Algoritmos de Ordenamiento

mat-151

Algoritmos de ordenamiento

Entrada: secuencia $<a_1, a_2, \ldots, a_n>$ de números.

Salida: permutación $<a'_1, a'_2, \ldots, a'_n>$ tal que $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

Ejemplo:

entrada: 8 2 4 9 3 6

salida: 2 3 4 6 8 9

Algortimos de ordenamiento

- algoritmos para arreglar elementos de una **lista** en cierto **orden**.
- los ordenes más comunes son **numérico** y **lexicográfico**.

  El orden lexicográfico no es igual al orden numérico

  Si a = [19], b = [138] tenemos b < a, porque el prefijo es a1 = b1 = 1 y b2 = 3 < a2 = 9

- la salida debe satisfacer dos condiciones:
  - la salida debe encontrarse en **orden no-decreciente**.
  - la salida es una **permutación** o **re-ordenamiento** de la entrada.
- los elementos pueden ordenarse respecto a una o más ***llaves*** o ***claves (keys)***
  - clave primaria
  - claves secundarias

Algoritmos de ordenamiento

- *Un método de ordenamiento se dice **estable** si conserva el orden relativo de los elementos con **claves duplicadas** en el archivo.*

- Ejemplo:

llave (key)

Aviña	1
Burrón	2
Barranco	4
Jiménez	2
Jaramillo	4
López	1
Martínez	4
Miranda	2
Nuñez	3
Pérez	3

Aviña	1
López	1
Miranda	2
Burrón	2
Jiménez	2
Pérez	3
Nuñez	3
Martínez	4
Barranco	4
Jaramillo	4

Aviña	1
López	1
Burrón	2
Jiménez	2
Miranda	2
Nuñez	3
Pérez	3
Barranco	4
Jaramillo	4
Martínez	4

Algoritmos de ordenamiento

- Una medida útil para analizar la entrada del algoritmo es el ***número de inversiones*:**

  - el número de pares de enteros ***(i,j)***, tales que ***i<j*** y $k_i > k_j$.

  - Ejemplos:

    - la secuencia **Charlie, Alpha, Bravo** tiene dos inversiones.
    - la secuencia **Charlie, Bravo, Alpha** tiene tres inversiones.

- cuando las claves están en orden, el número de inversiones es ***0***.

- cuando están en orden reverso (i.e. cada par está en el orden equivocado) el número de inversiones es ***N(N-1)/2***, que es el número total de pares de claves.

Algoritmos de ordenamiento

- ***O(n^2)***
  - bubble sort
  - selection sort
  - insertion sort
  - shell sort

O(n log n)

- quicksort
- mergesort
- heapsort

Bubble sort

- ***Idea:***
  - Recorrer un conjunto de datos comparando pares de elementos. Si el primero es mayor al segundo, invertir sus lugares. Repetir el procedimiento hasta recorrer todo el conjunto.
  - Opcionalmente. El procedimiento se repite hasta que en la iteración anterior no se encuentran operaciones de intercambio.
- ***Ejemplo:***
  - **5 1 4 2 8**

Bubble sort

- ¿qué pasa con los elementos grandes? ¿y con los pequeños?
  - los grandes van muy rápido a su lugar en el orden (burbujas, conejos)
  - los pequeños se hunden lentamente (tortugas)
- ¿cuál es el mejor caso?
  - cuando los elementos están ordenados
- ¿cuál es su complejidad?
  - cerca de $O(n^2)$

Bubble sort

- ***Ventajas:***
    - Es muy **simple** y **fácil** de implementar

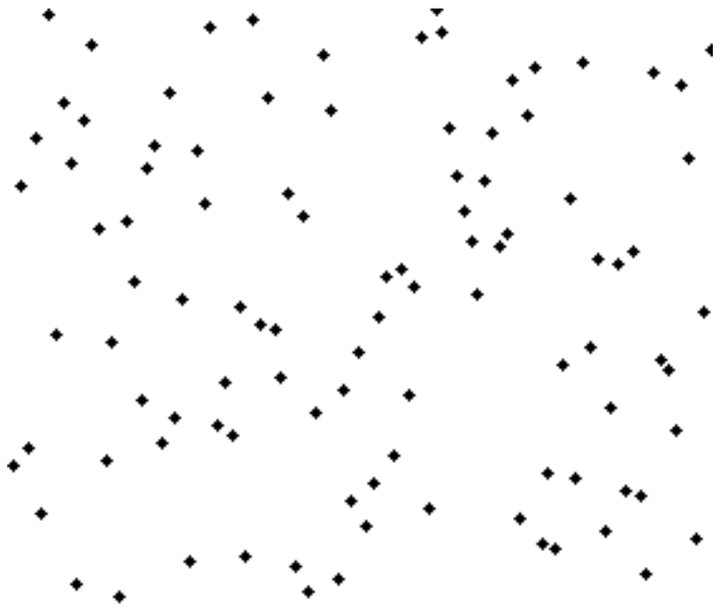
- ***Desventajas:***
    - es el algoritmo más lento e ineficiente entre los algoritmos de ordenamiento.

