METODOS DE ORDENAMIENTO

Yeisson Stiven Castro ([yeissoncc@gmail.com](mailto:yeissoncc@gmail.com))

Jhon Esteban Montoya ([j-esteban2004@hotmail.com](mailto:j-esteban2004@hotmail.com))

Corporación de estudios tecnológicos del norte del valle (Cotecnova)

**Resumen**

**Esta investigación es con el fin de saber cuál de los métodos de ordenamiento utilizados es más eficiente en cuanto a tiempo, a la hora de ejercer su trabajo con una gran cantidad de números que varían desde 1 millón hasta 20 millones, los métodos de ordenamiento a utilizar son: Ordenamiento por inserción, por mezcla, por montones (heap sort), rápido (quicksort), por conteo (counting sort) y por radix sort, para ello se consultaron algunos algoritmos para así poder implementar cada uno de estos métodos.**

**Palabras claves: Algoritmos, Métodos, Métodos de ordenamiento.**

**Abstract**

**This research is in order to know which of the methods of ordering used is more efficient in terms of time, when exercising their work with a large number of numbers ranging from 1 million to 20 million, Use are: Ordering by insertion, by heap sort, quicksort, by counting sort and by radix sort, for this we consulted some algorithms in order to implement each of these methods.**

**Key words: Algorithms, Methods, Methods of ordering.**

Introducción

Podremos observar los resultados de los diferentes métodos de ordenamiento que existen, utilizando una gran cantidad de datos. Los método utilizados son solo algunos, como por ejemplo, ordenamiento por inserción, ordenamiento por mezcla, ordenamiento por montones, ordenamiento rápido, ordenamiento por conteo y por último el ordenamiento por radix sort.

Métodos

Para poner a prueba estos métodos de ordenamiento se consultaron algunos códigos de los diferentes métodos en diferentes páginas web, pudiendo así implementarlos en el programa para obtener unos buenos resultados al momento de ejecutarlo.

Lenguaje de programación usado

JAVA: Es un [lenguaje de programación](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n) de [propósito general](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_de_prop%C3%B3sito_general), [concurrente](https://es.wikipedia.org/wiki/Computaci%C3%B3n_concurrente), [orientado a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos) que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los [desarrolladores](https://es.wikipedia.org/wiki/Desarrollador_de_software) de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como *WORA*, o "*write once, run anywhere*"), lo que quiere decir que el [código](https://es.wikipedia.org/wiki/C%C3%B3digo_fuente) que es ejecutado en una plataforma no tiene que ser [recompilado](https://es.wikipedia.org/wiki/Compilaci%C3%B3n_en_tiempo_de_ejecuci%C3%B3n) para correr en otra. Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de [cliente-servidor](https://es.wikipedia.org/wiki/Cliente-servidor) de web, con unos 10 millones de usuarios reportados.

Código de los algoritmos

**-Ordenamiento por inserción:** package metodos;

import java.util.ArrayList;

/\*Es una manera muy natural de ordenar para un ser humano, y puede usarse fácilmente para ordenar un mazo de cartas numeradas en forma arbitraria. Requiere O(nÂ²) operaciones para ordenar una lista de n elementos.\*/

public class insercion extends metodos\_ord{

public insercion(ArrayList<Integer> lis) {

super(lis);

}

@Override

public void ordenar(){

long time\_s = System.currentTimeMillis();

for(int i = 0; i < this.get\_tam()-1; i++){

for(int j = 0; j < this.get\_tam()-1; j++){

int a = this.lista\_desordenada.get(j);

int b = this.lista\_desordenada.get(j+1);

if(a>b){

this.lista\_desordenada.set(j, b);

this.lista\_desordenada.set(j+1,a);

}

}

}

long time\_e = System.currentTimeMillis();

this.tiempo\_ejecucion = time\_e - time\_s;

}

}

-Ordenamiento por conteo: package metodos;

import java.util.ArrayList;

/\*

Es un algoritmo de ordenamiento en el que se cuenta el número de elementos de cada clase para luego ordenarlos.

