METODOS DE ORDENAMIENTO

Karen Guzmán Valencia

Cartago-Colombia

[Karenvalencia-96@hotmail.com](mailto:Karenvalencia-96@hotmail.com)

**RESUMEN**

Desde los comienzos de la computación, el problema del ordenamiento ha atraído gran cantidad de investigación, tal vez debido a la complejidad de resolverlo eficientemente a pesar de su planteamiento simple y familiar.

Por ejemplo, [BubbleSort](https://es.wikipedia.org/wiki/Bubblesort) fue analizado desde 1956.[1](https://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento#cite_note-1) Aunque muchos puedan considerarlo un problema resuelto, nuevos y útiles algoritmos de ordenamiento se siguen inventado hasta el día de hoy (por ejemplo, el [ordenamiento de biblioteca](https://es.wikipedia.org/wiki/Library_sort) se publicó por primera vez en el 2004).

Los algoritmos de ordenamiento son comunes en las clases introductorias a la computación, donde la abundancia de algoritmos para el problema proporciona una gentil introducción a la variedad de conceptos núcleo de los algoritmos, como [notación de O mayúscula](https://es.wikipedia.org/wiki/Cota_superior_asint%C3%B3tica), [algoritmos divide y vencerás](https://es.wikipedia.org/wiki/Algoritmo_divide_y_vencer%C3%A1s), [estructuras de datos](https://es.wikipedia.org/wiki/Estructura_de_datos), análisis de los [casos peor, mejor, y promedio](https://es.wikipedia.org/wiki/Casos_peor,_mejor,_y_promedio), y límites inferiores.

**Palabras Claves**

* Ordenamiento
* Complejidad
* Computación
* Planteamiento
* Problema

**ABSTRACT**

Since the beginning of computing, the problem of ordering has attracted a great deal of research, perhaps due to the complexity of solving it efficiently despite its simple and familiar approach.

 For example, BubbleSort has been analyzed since 1956.1 Although many may consider it a problem solved, new and useful sorting algorithms are still invented to this day (for example, library ordering was first published in 2004).

 Ordering algorithms are common in introductory classes to computation, where the abundance of algorithms for the problem provides a gentle introduction to the variety of core concepts of algorithms, such as uppercase O notation, divide and conquer algorithms, data structures , Worse, better, and average case analysis, and lower limits.

**KEYWORDS**

* Ordering
* Complexity
* Computational
* Approach
* Problem

**INTRODUCCIÓN**

En el siguiente trabajo se muestra cómo se pueden implementar los métodos de ordenamiento en la programación orientada a objetos por medio de algoritmos y tablas de resultados de cada uno de los diferentes tipos que existen para la programación.

Se puede decir que es un tema muy importante ya que en la vida actual la tecnología está tomando un poder que crece cada día para así facilitar la vida de muchas personas, ya que gracias a ella se pueden realizar actividades como: estudiar, trabajar, investigar entre otras.

Para este trabajo se utilizó el método de investigación descriptiva ya que nos ayuda a realizar un estudio y poder mostrar los resultados obtenidos del mismo.

**METODOLOGIA DESCRIPTIVA**

Su propósito se basa en exponer el evento estudiado, haciendo una enumeración detallada de sus características, de modo tal que en los resultados se pueden obtener dos niveles, dependiendo del fenómeno y del propósito del investigador:

1. Nivel más elemental: se logra una clasificación de la información en función de características comunes.
2. Nivel más sofisticado: se ponen en relación los elementos observados a fin de obtener una descripción más detallada.

**“Este tipo de investigación, la indagación va dirigida a responder las preguntas: quién, qué, dónde, cuándo, cuántos”** (Borderleau, 1997)

**ALGORITMOS USADOS**

* Ordenamiento por inserción: manera natural de ordenar para un ser humano y puede usarse fácilmente para ordenar un mazo de cartas numeradas. Complejidad O (n²).
* Ordenamiento por mezcla: algoritmo de ordenamiento externo estable basado en la técnica divide y vencerás. Complejidad O(n log n).
* Ordenamiento por montones (heap sort): algoritmo de ordenamiento no recursivo, no estable. Complejidad Θ(n log n).
* Ordenamiento rápido (quicksort): permite ordenar N elementos en un tiempo proporcional. Complejidad n log n. (Hoare, s.f.)
* Ordenamiento por conteo (counting sort): algoritmo de ordenamiento en el que se cuenta el número de elementos de cada clase para luego ordenarlos. Complejidad: n log n.
* Ordenamiento por radix sort: algoritmo que ordena enteros procesando sus dígitos de forma individual.

