



# Estructuras de Datos

## IIS262

Profesor: Patricio Galeas

# CAPÍTULO 3 : Ordenamiento Simple



# Introducción

- El ordenamiento de datos **es una tarea común**:
  - Nombres
  - Estudiantes por curso
  - Clientes por código postal
  - Países por su PIB, etc.
- El ordenamiento de los datos es el **paso preliminar a la búsqueda** (búsqueda binaria).
- El ordenamiento ha sido un tema **importante en IT**, => bastante investigación al respecto.
- Revisaremos los **tres algoritmos** de búsqueda simple:
  - Burbuja
  - Selección
  - Inserción

# ¿cómo ordenar por tamaño?



- El cerebro humano tiene algunas ventajas, generando una vista general de todo el cuadro ...
  - Seleccionar a los niños,
  - Seleccionar al más alto, etc.
  - No necesitamos hacer comparaciones entre cada uno de los elementos/personas.
- Un computador no es capaz de generar una vista general del problema ...
  - Está limitado por la comparación de sólo dos elementos.
  - Esta es la naturaleza del operador de comparación (=).

# Procedimiento Básico

- **Comparar** dos elementos

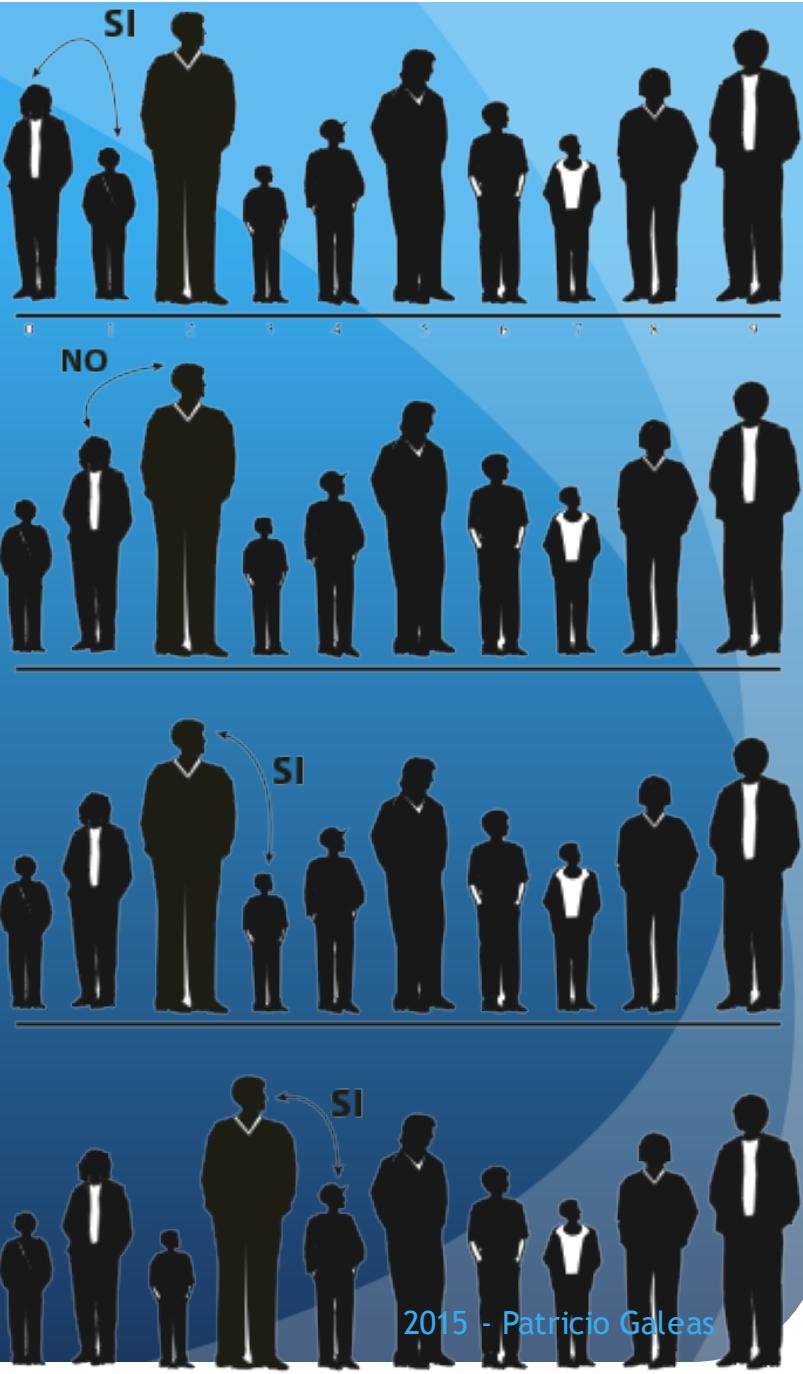