SÃ³lo puede ser utilizado por tanto para ordenar elementos que sean contables (como los números enteros en un

determinado intervalo, pero no los números reales)

\*/

public class conteo extends metodos\_ord{

public conteo(ArrayList<Integer> lis){

super(lis);

}

@Override

public void ordenar(){

long time\_s = System.currentTimeMillis();

this.lista\_desordenada = this.couting\_sort(this.lista\_desordenada);

long time\_e = System.currentTimeMillis();

this.tiempo\_ejecucion = time\_e - time\_s;

}

private ArrayList<Integer> couting\_sort(ArrayList<Integer> lis){

ArrayList<ArrayList<Integer>> tmp = this.sep\_num(lis);

ArrayList<Integer> res = new ArrayList<Integer>();

ArrayList<Integer> pos = tmp.get(0);

ArrayList<Integer> neg = tmp.get(1);

ArrayList<Integer> N\_neg = this.inv\_neg(neg);

ArrayList<Integer> res\_pos = this.couting(pos);

ArrayList<Integer> res\_neg = this.couting(N\_neg);

ArrayList<Integer> Nres\_neg = this.vol\_neg\_ord(res\_neg);

for(int i = 0; i < Nres\_neg.size(); i++){

res.add(Nres\_neg.get(i));

}

for(int i = 0; i < res\_pos.size(); i++){

res.add(res\_pos.get(i));

}

return res;

}

private ArrayList<Integer> couting(ArrayList<Integer> lis){

int n = lis.size();

int m = this.max\_val(lis) + 1;

ArrayList<Integer> count = new ArrayList<Integer>();

for(int i = 0; i < m; i++){

count.add(0);

}

for(int j = 0; j < n; j++){

int tmp = lis.get(j);

int tmp2 = count.get(tmp);

tmp2++;

count.set(tmp,tmp2);

}

int i = 0;

for(int k = 0; k < m; k++){

for(int l = 0; l < count.get(k); l++){

lis.set(i, k);

i++;

}

}

return lis;

}

private ArrayList<ArrayList<Integer>> sep\_num(ArrayList<Integer> lis){

ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();

ArrayList<Integer> pos = new ArrayList<Integer>();

ArrayList<Integer> neg = new ArrayList<Integer>();

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

if(lis.get(i) >= 0){

pos.add(lis.get(i));

}

else{

neg.add(lis.get(i));

}

}

res.add(pos);

res.add(neg);

return res;

}

private ArrayList<Integer> vol\_neg\_ord(ArrayList<Integer> neg){

ArrayList<Integer> res = new ArrayList<Integer>();

if (neg != null){

for(int i = neg.size()-1; i >= 0; i-- ){

res.add(neg.get(i)\* -1);

}

}

return res;

}

private ArrayList<Integer> inv\_neg(ArrayList<Integer> neg){

if(neg != null){

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

neg.set(i, neg.get(i) \* -1);

}

}

return neg;

}

private int max\_val(ArrayList<Integer> lis){

int max = 0;

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

if(max < lis.get(i)){

max = lis.get(i);

}

}

return max;

}

}

**-Ordenamiento por mezcla:** package metodos;

import java.util.ArrayList;

/\*

Es un algoritmo de ordenamiento externo estable basado en la técnica divide y vencerás.

Es de complejidad O(n log n).

\*/

public class mezcla extends metodos\_ord{

public mezcla(ArrayList<Integer> lis){

super(lis);

}

@Override

public void ordenar(){

long time\_s = System.currentTimeMillis();

this.lista\_desordenada = this.ordenar\_lis(this.lista\_desordenada);

long time\_e = System.currentTimeMillis();

this.tiempo\_ejecucion = time\_e - time\_s;

}

private ArrayList<Integer> ordenar\_lis(ArrayList<Integer> lis){

int n = lis.size();

int m = (int) n/2;

if (n==1){

return lis;

}

ArrayList<Integer> left = new ArrayList<Integer>();

for(int i = 0; i < m; i++){

int tmp = lis.get(i);

left.add(tmp);

}

left = this.ordenar\_lis(left);

ArrayList<Integer> right = new ArrayList<Integer>();

for(int i = m; i < lis.size(); i++){

int tmp = lis.get(i);

right.add(tmp);

}

right = this.ordenar\_lis(right);

return this.merge(left,right);

}

private ArrayList<Integer> merge(ArrayList<Integer> left,ArrayList<Integer> right){

ArrayList<Integer> result = new ArrayList<Integer>();

int i = 0;

int j = 0;

int len\_left = left.size();

int len\_right = right.size();

while(i < len\_left || j < len\_right){

if(i >= len\_left){

result.add(right.get(j));

j = j + 1;

}

else{

if(j >= len\_right){

result.add(left.get(i));

i = i + 1;

}

else{

if(left.get(i) < right.get(j)){

result.add(left.get(i));

i = i + 1;

}

else{

result.add(right.get(j));

j = j + 1;

}

}

}

}

return result;

}

}

**-Ordenamiento por montones:** package metodos;

import java.util.ArrayList;

/\*

Es un algoritmo de ordenamiento no recursivo, no estable, con complejidad computacional O(n\log n).

Este algoritmo consiste en almacenar todos los elementos del vector a ordenar en un montículo, y luego extraer el nodo

que queda como nodo raíz del montículo en sucesivas iteraciones obteniendo el conjunto ordenado.

\*/

public class monticulo extends metodos\_ord {

public monticulo(ArrayList<Integer> lis){

super(lis);

}

@Override

public void ordenar(){

long time\_s = System.currentTimeMillis();

this.lista\_desordenada = heap\_sort(this.lista\_desordenada);

long time\_e = System.currentTimeMillis();

this.tiempo\_ejecucion = time\_e - time\_s;

}

private ArrayList<Integer> heap\_sort(ArrayList<Integer> lis){

int fin = lis.size();

int start = ((int)(fin / 2)) - 1;

for(int i = start; i >= 0; i--){

this.heapify(lis,fin,i);

}

for(int i = fin-1; i > 0; i--){

this.swap(lis,i,0);

this.heapify(lis, i, 0);

}

return lis;

}

private void swap(ArrayList<Integer> lis, int i, int j){

int aux = lis.get(i);

lis.set(i, lis.get(j));

lis.set(j, aux);

}

private void heapify(ArrayList<Integer> lis, int fin, int i){

int l = 2 \* i + 1;

int r = 2 \* (i + 1);

int max = i;

if((l < fin) && (lis.get(i) < lis.get(l))){

max = l;

}

if((r < fin) && (lis.get(max) < lis.get(r))){

max = r;

}

if(max != i){

this.swap(lis,i,max);

this.heapify(lis, fin, max);

}

}

}

**-Ordenamiento rapido:** package metodos;

import java.util.ArrayList;

/\*

Es un algoritmo basado en la técnica de divide y vencerás, que permite, en promedio, ordenar n elementos en un

tiempo proporcional a n log n.

\*/

public class rapido extends metodos\_ord{

public rapido(ArrayList<Integer> lis){

super(lis);

}

@Override

public void ordenar(){

long time\_s = System.currentTimeMillis();

this.lista\_desordenada = this.quicksort(this.lista\_desordenada);

long time\_e = System.currentTimeMillis();

this.tiempo\_ejecucion = time\_e - time\_s;

}

private ArrayList<Integer> quicksort(ArrayList<Integer> lis){

int izq = 0;

int der = lis.size()-1;

ArrayList<Integer> res = quick(lis,izq,der);

return res;

}

private ArrayList<Integer> quick(ArrayList<Integer> lis,int izq,int der){

int i = izq;

int j = der;

int m = (int)((izq + der) / 2);

int x = lis.get(m);

while(i <= j){

while(lis.get(i) < x && j <= der){

i++;

}

while(x < lis.get(j) && j > izq){

j--;

}

if(i <= j){

int aux = lis.get(i);

lis.set(i, lis.get(j));

lis.set(j, aux);

i++;

j--;

}

if(izq < j){

this.quick(lis, izq, j);

}

if(i < der){

this.quick(lis, i, der);

}

}

return lis;

}

}

**-Ordenamiento por radix sort:** package metodos;

import java.util.ArrayList;

/\*

Es un algoritmo de ordenamiento que ordena enteros procesando sus digitos de forma individual.