**LENGUAJE DE PROGRAMACION “JAVA”**

Es un lenguaje de programación con un propósito en general y concurrente orientado 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. 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.

Este lenguaje a partir del año 2012 es uno de los más importantes y populares ya que permite interactuar entre el cliente y el servidor web y fue desarrollado por el señor [James Gosling](https://es.wikipedia.org/wiki/James_Gosling).

**CODIGOS**

**Main**

package com.karen.mezcla;  
  
import java.util.Random;  
import java.util.Scanner;  
  
import static java.lang.System.*exit*;  
  
public class Main {  
 public static void main(String[] args){  
 char seleccionAlgoritmo = *seleccionAlgoritmo*();  
 AlgoritmoOrdenamiento algoritmo = *instanciarAlgoritmo*(seleccionAlgoritmo);  
 char seleccion1 = *primeraOpcion*();  
 int tamano = *cantidadNumeros*(seleccion1);  
 int[] arreglo = *generarNumeros*(tamano);  
 int[] arregloOrdenado=null;  
 char seleccion2 = *segundaOpcion*();  
 boolean imprimirPantalla= *imprimirSN*(seleccion2);  
 *ordenarArreglo*(algoritmo,arreglo, imprimirPantalla);  
 }  
  
 private static void ordenarArreglo(AlgoritmoOrdenamiento algoritmo,int[] arreglo, boolean imprimir) {  
 long inicio = System.*currentTimeMillis*();  
 if (imprimir){  
 System.*out*.println("Orden original");  
 algoritmo.mostrarArreglo(arreglo);  
 System.*out*.println(algoritmo.getNombre());  
 algoritmo.ordenar(arreglo);  
 algoritmo.mostrarArreglo(arreglo);  
 }else{  
 algoritmo.ordenar(arreglo);  
 }  
 long fin = System.*currentTimeMillis*();  
  
 long total = fin - inicio;  
 System.*out*.println("-------------------------\n");  
 System.*out*.println("Duracion total: " +total+" milisegundos");  
 }  
  
 private static AlgoritmoOrdenamiento instanciarAlgoritmo(char seleccionAlgoritmo) {  
 AlgoritmoOrdenamiento algoritmo=null;  
 switch (seleccionAlgoritmo){  
  
 case 'a':  
 algoritmo = new OrdenInsercion();  
 break;  
  
 case 'b':  
 algoritmo = new OrdenMezcla();  
 break;  
  
 case 'c':  
 algoritmo = new OrdenMontones();  
 break;  
  
 case 'd':  
 algoritmo = new OrdenRapido();  
 break;  
  
 case 'e':  
 algoritmo = new OrdenConteo();  
 break;  
  
 case 'f':  
 algoritmo = new OrdenRadixSort();  
 break;  
 case 'g':  
 *exit*(0);  
 break;  
 default:  
 *exit*(0);  
 break;  
 }  
 return algoritmo;  
 }  
  
 private static boolean imprimirSN(int seleccion2) {  
 boolean imprimir=false;  
 if (seleccion2 == 'a'){  
 imprimir = true;  
 }else if(seleccion2 == 'b'){  
 imprimir = false;  
 }else{  
 *exit*(0);  
 }  
 return imprimir;  
 }  
  
 private static char seleccionAlgoritmo() {  
 Scanner entrada = new Scanner(System.*in*);  
 System.*out*.println("Que algoritmo desea utilizar?");  
 System.*out*.println("-------------------------\n");  
 System.*out*.println("a. Ordernamiento por insercion ");  
 System.*out*.println("b. Ordernamiento por mezcla");  
 System.*out*.println("c. Ordernamiento por montones (heap sort)");  
 System.*out*.println("d. Ordernamiento rapido (quicksort)");  
 System.*out*.println("e. Ordernamiento por conteo (counting sort)");  
 System.*out*.println("f. Ordernamiento por radix sort");  
 System.*out*.println("g. Salir");  
 return entrada.next().charAt(0);  
 }  
  
