa),perro(gato,leon)) Resultado: true

Arbol:cafe(taza(bote,bolsa),perro(gato,\_)) Resultado: false

Arbol:cafe(taza(bote,bolsa),perro(gato,tortuga)) Resultado: false

**Texto

Descripción generada automáticamenteNARIO**

Arbol:pepe(pepa,pepe,pepo) Resultado: true

Arbol:pepe(pepa,pepe,pep) Resultado: false

Arbol:ada(eda(ola,ale,elo),eda(ele,ale,alo)) Resultado: true

Arbol:ada(eda(ola,ale,elo),eda(ele,ale,al)) Resultado: false

Arbol:cafe(taza(bote,bolsa,vaso),perro(gato,leon,tigre)) Resultado: true

Arbol:cafe(taza(bote,bolsa,vaso),perro(gato,leon)) Resultado: false

Arbol:cafe(taza(bote,bolsa,vaso),perro(gato,tortuga)) Resultado: false