Bubble sort: ejemplo de implementación

```
void bubbleSort( Item a[], int l, int r )
{
    for( int i=r; i>=l; i--)
        for( int j=l+1; j<=i; j++)
            {
                if( a[j-1] > a[j]){
                    temp = a[j-1];
                    a[j-1] = a[j];
                    a[j] = temp;
            }
    }
```

Bubble sort: ejemplo de implementación

```
void bubbleSort( Item a[], int l, int r )
{
    for( int i=r; i>=l; i--)
        for( int j=l+1; j<=i; j++)
            {
                if( a[j-1] > a[j]){
                    temp = a[j-1];
                    a[j-1] = a[j];
                    a[j] = temp;
            }
    }
```



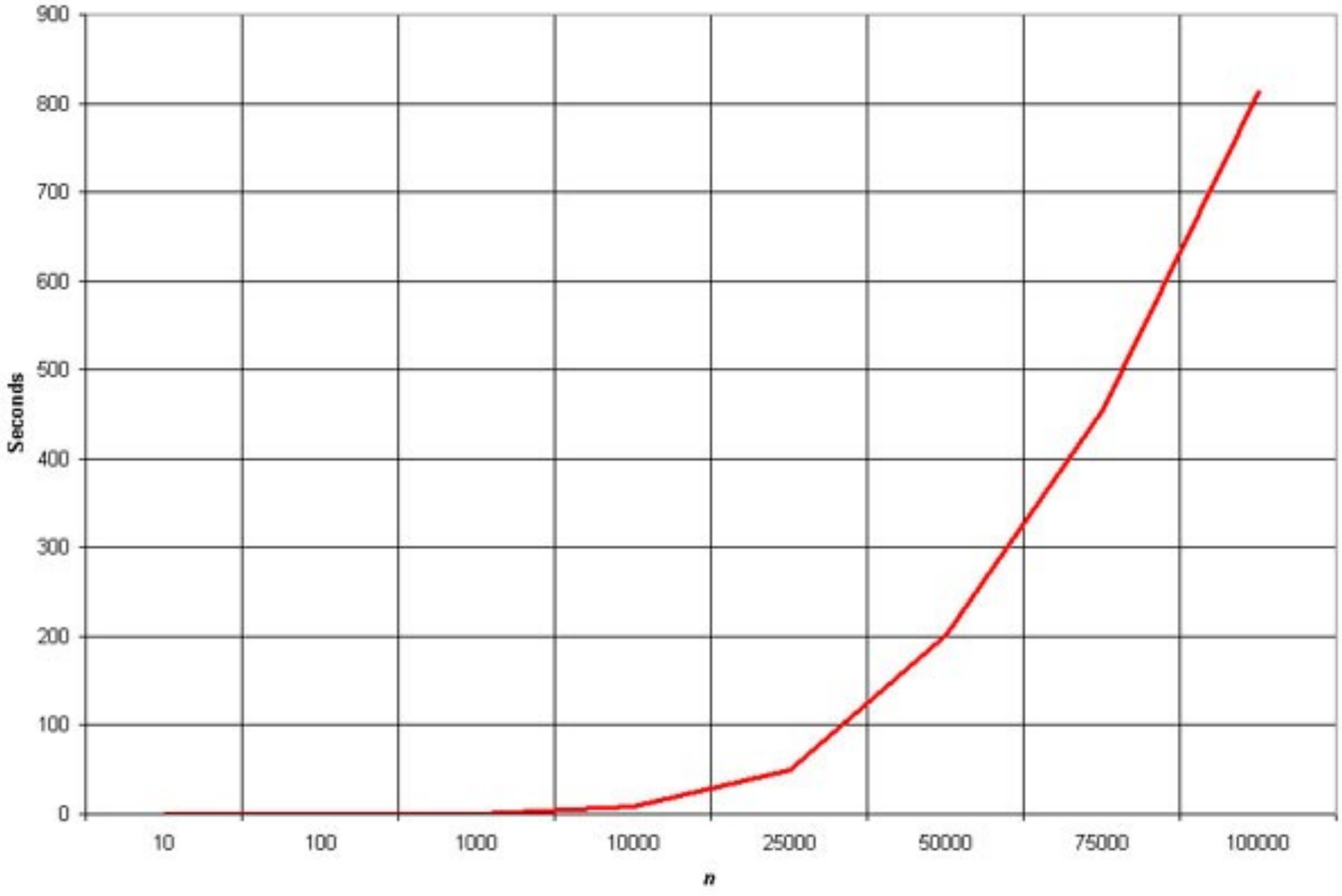
Bubble sort

A	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	A	S	O	R	T	I	N	G	E	X	E	M	P	L
A	A	E	S	O	R	T	I	N	G	E	X	L	M	P
A	A	E	E	S	O	R	T	I	N	G	L	X	M	P
A	A	E	E	G	S	O	R	T	I	N	L	M	X	P
A	A	E	E	G	I	S	O	R	T	L	N	M	P	X
A	A	E	E	G	I	L	S	O	R	T	M	N	P	X
A	A	E	E	G	I	L	M	S	O	R	T	N	P	X
A	A	E	E	G	I	L	M	N	S	O	R	T	P	X
A	A	E	E	G	I	L	M	N	O	S	P	R	T	X
A	A	E	E	G	I	L	M	N	O	P	S	R	T	X
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Figure 6.4
Bubble sort example

Aqui esta invertido, primero se ordenan los de llave mas pequeña, pero es exactamente la misma idea

Bubble sort



http://linux.wku.edu/~lamonml/algor/sort/bubble.html

Selection sort

Idea:

Recorrer un conjunto de datos encontrando el elemento más pequeño.