 private static char primeraOpcion() {  
 Scanner entrada = new Scanner(System.*in*);  
 System.*out*.println("Cuanto enteros desea ordenar?");  
 System.*out*.println("-------------------------\n");  
 System.*out*.println("a. 1 Millon de datos ");  
 System.*out*.println("b. 2 Millones de datos");  
 System.*out*.println("c. 5 Millones de datos");  
 System.*out*.println("d. 10 Millones de datos");  
 System.*out*.println("e. 20 Millones de datos");  
 System.*out*.println("f. Salir");  
 return entrada.next().charAt(0);  
 }  
  
 private static char segundaOpcion() {  
 Scanner entrada = new Scanner(System.*in*);  
 System.*out*.println(" ");  
 System.*out*.println("Desea imprimir los valores?");  
 System.*out*.println("-------------------------\n");  
 System.*out*.println("a. Si ");  
 System.*out*.println("b. No");  
 System.*out*.println("c. Salir");  
 return entrada.next().charAt(0);  
 }  
  
 private static int cantidadNumeros(char seleccion) {  
 int tamano=0;  
  
 switch (seleccion){  
  
 case 'a':  
 tamano=10000;  
 break;  
 case 'b':  
 tamano=2000000;  
 break;  
 case 'c':  
 tamano=5000000;  
 break;  
 case 'd':  
 tamano=10000000;  
 break;  
 case 'e':  
 tamano=20000000;  
 break;  
 case 'f':  
 *exit*(0);  
 break;  
  
 }  
 return tamano;  
 }  
  
 private static int[] generarNumeros(int tamano) {  
 int[] arreglo;Random aleatorio = new Random();  
 arreglo= new int[tamano];  
 boolean bandera = false;  
 for(int i=0; i< tamano; i++ ){  
 arreglo[i] = aleatorio.nextInt(99999) ;  
 if (!bandera){  
 arreglo[i] = arreglo[i]\*-1;  
 }  
 bandera = !bandera;  
 }  
 return arreglo;  
 }  
  
  
}

**Ordenamiento por inserción**

package com.karen.mezcla;  
  
*/\*\*  
 \* Created by The Colombia karen guzman.  
 \*/*//herencia  
public class OrdenInsercion extends AlgoritmoOrdenamiento {  
  
 public OrdenInsercion(){  
 nombre="Orden por inserccion";  
 }  
  
 public void ordenarInsercion (int [] arreglo, int n){  
 int i,j,auxiliar;  
 for (i =1;i < n; i++){  
 auxiliar=arreglo[i];  
 j = i - 1;  
 while (j>=0 && arreglo[j] > auxiliar){  
 arreglo[j+1]=arreglo[j];  
 j=j-1;  
 }  
 arreglo[j+1]=auxiliar;  
 }  
  
 }  
 public void ordenar(int[] array){  
 ordenarInsercion(array,array.length);  
 }  
  
}

**Ordenamiento por mezcla**

package com.karen.mezcla;  
  
*/\*\*  
 \* Created by The Colombia karen guzman.  
 \*/*public class OrdenMezcla extends AlgoritmoOrdenamiento{  
  
 public OrdenMezcla(){  
 nombre="Orden por mezcla";  
 }  
  
 public void ordenamientoMezcla (int [] arreglo){  
 int i,j,k;  
 if( arreglo.length >1){  
 int numElementosIzquierdos = arreglo.length/2;  
 int numElementosDerechos = arreglo.length - numElementosIzquierdos;  
 int[] arregloIzquierdo = new int[numElementosIzquierdos];  
 int [] arregloDerecho = new int[numElementosDerechos];  
 for (i =0;i < numElementosIzquierdos; i++){  
 arregloIzquierdo[i] = arreglo[i];  
 }  
 for (i = numElementosIzquierdos; i < numElementosIzquierdos + numElementosDerechos; i++){  
 arregloDerecho[i - numElementosIzquierdos]= arreglo[i];  
 }  
 //Recursividad  
  
 ordenamientoMezcla(arregloIzquierdo);  
 ordenamientoMezcla(arregloDerecho);  
 //ordenar  
 i=0;  
 j=0;  
 k=0;  
 while (arregloIzquierdo.length != j && arregloDerecho.length !=k){  
 if (arregloIzquierdo[j] < arregloDerecho[k]){  
 arreglo[i] = arregloIzquierdo[j];  
 i++;  
 j++;  
 }else{  
 arreglo[i] = arregloDerecho[k];  
 i++;  
 k++;  
 }  
 }  
 //areglo final  
 while (arregloIzquierdo.length !=j){  
 arreglo[i] = arregloIzquierdo[j];  
 i++;  
 j++;  
 }  
 while (arregloDerecho.length !=k){  
 arreglo[i]= arregloDerecho[k];  
 i++;  
 k++;  
 }  
 }  
 }  
 public void ordenar(int[] arreglo){  
 ordenamientoMezcla(arreglo);  
 }  
}