Como los enteros pueden representar cadenas de caracteres y, especialmente, numeros en punto flotante especialmente

formateados, radix sort no esta limitado solo a los enteros.

\*/

public class radix extends metodos\_ord{

public radix(ArrayList<Integer> lis){

super(lis);

}

@Override

public void ordenar(){

long time\_s = System.currentTimeMillis();

this.lista\_desordenada = this.radixsort(this.lista\_desordenada);

long time\_e = System.currentTimeMillis();

this.tiempo\_ejecucion = time\_e - time\_s;

}

private ArrayList<Integer> radixsort(ArrayList<Integer> lis){

ArrayList<Integer> res = new ArrayList<Integer>();

ArrayList<ArrayList<Integer>> tmp = this.pos\_neg(lis);

ArrayList<Integer> pos = tmp.get(0);

ArrayList<Integer> neg = tmp.get(1);

neg = this.conv\_pos(neg);

int max\_pos = this.det\_digitos(pos);

int max\_neg = this.det\_digitos(neg);

int max = 0;

if (max\_pos >= max\_neg){

max = max\_pos;

}

else{

max = max\_neg;

}

for(int i = 0; i < max; i++){

ArrayList<ArrayList<Integer>> cola = new ArrayList<ArrayList<Integer>>();

for(int j = 0; j < 10; j++){

ArrayList<Integer> l = new ArrayList<Integer>();

cola.add(l);

}

pos = this.radix\_sort\_lsd(pos, cola, i+1);

}

for(int i = 0; i < max; i++){

ArrayList<ArrayList<Integer>> cola = new ArrayList<ArrayList<Integer>>();

for(int j = 0; j < 10; j++){

ArrayList<Integer> l = new ArrayList<Integer>();

cola.add(l);

}

neg = this.radix\_sort\_lsd(neg, cola, i+1);

}

ArrayList<Integer> tmp2 = new ArrayList<Integer>();

for(int i = neg.size() - 1; i >= 0; i--){

tmp2.add(neg.get(i) \* -1);

}

neg = tmp2;

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

res.add(neg.get(i));

}

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

res.add(pos.get(i));

}

return res;

}

private ArrayList<Integer> radix\_sort\_lsd(ArrayList<Integer> lis, ArrayList<ArrayList<Integer>> cola, int n){

int div = 1;

for(int i = 0; i < n; i++){

div = 10 \* div;

}

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

if (div == 10){

int pos = lis.get(i) % 10;

cola.get(pos).add(lis.get(i));

}

else{

int var = (int) (lis.get(i)/ (div / 10));

if (var == 0){

cola.get(0).add(lis.get(i));

}

else{

int mod = (int)(lis.get(i) % div);

int pos = (int)(mod/(div/10));

cola.get(pos).add(lis.get(i));

}

}

}

ArrayList<Integer> res = new ArrayList<Integer>();

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

for(int j = 0; j < cola.get(i).size(); j++){

if(cola.get(i).get(j) != null){

res.add(cola.get(i).get(j));

}

}

}

return res;

}

private ArrayList<Integer> conv\_pos(ArrayList<Integer> lis){

ArrayList<Integer> res = new ArrayList<Integer>();

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

res.add(lis.get(i)\* -1);

}

return res;

}

private ArrayList<ArrayList<Integer>> pos\_neg(ArrayList<Integer> lis){

ArrayList<ArrayList<Integer>> res = new ArrayList<ArrayList<Integer>>();

ArrayList<Integer> pos = new ArrayList<Integer>();

ArrayList<Integer> neg = new ArrayList<Integer>();

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

if(lis.get(i) >= 0){

pos.add(lis.get(i));