Ponerle al principio del conjunto ordenado.

Repetir el procedimiento con los elementos restantes del conjunto.

Ejemplo:

8 5 2 6 9 3 1 4 0 7

Selection sort

Idea:

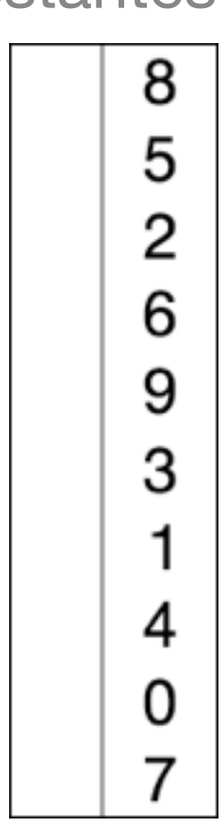
Recorrer un conjunto de datos encontrando el elemento más pequeño.

Ponerle al principio del conjunto ordenado.

Repetir el procedimiento con los elementos restantes del conjunto.

Ejemplo:

8 5 2 6 9 3 1 4 0 7



Selection sort

- Su tiempo de cálculo no depende del orden de los elementos de entrada.
- ¿Cuántas ***comparaciones*** requiere?

  •$(n-1)+(n-2)+(n-3)+...+2+1 = n(n-1)/2 = \Theta(n^2)$

- ¿Cuántos ***intercambios*** requiere?

  •$(n-1) = \Theta(n)$

- Se puede usar en arreglos o en listas ligadas, borrando e insertando al elemento en el primer lugar.

Selection sort

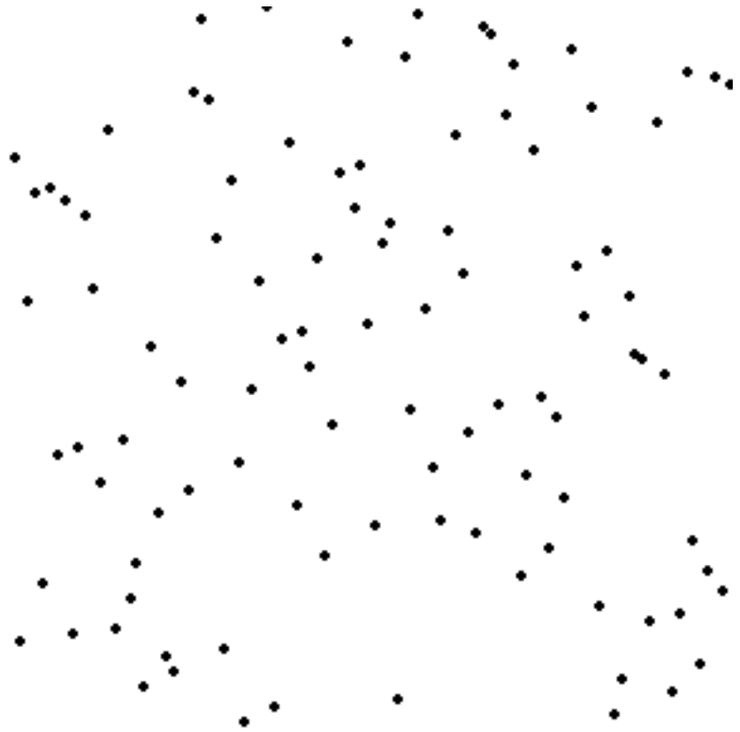
- ***Ventajas:***
    - Es muy **simple** y **fácil** de implementar.
    - Es alrededor de 60% más rápido que bubble sort en el peor caso.
    - Es útil en el caso que la escritura sea la operación más costosa.
- ***Desventajas:***
    - Es ineficiente para conjuntos grandes de elementos.

Selection sort: ejemplo de implementación

```
void selectionSort( Item a[], int l, int r )
{
    for( int i=l; i<r; i++ )
    {
        int min=i;
        for( int j=i+1; j<=r; j++ ){
            if( a[j] < a[min])
                min = j;
        }
        temp = a[i];
        a[i] = a[min];
        a[min] = temp;
    }
}
```

Selection sort: ejemplo de implementación

```
void selectionSort( Item a[], int l, int r )
{
    for( int i=l; i<r; i++ )
    {
        int min=i;
        for( int j=i+1; j<=r; j++ ){
            if( a[j] < a[min])
                min = j;
        }
        temp = a[i];
        a[i] = a[min];
        a[min] = temp;
    }
}
```

