Algoritmos de Ordenamiento

Ayudante: Nelson Contreras

INTRODUCCION

- Los algoritmos de ordenamiento nos permite, como su nombre lo dice, ordenar.
- En este caso, nos servirán para ordenar vectores o matrices con valores asignados aleatoriamente.
- Nos centraremos en los métodos más populares, analizando la cantidad de comparaciones que suceden, el tiempo que demora y revisando el código, escrito en Java, de cada algoritmo.

METODOS ITERATIVOS

- Estos métodos son simples de entender y de programar ya que son iterativos, simples ciclos y sentencias que hacen que el vector pueda ser ordenado. Dentro de los Algoritmos iterativos encontramos:

- Burbuja
- Inserción
- Selección
- Shellsort

METODOS RECURSIVOS

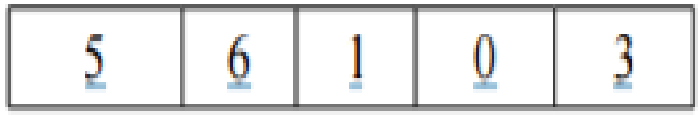
- Estos métodos son aún mas complejos, requieren de mayor atención y conocimiento para ser entendidos. Son rápidos y efectivos, utilizan generalmente la técnica Divide y vencerás, que consiste en dividir un problema grande en varios pequeños para que sea más fácil resolverlos.
- Mediante llamadas recursivas a si mismos, es posible que el tiempo de ejecución y de ordenación sea más optimo.

Dentro de los algoritmos recursivos encontramos:

- Ordenamiento por Mezclas (Merge)
- Ordenamiento Rápido (Quick)

METODO DE LA BURBUJA

- El método de la burbuja es uno de los mas simples, es tan fácil como comparar todos los elementos de una lista contra todos, si se cumple que uno es mayor o menor a otro, entonces los intercambia de posición.
- Por ejemplo, imaginemos que tenemos los siguientes valores:



- Lo que haría una burbuja simple, seria comenzar recorriendo los valores de izq. a derecha, comenzando por el 5. Lo compara con el 6, con el 1, con el 0 y con el 3, si es mayor o menor (dependiendo si el orden es ascendiente o descendiente) se intercambian de posición. Luego continua con el siguiente, con el 6, y lo compara con todos los elementos de la lista, esperando ver si se cumple o no la misma condición que con el primer elemento.
- Así, sucesivamente, hasta el ultimo elemento de la lista.

BURBUJA SIMPLE

- Como lo describimos en el item anterior, la burbuja mas simple de todas es la que compara todos con todos, generando comparaciones extras, por ejemplo, no tiene sentido que se compare con sigo mismo o que se compare con los valores anteriores a el, ya que supuestamente, ya están ordenados.

```
public void BubbleSort(){
    for(int i=0;i<n;i++){
        for(int j=0;j<(n-1)-i;j++){
            if(_bubble[j]>_bubble[j+1]){
                int swap = _bubble[j];
                _bubble[j] = _bubble[j+1];
                _bubble[j+1] = swap;
            }
        }
    }
    System.out.print("BubbleSort...\n---------\nOriginal: ");
    ImprimirVector(bubble);
    System.out.print("Ordenado: ");
    ImprimirVector(_bubble);
}
```

BURBUJA OPTIMIZADA

- Si al cambio anterior le sumamos otro cambio, el hecho que los elementos que están detrás del que se esta comparando, ya están ordenados, las comparaciones serian aun menos y el método seria aun mas efectivo.

- Si tenemos una lista de 10 elementos y estamos analizando el quinto elemento, que sentido tiene que el quinto se compare con el primero, el segundo o el tercero, si supuestamente, ya están ordenados?

Entonces optimizamos mas aun el algoritmo, quedando nuestra versión final del algoritmo optimizado de la siguiente manera:

```
Bubblesort(int matriz[])
{
        int buffer;
        int i,j;
        for(i = 0; i < matriz.length; i++)
        {
                for(j = 0; j < i; j++)
                {
                        if(matriz[i] < matriz[j])
                        {
                                buffer = matriz[j];
                                matriz[j] = matriz[i];
                                matriz[i] = buffer;
                        }
                }
        }
}
```

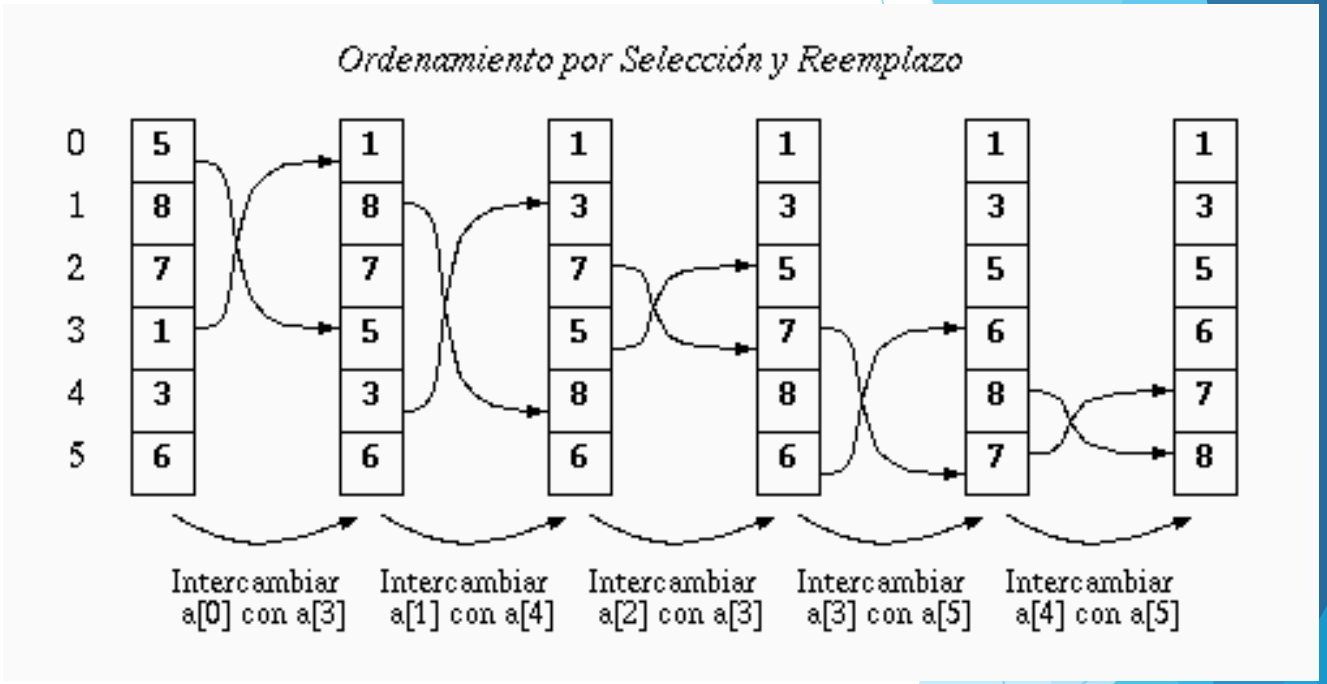
INSERCION

```
public void InsertionSort(){
    for(int i=1;i<n;i++){
        for(int j=i;j>0;j--){
            if(_insertion[j]<_insertion[j-1]){
                int swap = _insertion[j];
                _insertion[j] = _insertion[j-1];
                _insertion[j-1] = swap;
            }
        }
    }
    System.out.prin
    ImprimirVector(
    System.out.prin
    ImprimirVector(
}
```

30	15	2	21	44	8	Array original
30	15	2	21	44	8	Se empieza por el segundo elemento. Se compara con el primero. Como 15 < 30 se desplaza el 30 hacia la derecha y se coloca el 15 en su lugar
15	30	2	21	44	8	
15	30	2	21	44	8	Seguimos por el tercer elemento. Se compara con los anteriores y se van desplazando hasta que el 2 queda en su lugar.
2	15	30	21	44	8	
2	15	30	21	44	8	Continuamos por el cuarto elemento. Se compara con los anteriores y se van desplazando hasta que el 21 queda en su lugar.
2	15	21	30	44	8	
2	15	21	30	44	8	Lo mismo para el quinto elemento En este caso ya está en su posición correcta respecto a los anteriores.
2	15	21	30	44	8	
2	15	21	30	44	8	Y finalmente se coloca el último elemento El array queda ordenado
2	8	15	21	30	44	

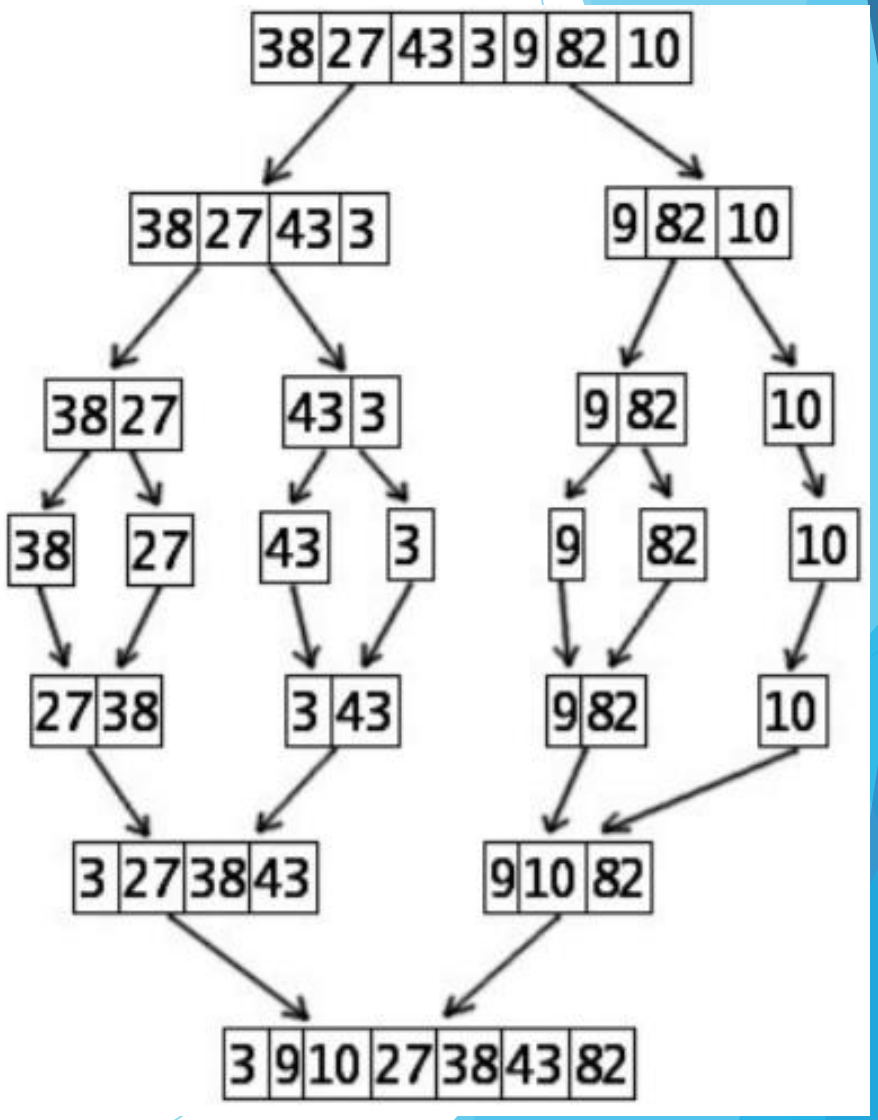
SELECTION

```
public void SelectionSort(){
    for(int i=0;i<n-1;i++){
        int min = i;
        for(int j=i+1;j<n;j++){
            if(_selection[j]<_selection[min]){
                min = j;
            }
            if(min != i){
                int swap = _selection[i];
                _selection[i] = _selection[min];
                _selection[min] = swap;
            }
        }
    }
    System.out.print("\nSelectionSort...\n---------\nOriginal: ");
    ImprimirVector(selection);
    System.out.print("Ordenado: ");
    ImprimirVector(_selection);
}
```

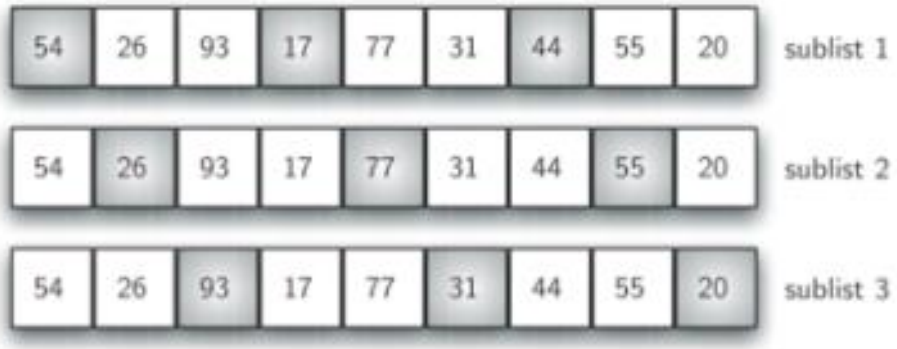


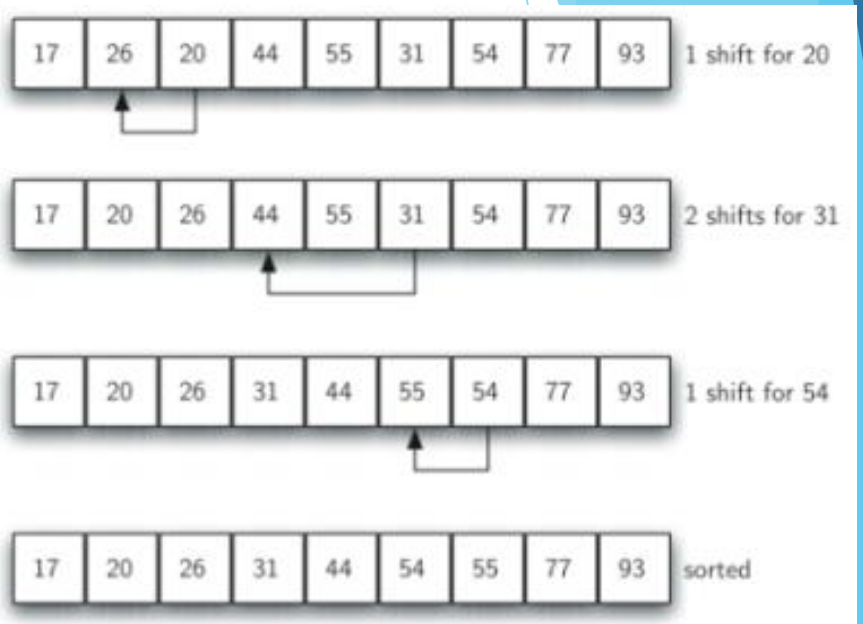
ORDENAMIENTO POR MEZCLA (MERGESORT)

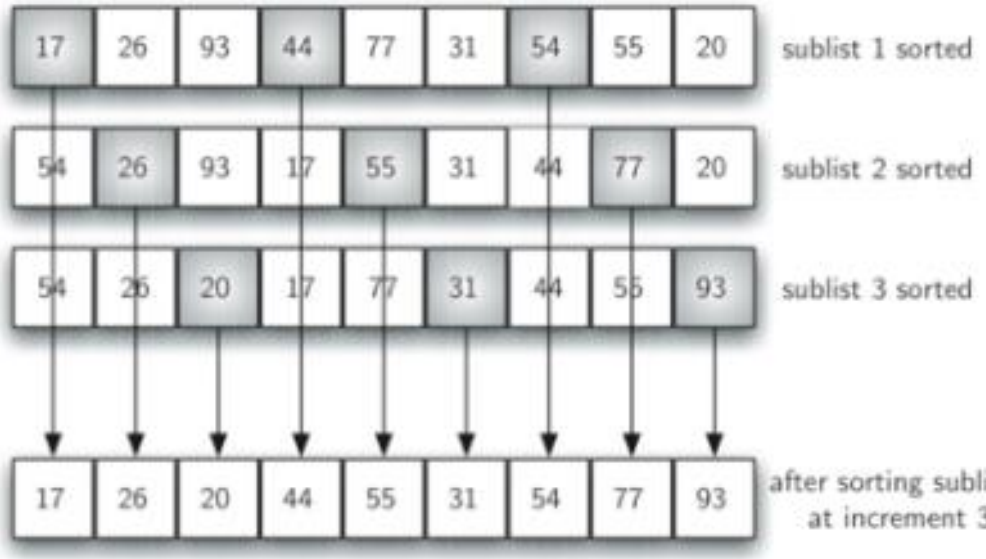
- Este algoritmo consiste básicamente en dividir en partes iguales la lista de números y luego mezclarlos comparándolos, dejándolos ordenados.
- Si se piensa en este algoritmo recursivamente, podemos imaginar que dividirá la lista hasta tener un elemento en cada lista, luego lo compara con el que está a su lado y según corresponda, lo sitúa donde corresponde.

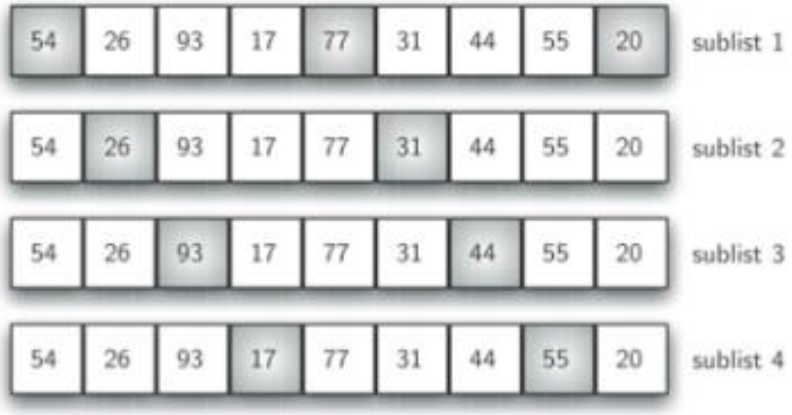


Shell Sort

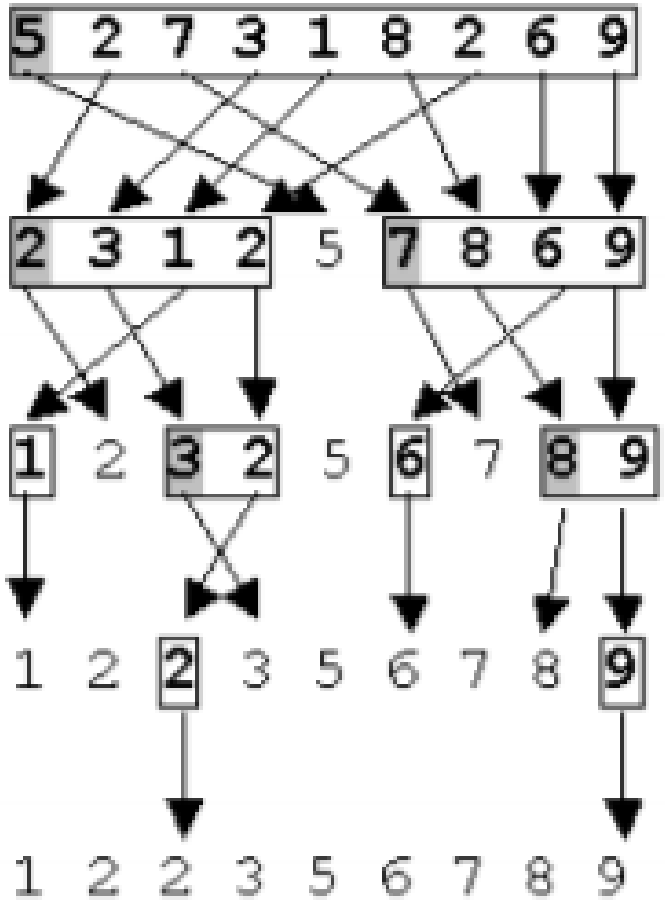

54 26 93 17 77 31 44 55 20
sublist 1
54 26 93 17 77 31 44 55 20
sublist 2
54 26 93 17 77 31 44 55 20
sublist 3


17 26 20 44 55 31 54 77 93
1 shift for 20
17 20 26 44 55 31 54 77 93
2 shifts for 31
17 20 26 31 44 55 54 77 93
1 shift for 54
17 20 26 31 44 54 55 77 93
sorted


17 26 93 44 77 31 54 55 20
sublist 1 sorted
54 26 93 17 55 31 44 77 20
sublist 2 sorted
54 26 20 17 77 31 44 55 93
sublist 3 sorted
17 26 20 44 55 31 54 77 93
after sorting sublists
at increment 3


54 26 93 17 77 31 44 55 20
sublist 1
54 26 93 17 77 31 44 55 20
sublist 2
54 26 93 17 77 31 44 55 20
sublist 3
54 26 93 17 77 31 44 55 20
sublist 4

METODO RAPIDO (Quicksort)




```
public void QuickSort(int a,int b){
    int pivote = _quick[(a+b)/2];
int i = a, j = b;
while (i <= j) {
while (_quick[i] < pivote) {
        i++;
}
while (_quick[j] > pivote) {
        j--;
    }
if (i <= j) {
        int swap = _quick[i];
    _quick[i] = quick[j];
    _quick[j] = swap;
    i++;
    j--;
}
}
if (a < j) QuickSort(a, j);
if (b > i) QuickSort(i, b);
}
```

COMPLEJIDAD

Algoritmo	Operaciones máximas
Burbuja	$\Omega(n^2)$
Insercion	$\Omega(n^2/4)$
Selección	$\Omega(n^2)$
Shell	$\Omega(n \log^2 n)$
Merge	$\Omega(n \log n)$
Quick	$\Omega(n^2)$ en peor de los casos y $\Omega(n \log n)$ en el promedio de los casos.

COMPARACION DE TIEMPOS

256 elementos	
Burbuja:	0.0040
Seleccion:	0.0030
Insercion:	0.0040
Rapido:	0.0010
Shell:	0.0010
Merge:	0.0040

512 elementos	
Burbuja:	0.0050
Seleccion:	0.0040
Insercion:	0.0050
Rapido:	0.0010
Shell:	0.0020
Merge:	0.0030

2048 elementos	
Burbuja:	0.022
Seleccion:	0.015
Insercion:	0.013
Rapido:	0.0010
Shell:	0.0060
Merge:	0.0050

16384 elementos	
Burbuja:	1.055
Seleccion:	0.9
Insercion:	0.577
Rapido:	0.0080
Shell:	0.0090
Merge:	0.014

262144 elementos	
Burbuja:	178.205
Seleccion:	158.259
Insercion:	94.461
Rapido:	0.061
Shell:	0.086
Merge:	0.201

2097152 elementos	
Burbuja:	11978.107
Seleccion:	10711.01
Insercion:	7371.727
Rapido:	0.596
Shell:	0.853
Merge:	1.327

- Como podemos analizar, el algoritmo que se va demorando cada vez mas tiempo es burbuja, luego de selección y tercero el inserción.
- Los algoritmos que los siguen son el Shell y el de ordenación por mezcla, pero el más optimo es el “Rapido”

Referencias

- https://blog.zerial.org/ficheros/Informe_Ordenamiento.pdf
- http://puntocomnoesunlenguaje.blogspot.cl/2015/02/ordenamiento-insercion-directa-java.html
- http://interactivepython.org/KKOkZ/courselib/static/pythonds/SortSearch/TheShellSort.html
- https://commons.wikimedia.org/wiki/File:Merge_sort_algorithm_diagram.svg
- https://users.dcc.uchile.cl/~lmateu/CC10A/Apuntes/ordenamiento/
- http://curriculares.weebly.com/estructura-de-datos.html