- **Intercambiar** dos elementos o **copiar** un elemento.



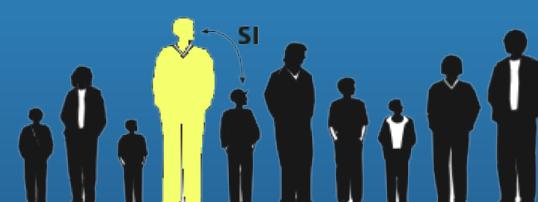
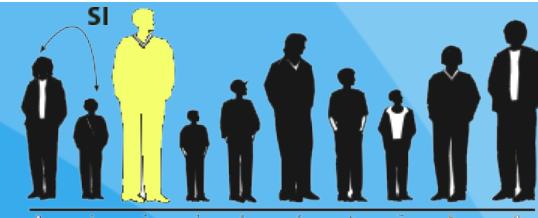
# Ordenamiento de la Burbuja

- Se parte por la izquierda
- Se toman dos elementos a la vez
- Se comparan los dos primeros elementos (pos. 0 y pos. 1)
  - a) Si el primero es mas alto, se intercambia.
  - b) Si el de la derecha es mas alto, no se hace nada.
- Se continua hacia la derecha con los dos elementos siguientes (pos 1 y pos. 2) aplicando (a) y (b)
- Se sigue igual hasta alcanzar el final  
...



# Ordenamiento de la Burbuja

- Después de la **1<sup>a</sup> iteración**, las personas **todavía no** están ordenadas.
- Pero, la persona más alta ya se encuentra la final del arreglo.
- Una vez que se encuentra a la persona más alta, se le va intercambiando hasta ubicarla en la posición final.
- El elemento más grande **flota** hacia la superficie como una **burbuja**



# Ordenamiento de la Burbuja

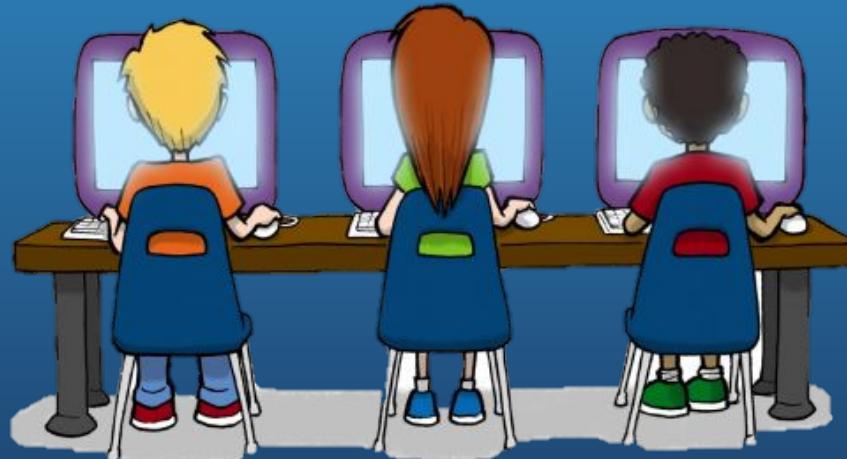


- En la 2<sup>a</sup> iteración, se toman N-2 elementos
- Se sigue iterando hasta que todos los elementos estén ordenados.