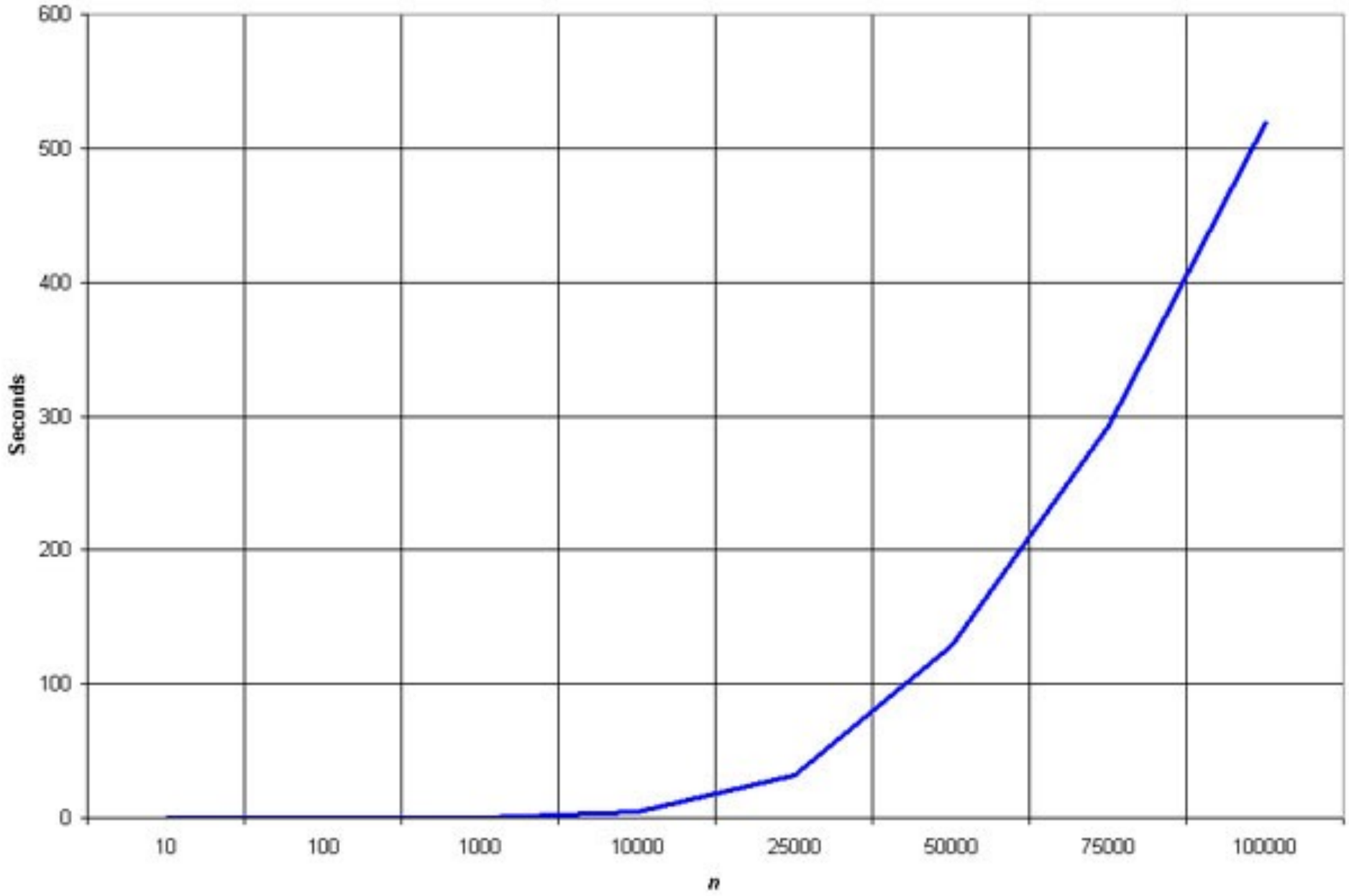


Selection sort

(A)	S	O	R	T	I	N	G	E	X	A	M	P	L	E
A	S	O	R	T	I	N	G	E	X	(A)	M	P	L	E
A	A	O	R	T	I	N	G	(E)	X	S	M	P	L	E
A	A	E	R	T	I	N	G	O	X	S	M	P	L	(E)
A	A	E	E	T	I	N	(G)	O	X	S	M	P	L	R
A	A	E	E	G	(I)	N	T	O	X	S	M	P	L	R
A	A	E	E	G	I	N	T	O	X	S	M	P	(L)	R
A	A	E	E	G	I	L	T	O	X	S	(M)	P	N	R
A	A	E	E	G	I	L	M	O	X	S	T	P	(N)	R
A	A	E	E	G	I	L	M	N	X	S	T	P	(O)	R
A	A	E	E	G	I	L	M	N	O	S	T	(P)	X	R
A	A	E	E	G	I	L	M	N	O	P	T	S	X	(R)
A	A	E	E	G	I	L	M	N	O	P	R	(S)	X	T
A	A	E	E	G	I	L	M	N	O	P	R	S	X	(T)
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X
A	A	E	E	G	I	L	M	N	O	P	R	S	T	X

Figure 6.2
Selection sort example

Selection sort



Insertion sort

- ***Idea:***
  - Tomar los elementos de la lista uno a uno e insertarlos el orden correspondiente.
  - ***Ejemplo:***

Insertion sort

- ***Idea:***
  - Tomar los elementos de la lista uno a uno e insertarlos el orden correspondiente.
  - ***Ejemplo:*** 8 2 4 9 3 6