**Ordenamiento por montones**

package com.karen.mezcla;  
  
*/\*\*  
 \* Created by The Colombia karen guzman.  
 \*/*//herencia  
public class OrdenMontones extends AlgoritmoOrdenamiento {  
  
 public OrdenMontones(){  
 nombre="Orden por heapSort";  
 }  
  
 public void heapSort(int[] arr) {  
 for (int i = arr.length; i > 1; i--) {  
 fnSortHeap(arr, i - 1);  
 }  
 }  
 public void fnSortHeap(int arr[], int arr2) {  
 int i, o;  
 int lCh, rCh, mCh, root, tmp;  
 root = (arr2 - 1) / 2;  
  
 for (o = root; o >= 0; o--) {  
 for (i = root; i >= 0; i--) {  
 lCh = (2 \* i) + 1;  
 rCh = (2 \* i) + 2;  
 if ((lCh <= arr2) && (rCh <= arr2)) {  
 if (arr[rCh] >= arr[lCh]) {  
 mCh = rCh;  
 } else {  
 mCh = lCh;  
 }  
 } else {  
 if (rCh > arr2) {  
 mCh = lCh;  
 } else {  
 mCh = rCh;  
 }  
 }  
  
 if (arr[i] < arr[mCh]) {  
 tmp = arr[i];  
 arr[i] = arr[mCh];  
 arr[mCh] = tmp;  
 }  
 }  
 }  
 tmp = arr[0];  
 arr[0] = arr[arr2];  
 arr[arr2] = tmp;  
 return;  
 }  
  
  
 public void ordenar(int[] arreglo){  
 heapSort(arreglo);  
 }  
  
}

**Ordenamiento rápido (quick sort)**

package com.karen.mezcla;  
  
*/\*\*  
 \* Created by The Colombia karen guzman.  
 \*/*//herencia  
public class OrdenRapido extends AlgoritmoOrdenamiento{  
  
 public OrdenRapido(){  
 nombre="Orden por quicksort";  
 }  
  
 public void ordenar(int[] arreglo){  
 *ordenamientoRapido*(arreglo,0,arreglo.length-1);  
 }  
 public static void ordenamientoRapido(int arreglo[],int primero, int ultimo){  
 int i = primero;  
 int j = ultimo;  
 if (i >= ultimo) {  
 return;  
 }  
 int pivote = arreglo[(i + j) / 2];  
 while (i < j) {  
 while (i<j && arreglo[i] < pivote) {  
 i++;  
 }  
 while (i<j && arreglo[j] > pivote) {  
 j--;  
 }  
 if (i < j) {  
 int auxiliar = arreglo[i];  
 arreglo[i] = arreglo[j];  
 arreglo[j] = auxiliar;  
 }  
 }  
 if (j < i) {  
 int T = j;  
 j = i;  
 i = T;  
 }  
 *ordenamientoRapido*(arreglo, primero, i);  
 *ordenamientoRapido*(arreglo, i == primero ? i+1 : i, ultimo);  
 }  
  
}

**Ordenamiento por conteo (Counting Sort)**

package com.karen.mezcla;  
  
*/\*\*  
 \* Created by The Colombia karen guzman.  
 \*/*//herencia  
public class OrdenConteo extends AlgoritmoOrdenamiento {  
  
 public OrdenConteo(){  
 nombre="Orden por conteo";  
 }  
  
 public static void countSort( int[] arr )  
 {  
 int N = arr.length;  
 if (N == 0)  
 return;  
 //valores maximos y minimos  
 int max = arr[0], min = arr[0];  
 for (int i = 1; i < N; i++)  
 {  
 if (arr[i] > max)  
 max = arr[i];  
 if (arr[i] < min)  
 min = arr[i];  
 }  
 int range = max - min + 1;  
 int[] count = new int[range];  
 //matriz de cuenta frecuancia  
 for (int i = 0; i < N; i++)  
 count[arr[i] - min]++;  
  