# Ordenamiento de la Burbuja

- BubbleSort.java



# Ordenamiento de la Burbuja

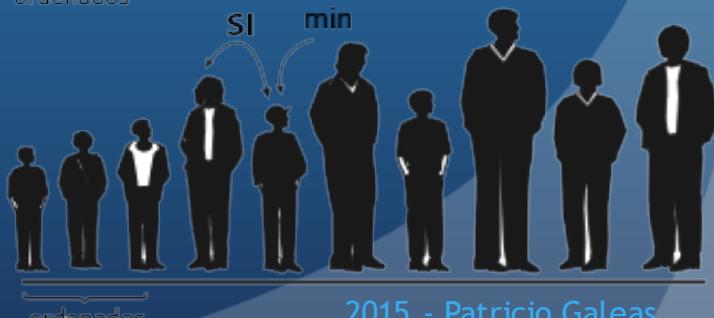
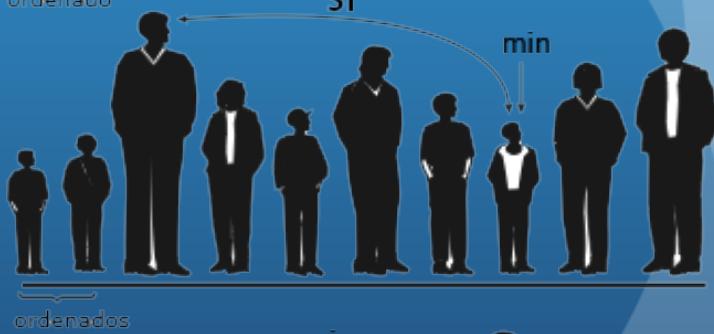
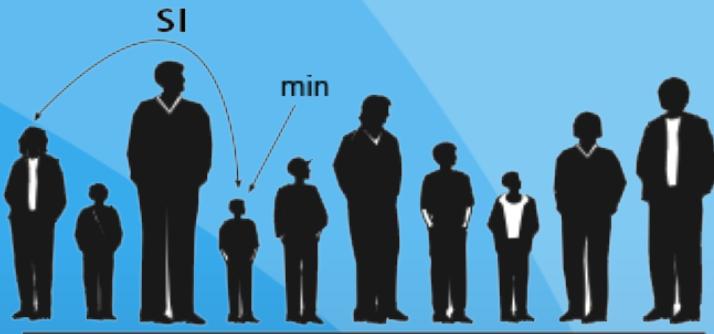
## Eficiencia

- El ordenamiento de 10 barras requiere
  - 9 comparaciones en la 1<sup>a</sup> vuelta
  - 8 comparaciones en la 2<sup>a</sup> vuelta
  - ...
  - 1 comparación en la 9<sup>a</sup> vuelta
- En total :  $9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 45$
- Si N es el número de elementos :  
$$(N-1) + (N-2) + (N-3) + \dots + 1 = N*(N-1)/2$$
- El algoritmo hace aprox.  $N^2/2$  comparaciones.  
En promedio  $(N^2/4)$
- El ordenamiento de la burbuja tiene un orden:  $O(N^2)$