Insertion sort

- ***Idea:***
  - Tomar los elementos de la lista uno a uno e insertarlos el orden correspondiente.
  - ***Ejemplo:***

Insertion sort

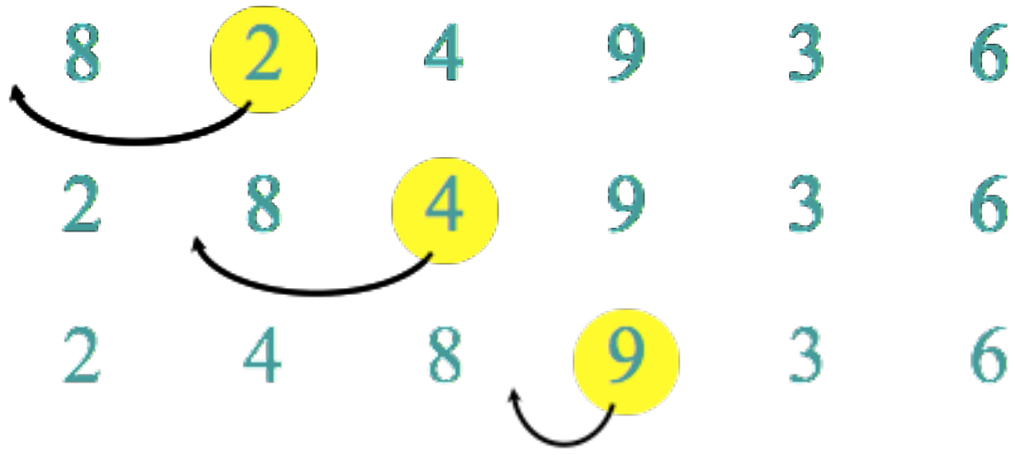
- ***Idea:***
  - Tomar los elementos de la lista uno a uno e insertarlos el orden correspondiente.
  - ***Ejemplo:***

Insertion sort

- ***Idea:***
  - Tomar los elementos de la lista uno a uno e insertarlos el orden correspondiente.
  - ***Ejemplo:***

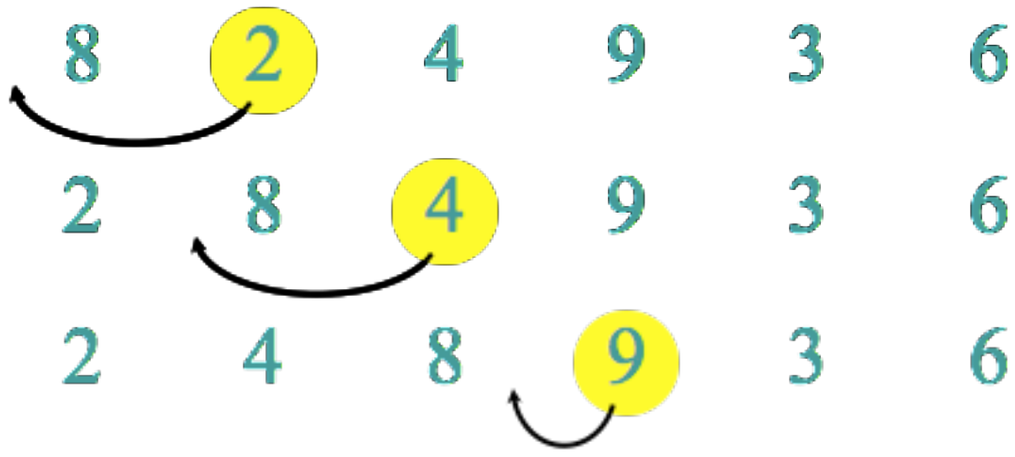
Insertion sort

- ***Idea:***
  - Tomar los elementos de la lista uno a uno e insertarlos el orden correspondiente.
  - ***Ejemplo:***



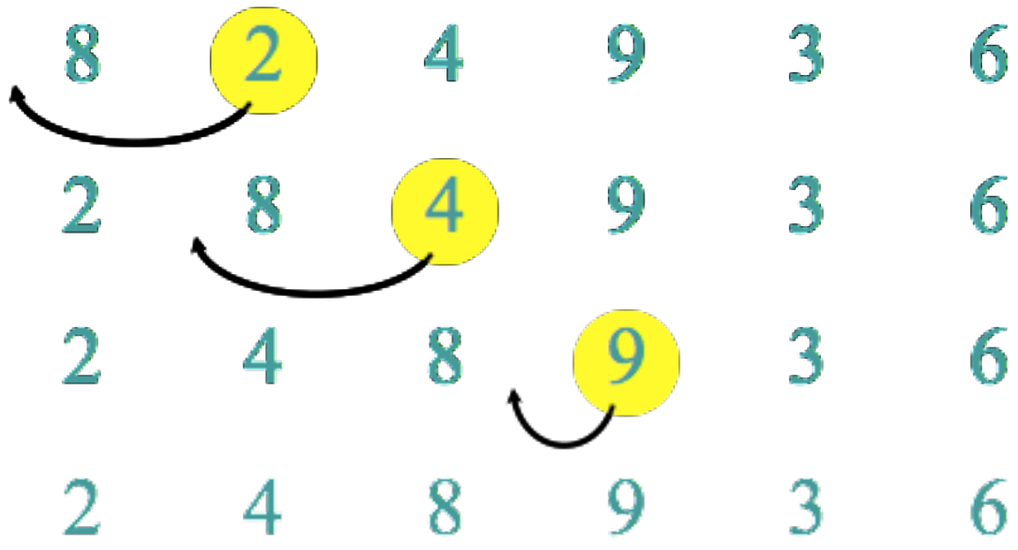
Insertion sort

- ***Idea:***
  - Tomar los elementos de la lista uno a uno e insertarlos el orden correspondiente.
  - ***Ejemplo:***



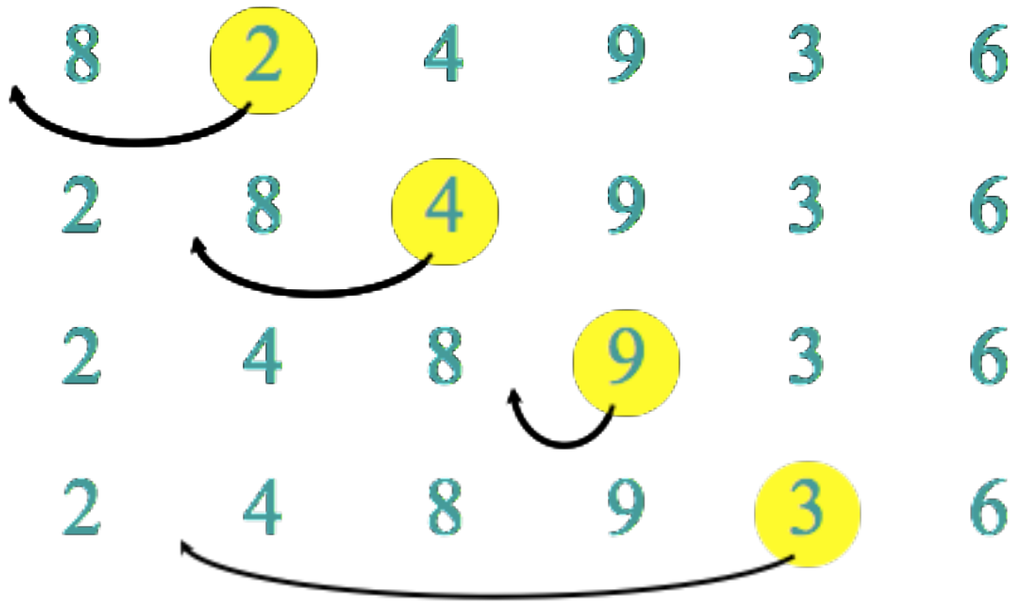
Insertion sort

- ***Idea:***
  - Tomar los elementos de la lista uno a uno e insertarlos el orden correspondiente.
  - ***Ejemplo:***



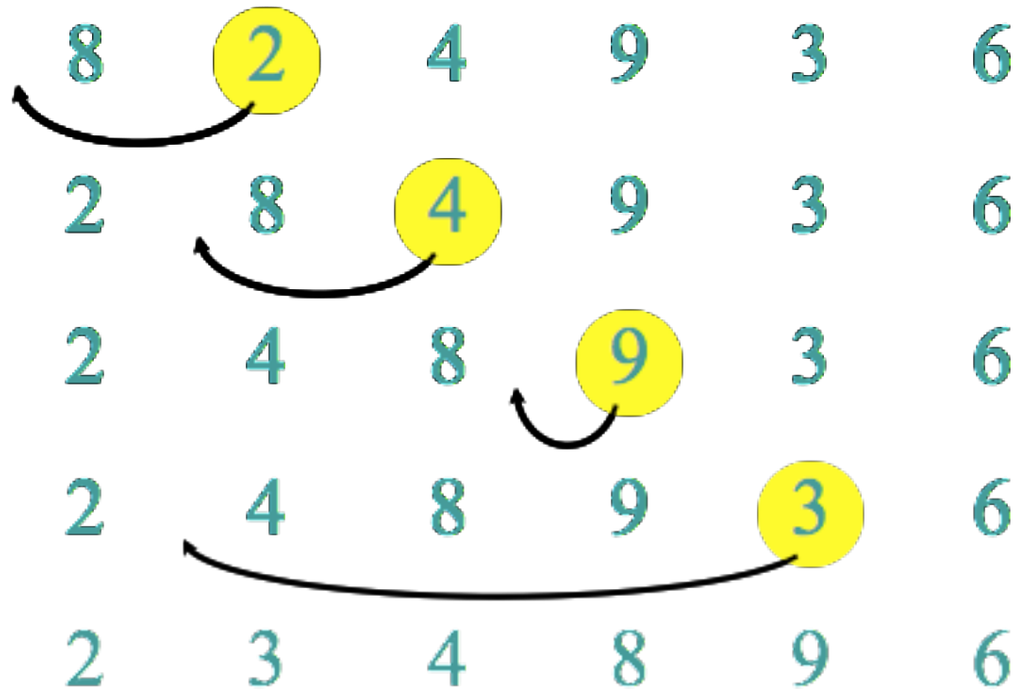
Insertion sort

- ***Idea:***
  - Tomar los elementos de la lista uno a uno e insertarlos el orden correspondiente.
  - ***Ejemplo:***



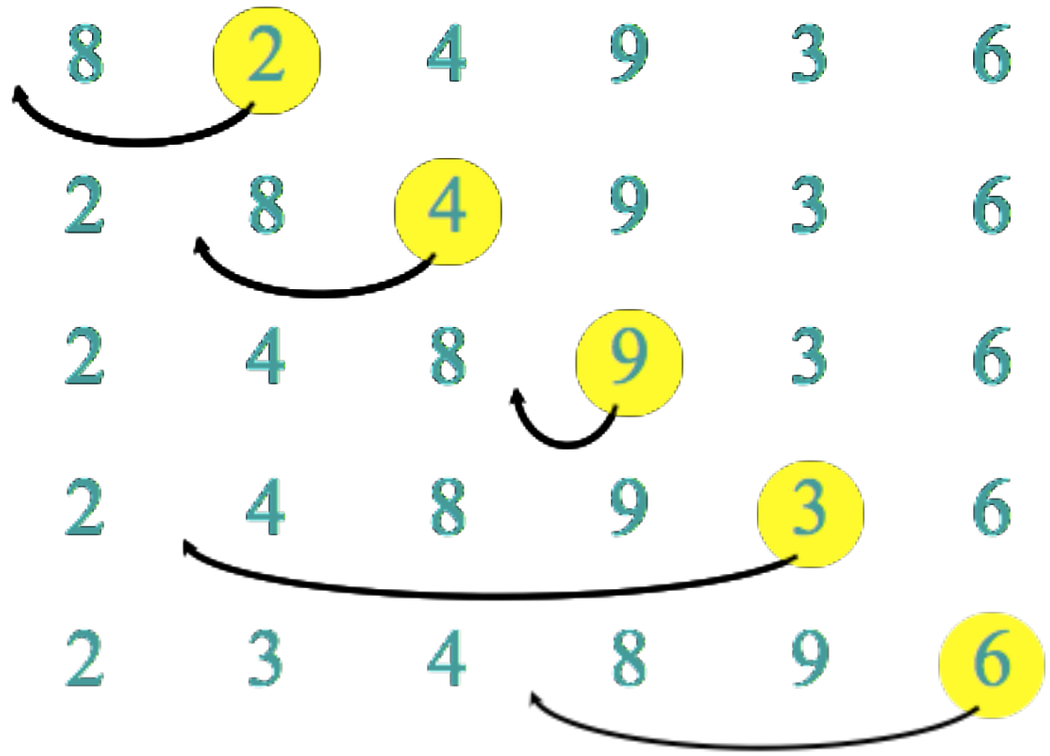
Insertion sort

- ***Idea:***
  - Tomar los elementos de la lista uno a uno e insertarlos el orden correspondiente.
  - ***Ejemplo:***