}

else{

neg.add(lis.get(i));

}

}

res.add(pos);

res.add(neg);

return res;

}

private int det\_digitos(ArrayList<Integer> lis){

int digitos = 0;

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

int tmp = con\_dig(lis.get(i));

if(tmp > digitos){

digitos = tmp;

}

}

return digitos;

}

private int con\_dig(int num){

String a = Integer.toString(num);

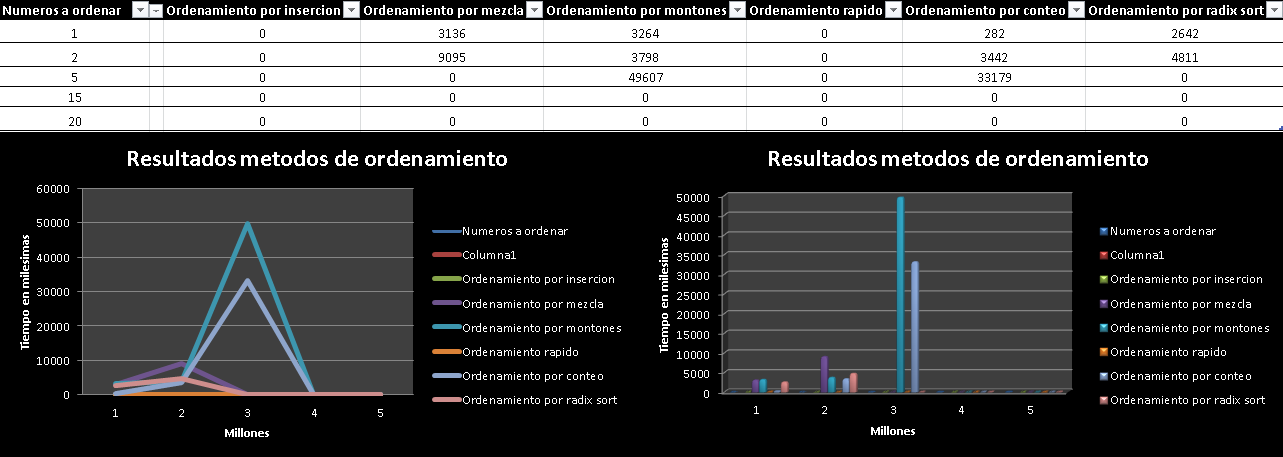
int tam = a.length();

return tam;

}

}

Evaluación de los resultados



Conclusiones

Con los anteriores resultados podemos concluir que algunos métodos de ordenamiento como lo son el ordenamiento por inserción y el ordenamiento rápido no pueden ordenar tanta capacidad de número, solo tienen capacidad para una cantidad de números más pequeña, en cambio con los otros métodos de ordenamiento se puede observar que su capacidad de ordenamiento es mayor.

Recomendaciones

Para ordenar cantidades hasta los 5 millones los métodos que pueden hacerlo son, ordenamiento por conteo y ordenamiento por montones, el que lo hace más rápido es el ordenamiento por conteo. Los demás métodos lo podrían hacer pero, tardarían horas incluso días para realizar este tipo de tareas.

Computador usado

DELL

Procesador: Intel® Pentium® 3558U @ 1.70GHz

Memoria RAM: 4,00GB

Sistema: Sistema operativo de 64bits

Referencias bibliográficas

<https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)>

<https://codigo--java.blogspot.com.co/2013/05/java-basico-021-ordenamiento-por.html>

<http://www.javamexico.org/foros/javam%C3%A9xico_20/ordenamiento_por_mezcla_ayudaa_urgente_porfavor>

<http://novella.mhhe.com/sites/dl/free/844814077x/619434/A06.pdf>

<http://puntocomnoesunlenguaje.blogspot.com.co/2012/12/java-quicksort.html>

<http://pier.guillen.com.mx/algorithms/03-ordenacion/03.5-countingsort.htm>

<http://foro.elhacker.net/java/algoritmo_radix_sort_en_java-t276529.0.html>