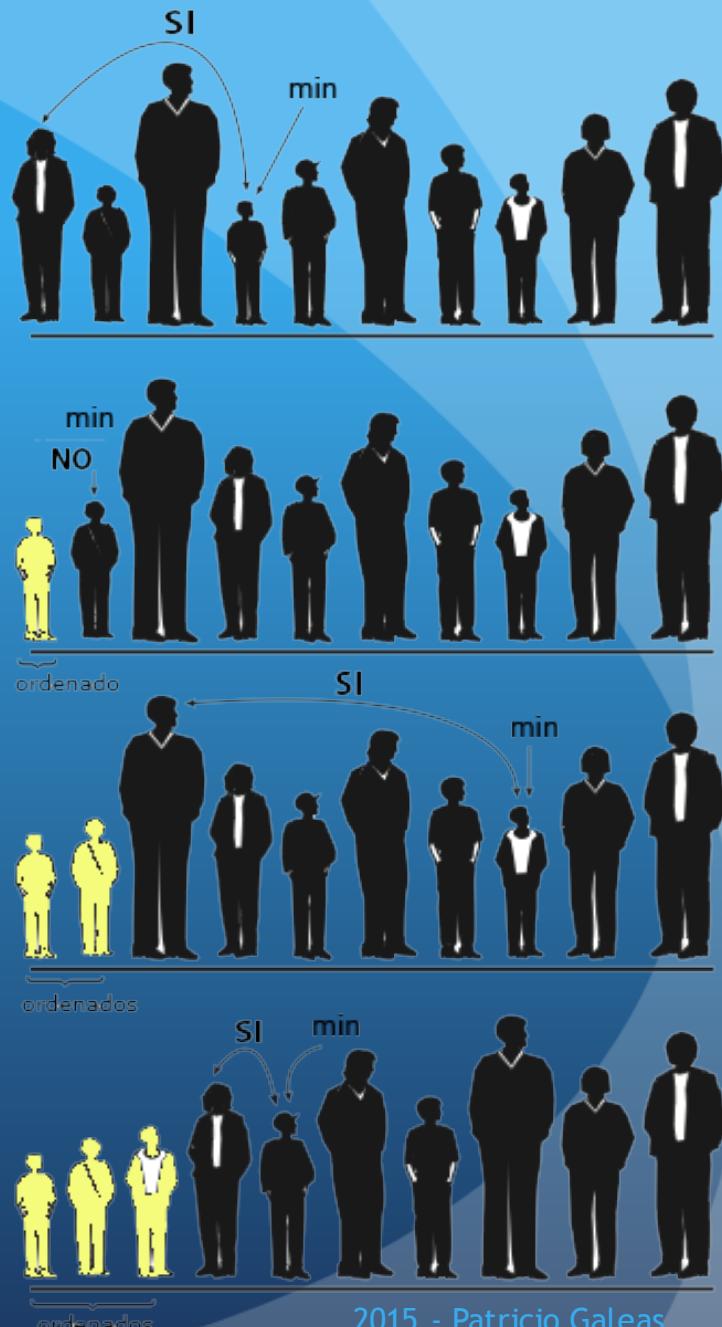
# Ordenamiento por Selección

- Mejora al de la burbuja, reduciendo **intercambios** de  $O(N^2)$  a  $O(N)$ . Pero desafortunadamente el número de **comparaciones** permanece  $O(N^2)$ .
- Se parte de la **izquierda**
- Se **anota** la altura del primer elemento ( $x$ ) y se busca **al menor en el resto** de los elementos ( $min$ ).
  - a) Si  $x > min$ , se intercambia
  - b) Si  $x \leq min$ , se mantiene la posición.
- Se continua con el siguiente elemento, hasta que todos estén ordenados.