 //modificacion de cuenta que se obtienen en la posicion final  
 for (int i = 1; i < range; i++)  
 count[i] += count[i - 1];  
  
 //modificacion de array original  
 int j = 0;  
 for (int i = 0; i < range; i++)  
 while (j < count[i])  
 arr[j++] = i + min;  
 }  
  
 public void ordenar(int[] arreglo){  
 *countSort*(arreglo);  
 }  
  
}

**Ordenamiento Radix sort**

package com.karen.mezcla;  
  
*/\*\*  
 \* Created by The Colombia karen guzman.  
 \*/*//herencia  
public class OrdenRadixSort extends AlgoritmoOrdenamiento{  
  
 public OrdenRadixSort(){  
 nombre=" Order por radixsort";  
 }  
  
 public void ordenamientoRadixSort (int [] array){  
 int k,i,j;  
 for (k = Integer.*SIZE* - 1; k >= 0; k--){  
 int Auxiliar[] = new int[array.length];  
 j=0;  
 for (i =0 ;i < array.length;i++){  
 boolean Mover = array[i] << k>= 0;

//uso de Ternario?  
 if (k==0 ? !Mover:Mover){  
 Auxiliar[j] = array[i];  
 j++;  
 }else{  
 array[i-j] = array[i];  
 }  
 }  
 for (i=j;i<Auxiliar.length;i++){  
 Auxiliar[i] = array[i-j];  
 }  
 array = Auxiliar;  
 }  
  
 }  
 public void ordenar(int[] array){  
 ordenamientoRadixSort(array);  
 }  
  
}

**RESULTADOS**

Se puede notar en la tablas comparativas donde están ingresados todos los tiempos de cada uno de los métodos de ordenamiento que unos tardan más que otros, también dependiendo el computador utilizado el programa puede llegar a un punto donde ya no es capaz de soportar la cantidad de datos ingresados y no se puede obtener los tiempos entre un mejor procesador y memoria Ram tengas mejores resultados se pueden obtener.

**EVALUACION DE RESULTADOS**

[**..\Desktop\PROYECTO2\tablas proyecto2.xlsx**](../Desktop/PROYECTO2/tablas%20proyecto2.xlsx)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Miles** | **Inserción** | **Mezcla** | **Montones** | **Rapido** | **Conteo** | **RadixSort** |
| 1 | 3108Mls. | 2881Mls. | 8275Mls. | N/A | 3008Mls. | 3279Mls. |
| 2 | 5980Mls. | 5721Mls. | N/A | N/A | 6201Mls. | 6490Mls. |
| 5 | 18043Mls. | 17429Mls. | N/A | N/A | 14905Mls. | 16923Mls. |
| 10 | 29464Mls. | 28607Mls. | N/A | N/A | 29184Mls. | 32502Mls. |
| 20 | 60367Mls. | 57537Mls. | N/A | N/A | 59075Mls. | 65985Mls. |

**CONCLUSIONES Y RECOMENDACIONES**

A partir de los resultados obtenidos después de realizar el proyecto y observar sus respetivos resultados se puede decir que los métodos de ordenamiento son un poco complejos pero al aplicarlos y estudiarlos nos podemos dar cuenta de que unos funcionan mejor que otros dependiendo los datos que se le asignen ya que entre más cantidad datos se le apliquen a un método que solo puede ordenar una cantidad el programa se no puede colapsar.

Se puede concluir que uno de los métodos más eficientes es el ordenamiento por mezcla ya que nos ahorra más tiempo y se le pueden aplicar datos de 5 o más dígitos.

**COMPUTADOR USADO**

**Windows:** Windows 7 Ultimate.

**Procesador:** Pentium(R) Dual-Core CPU E5200 @2.50GHz 2.50GHz.

**Memoria Instalada (RAM):** 2,00 GB.

**Tipo de sistema:** Sistema operativo 32 bits.

**BIBLIOGRAFIA**

[**https://es.wikipedia.org/wiki/Java\_(lenguaje\_de\_programaci%C3%B3n)**](https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n))

[**http://aprenderlyx.com/tipos-de-metodologia-de-investigacion/**](http://aprenderlyx.com/tipos-de-metodologia-de-investigacion/)

[**https://es.wikipedia.org/wiki/Algoritmo\_de\_ordenamiento**](https://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento)