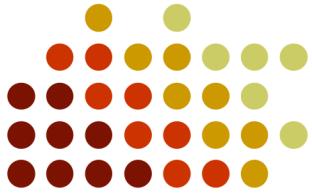


PROGRAMACION 3

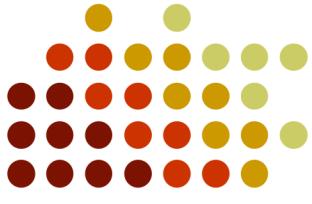
CURSADA 2022 VIRTUAL

Docentes:

Federico Améndola y Sebastián Vallejos.



Iniciar grabación



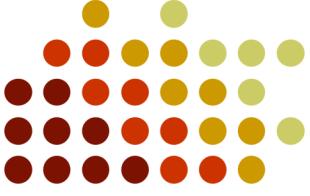
PROGRAMACION 3

TUDAI

Tema 2 - CURSADA 2021

**Algoritmos de Ordenamiento
Árboles**

Fac. Ciencias Exactas. UNICEN



El problema de ordenar

Se tiene un conjunto de elementos, sobre los cuales se establece una relación de orden.

Se quiere luego ordenarlos en forma creciente o decreciente.

Generalmente los elementos son números y la relación de orden viene dada por los operadores $<$, $>$, $=$

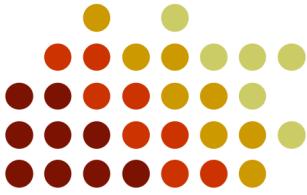
También es común requerir otros tipos de orden sobre elementos, por ejemplo el orden lexicográfico sobre palabras.

Mantener los datos ordenados nos permite en general acelerar las búsquedas en el conjunto, y brindar la posibilidad de listar ordenado.

Así es que desde la década del 50 se estudian algoritmos para ordenar conjuntos de datos en tiempos computacionales cada vez menores.

Los algoritmos que veremos no sólo difieren en cuanto a la complejidad computacional que tienen asociada, sino también difieren sobre qué estructuras de datos que mantienen el conjunto de elementos se pueden aplicar.

Ordenamiento Algoritmo de Burbujeo



Consiste en comparar pares de elementos adyacentes en un array y si están desordenados intercambiarlos hasta que estén todos ordenados.

El elemento mayor sube como una burbuja hacia la posición más alta.

6	5	3	1	8	7	2	4
5	6	3	1	8	7	2	4
5	3	6	1	8	7	2	4
5	3	1	6	8	7	2	4
5	3	1	6	7	8	2	4
5	3	1	6	7	2	8	4
5	3	1	6	7	2	4	8

Ordenamiento

Algoritmo de Burbujeo

Consiste en comparar pares de elementos adyacentes en un array y si están desordenados intercambiarlos hasta que estén todos ordenados. El elemento mayor sube como una burbuja hacia la posición más alta.

```
public static void burbujeo(int [ ] A) {  
    int i, j, aux;  
    for (i=0; i < A.length - 1; i++)  
        for (j=0; j < A.length - i - 1 ; j++)  
            if (A[j] > A[j+1]) {  
                aux = A[j+1];  
                A[j+1] = A[j];  
                A[j] = aux;  
            }  
    }  
  
    public void bubbleSortAdapt (int [ ] arr) {  
        boolean swapped = true;  
        int j = 0;  
        int tmp;  
        while (swapped) {  
            swapped = false;  
            j++;  
            for (int i=0; i<arr.length - j; i++) {  
                if (arr[i] > arr[i + 1]) {  
                    tmp = arr[i];  
                    arr[i] = arr[i + 1];  
                    arr[i + 1] = tmp;  
                    swapped = true;  
                }  
            }  
        }  
    }  
}
```

O(n^2)

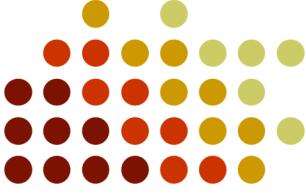
Mejorado (Adaptativo)

Para caso por ej:

8 5 7 9 11 15 25 32 41 50

Ordenamiento

Algoritmo Mergesort



Se pueden mezclar eficientemente dos secuencias **ordenadas** de tamaño $n/2$ en otra de tamaño n que también resulte ordenada ?
Cómo lo haría ?

1-12-14-15-25-30

4-5-10-17-20-45

1-4-5-10-12-14-15-17-20-25-30-45

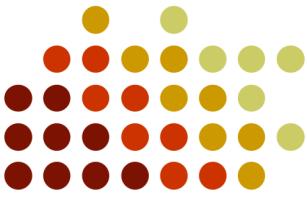
El tiempo que tarda (complejidad temporal) dependerá de la cantidad de elementos total n ? **$O(n)$**

Entonces si sabemos mezclar eficientemente dos mitades ordenadas. Podríamos repetir el algoritmo y decir que el problema de ordenar un array de tamaño n , se puede resolver si recursivamente ordenamos la primer mitad de $n/2$ elementos, luego la segunda mitad de $n/2$ elementos y luego las mezclamos para obtener el array ordenado de n elementos.

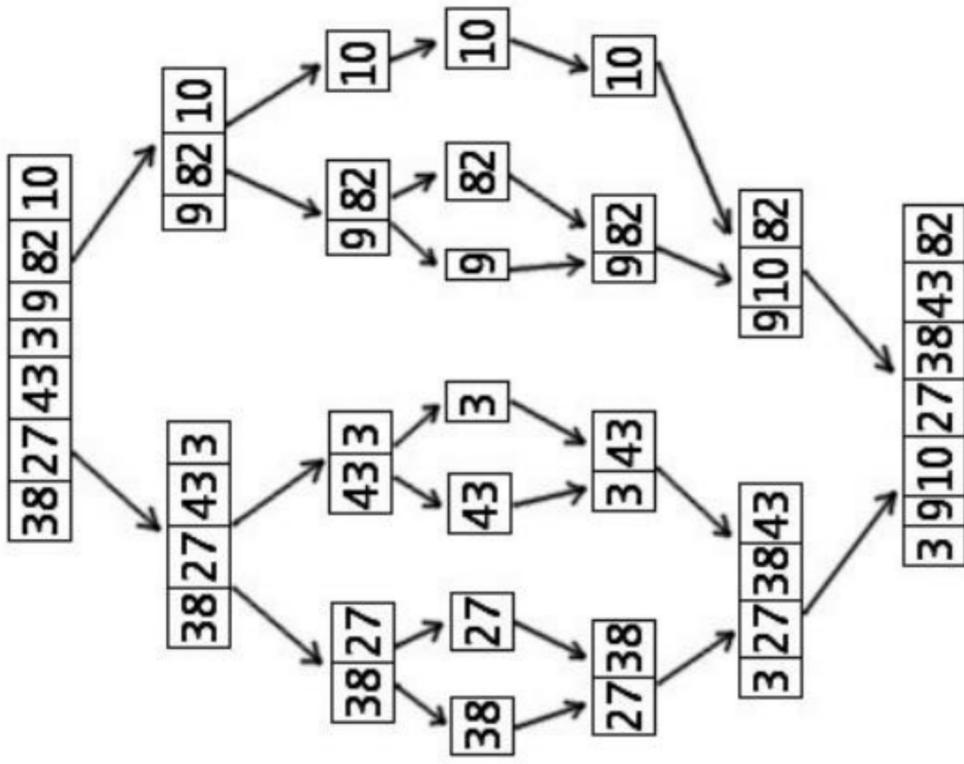
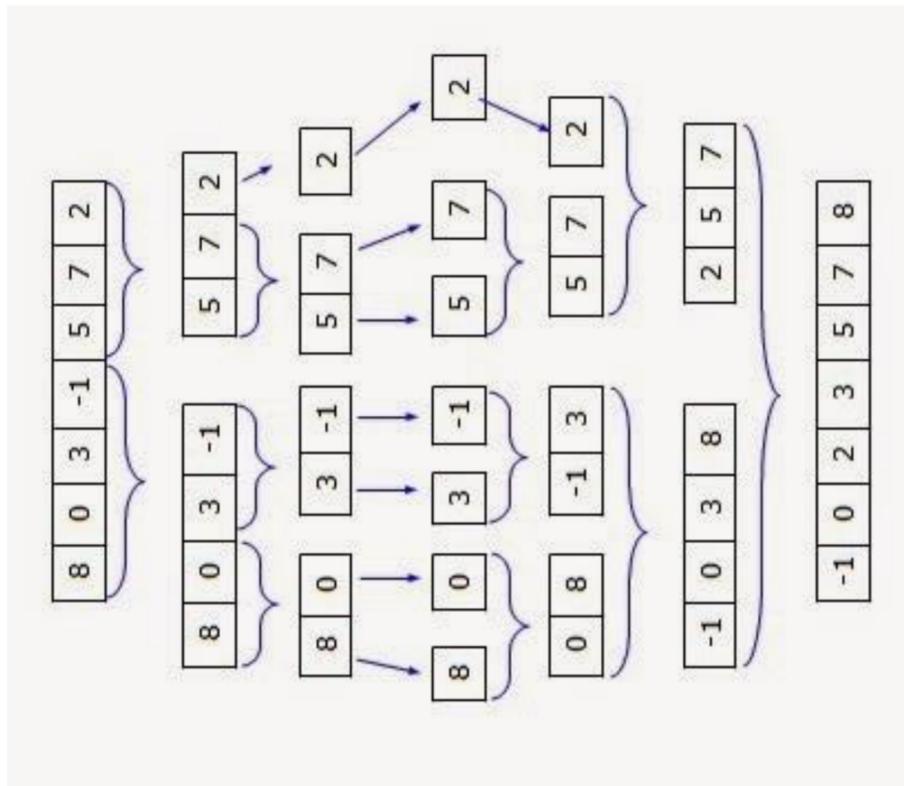
El caso base (o de corte) cuál será ?

Será cuando lleguemos a un array de un solo elemento.

Cuántas veces se puede ir dividiendo por 2 hasta obtener el caso base ? **$\log_2 n$**



Ordenamiento Algoritmo Mergesort



Tanto para este algoritmo como para los otros de ordenamiento les recomiendo ver animaciones tales como:

Ordenamiento Algoritmo Mergesort

```
private void merge(int low, int middle, int high) {
```

```
    // copiar ambas partes al array helper
    for (int i = low; i <= high; i++) {
        helper[i] = numbers[i];
    }

    int i = low;
    int j = middle + 1;
    int k = low;
    // copiar de manera ordenada al array original los valores de la
    // mitad izquierda o de la derecha
    while (i <= middle && j <= high) {
        if (helper[i] <= helper[j]) {
            numbers[k] = helper[i];
            i++;
        } else {
            numbers[k] = helper[j];
            j++;
        }
        k++;
    }
    // si quedaron elementos copiarlos al array original
    while (i <= middle) {
        numbers[k] = helper[i];
        k++;
    }
    while (j <= high) {
        numbers[k] = helper[j];
        k++;
        j++;
    }
}
```

Requiere memoria

auxiliar

```
public class Mergesort {
```

```
    private int[] numbers;
```

```
    private int[] helper;
```

```
    private int size;
```

```
    public void sort(int[] values) {
```

```
        this.numbers = values;
```

```
        size = values.length;
```

```
        this.helper = new int[size];
```

```
        mergesort(0, size - 1);
    }
```

```
    private void mergesort(int low, int high) {
```

```
        // si low es menor que high continua el ordenamiento
```

```
        // si low no es menor que high entonces el array está ordenado
```

```
        // ya que es el caso base donde el array tiene un solo elemento.
```

```
        if (low < high) {
```

```
            // obtener el indice del elemento que se encuentra en la mitad
```

```
            // al ser int redondea el resultado al entero menor
```

```
            int middle = (low + high) / 2;
```

```
            // ordenar la mitad izquierda del array
```

```
            mergesort(low, middle);
```

```
            // ordenar la mitad derecha del array
```

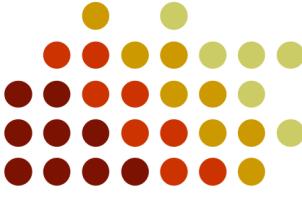
```
            mergesort(middle + 1, high);
```

```
            // combinar ambas mitades ordenadas
```

```
            merge(low, middle, high);
        }
    }
```

$O(n \log_2 n)$

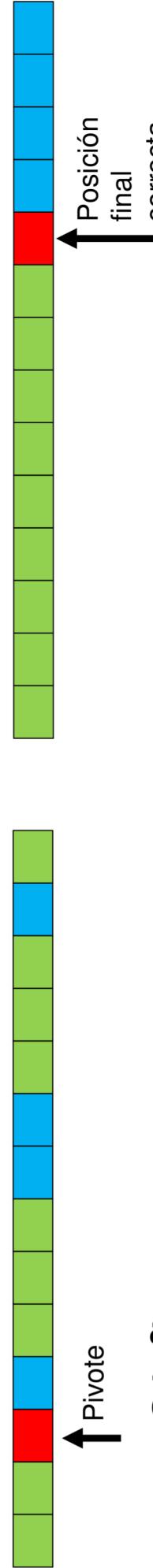
Ordenamiento Algoritmo Quick-Sort



El algoritmo consiste en seleccionar uno de los elementos del array como valor pivot alrededor del cual el resto de los elementos serán reordenados. Todo lo que sea menor que el pivot es movido a la izquierda del pivot (a la partición izquierda). De forma similar todo lo que sea mayor que el pivot es movido a la partición derecha.

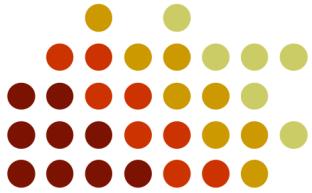
Notar que el pivot quedó en el lugar correcto del array final ordenado.

Una vez hecho esto realizar un QuickSort de cada partición.



$\Omega(n \log_2 n)$ $O(n^2)$ *No requiere memoria adicional !*

Buscar e implementar !



Detener grabación