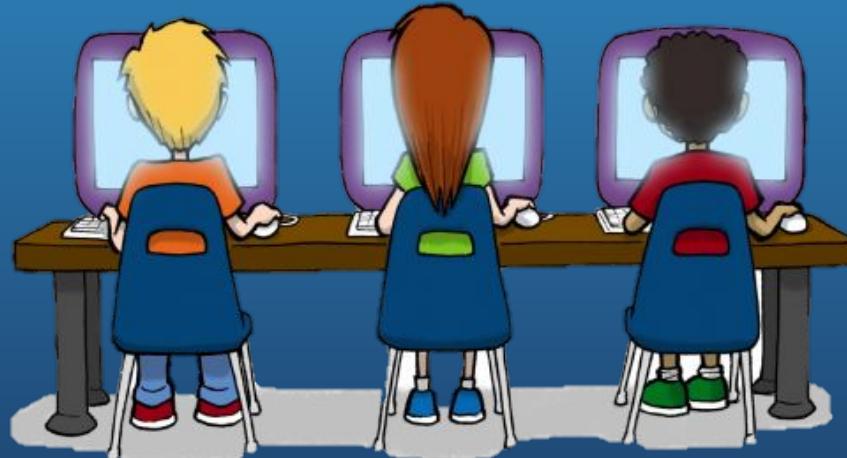
# Ordenamiento por Selección

- A diferencia de Burbuja, los elementos ordenados van quedando a la izquierda.
- Sólo se intercambia si hay un elemento menor.



# Ordenamiento por Selección

- SelectSort.java



# Ordenamiento por Selección

## Eficiencia

- Hace el mismo número de comparaciones que el de la burbuja:  $N(N-1)/2 \Rightarrow O(N^2)$
- Pero, el número de intercambios es en el peor caso  $N \Rightarrow O(N)$ .
- Cuando hay **pocos elementos**, este algoritmo es **considerablemente más rápido que el de la burbuja**.  
Especialmente si el tiempo de intercambio > tiempo de comparaciones.



# Ordenamiento por Inserción

- Es el algoritmo **más rápido** de los que hemos revisado.
- A pesar de que tiene un orden  $O(N^2)$ , es casi 2 veces **más rápido** que el de la **Burbuja** y algo más rápido que el de **Selección**
- A pesar de ser caracterizado como simple, su **complejidad es un poco mayor** a lo dos anteriores.
- Se utiliza normalmente como la **última etapa** de algoritmos más sofisticados como **quicksort**.



# Ordenamiento por Inserción

- La forma más fácil de explicarlo es con el applet (100 barras).
- Más detalles (10 barras)
  - (inner)/(outer) parten en pos 1
  - Copia (outer) a (temp).
  - Hace el espacio en N-1
- Si (inner-1) > temp
  - Desplaza (inner-1) a (inner)
- Sino
  - Copia (temp) a (inner)



# Ordenamiento por Inserción

## Eficiencia

- Comparaciones :  $1 + 2 + 3 + \dots + (N-1) = N(N-1)/2$
- Pero como en promedio solo comparamos con la mitad de los items =>  $N(N-1)/4$
- El # de copias es casi igual el # de comparaciones.
- Pero, copiar es mucho mas rápido que intercambiar =>, ordenamiento por Inserción es casi el doble más rápido que Burbuja y más rápido que Selección.
- Para datos aleatorios:  $O(N^2)$
- Para datos casi ordenados: casi  $O(N)$
- Cuando los datos están orden inverso : Inserción no es más rápido que Burbuja



# Resumen

- El ordenamiento considera la **comparación** y **movimiento** de items (referencias a estos) en un arreglo.
- Los algoritmos presentados tienen **todos un orden**  $O(N^2)$ . Sin embargo hay **algunos que son más rápidos** que otros.
- Los **invariantes** son condiciones que no cambian durante la ejecución del algoritmo.
- El ordenamiento de **Burbuja** es el **menos eficiente**, pero el **más simple**.
- El ordenamiento **por inserción** es el **más usado** de los  $O(N^2)$
- **Ninguno** de los algoritmos presentados **requiere** más de una **variable temporal**, aparte del arreglo.



# Experimentos

- Reescribir el main() de bubbleSort.java, de modo de poder **ingresar una gran cantidad de items (aleatorios)**. Ej: 10.000 items.  
Estimar el tiempo de ejecución y hacer lo mismo con los otros dos algoritmos.
- Generar un **arreglo en orden inverso** para comparar los tiempos con el arreglo original (de los ejemplos). Comparar que pasa en cada algoritmo.
- Genere un **arreglo ordenado** y compare los tiempos con el arreglo original. Compara que pasa en cada algoritmo.