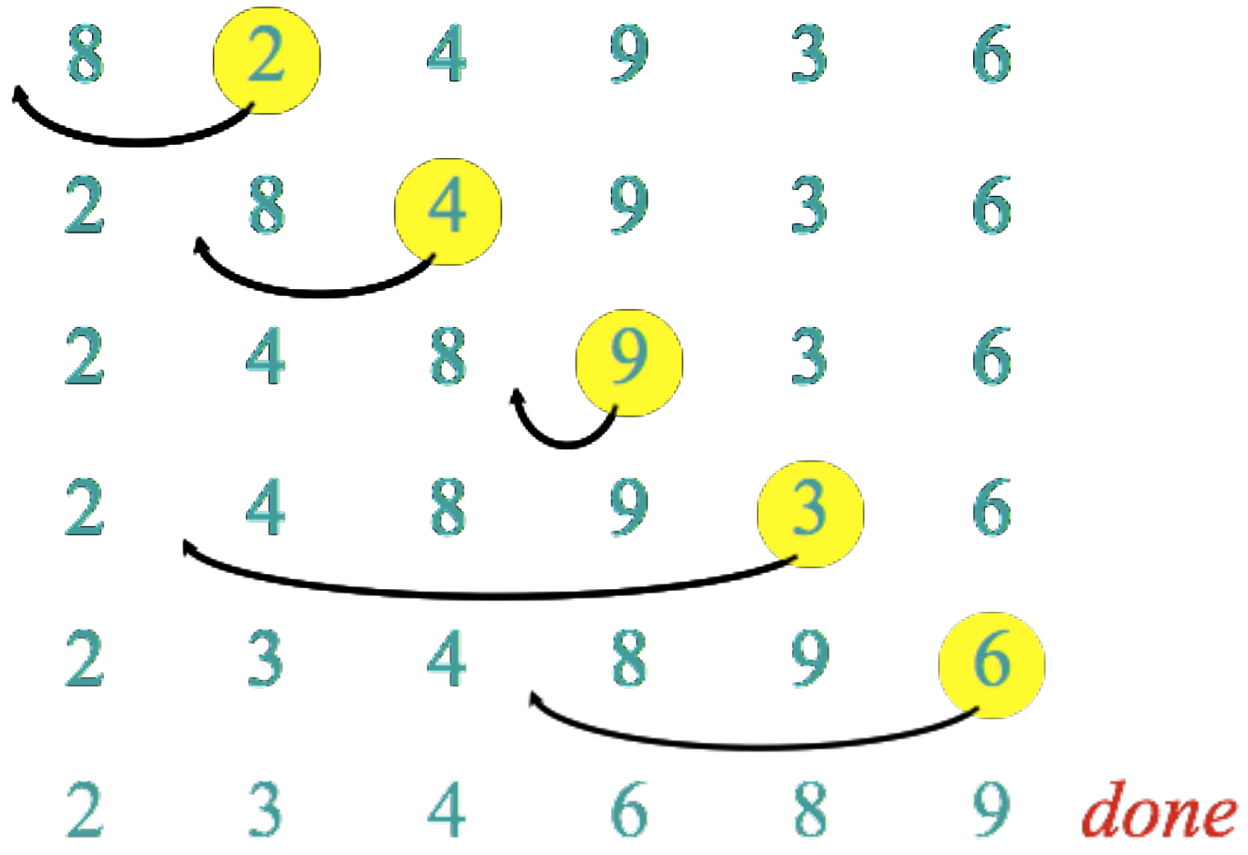
Insertion sort

- ***Idea:***
  - Tomar los elementos de la lista uno a uno e insertarlos el orden correspondiente.
  - ***Ejemplo:***



Insertion sort

- ***Idea:***
  - Tomar los elementos de la lista uno a uno e insertarlos el orden correspondiente.
  - ***Ejemplo:***



Insertion sort

- ¿ cuál es el mejor caso?
    - una lista ordenada: ***O(n)*** por iteración
- ¿ cuál es el peor caso?
    - una lista ordenada inversamente: ***O(n^2)***

Insertion sort

- ***Ventajas:***
    - Es muy **simple** y **fácil** de implementar.
    - Eficiente en conjuntos de datos pequeños
    - Eficiente en conjuntos de datos casi ordenados u ordenados.
    - Estable
    - El tiempo de ejecución depende de la entrada: una secuencia que ya está altamente ordenada tomará menos tiempo.
- ***Desventajas:***
    - Es ineficiente para conjuntos grandes de elementos.

Insertion sort

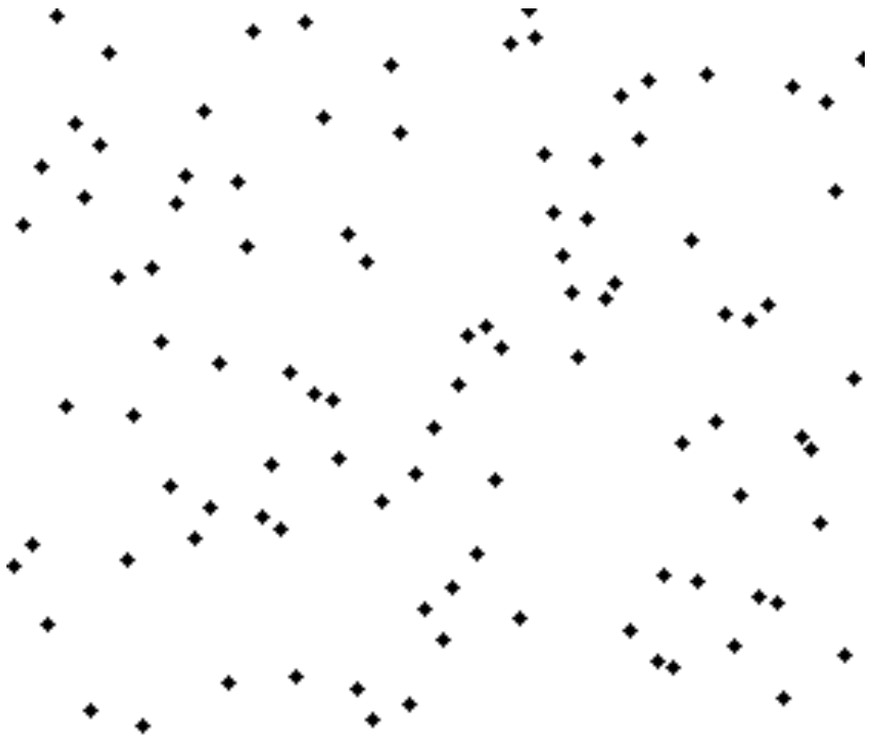
```
void insertionSort( Item a[], int l, int r )
{
    int temp, j;

    for ( int i=l+1; i<r; i++ ){
        j=i;
        while ( (j>l) && (a[j-1] > a[j])){
            temp = a[j-1];
            a[j-1] = a[j];
            a[j] = temp;
            j--;
        }
    }
}
```

Insertion sort

```
void insertionSort( Item a[], int l, int r )
{
    int temp, j;

    for ( int i=l+1; i<r; i++ ){
        j=i;
        while ( (j>l) && (a[j-1] > a[j])){
            temp = a[j-1];
            a[j-1] = a[j];
            a[j] = temp;
            j--;
        }
    }
}
```



Insertion sort

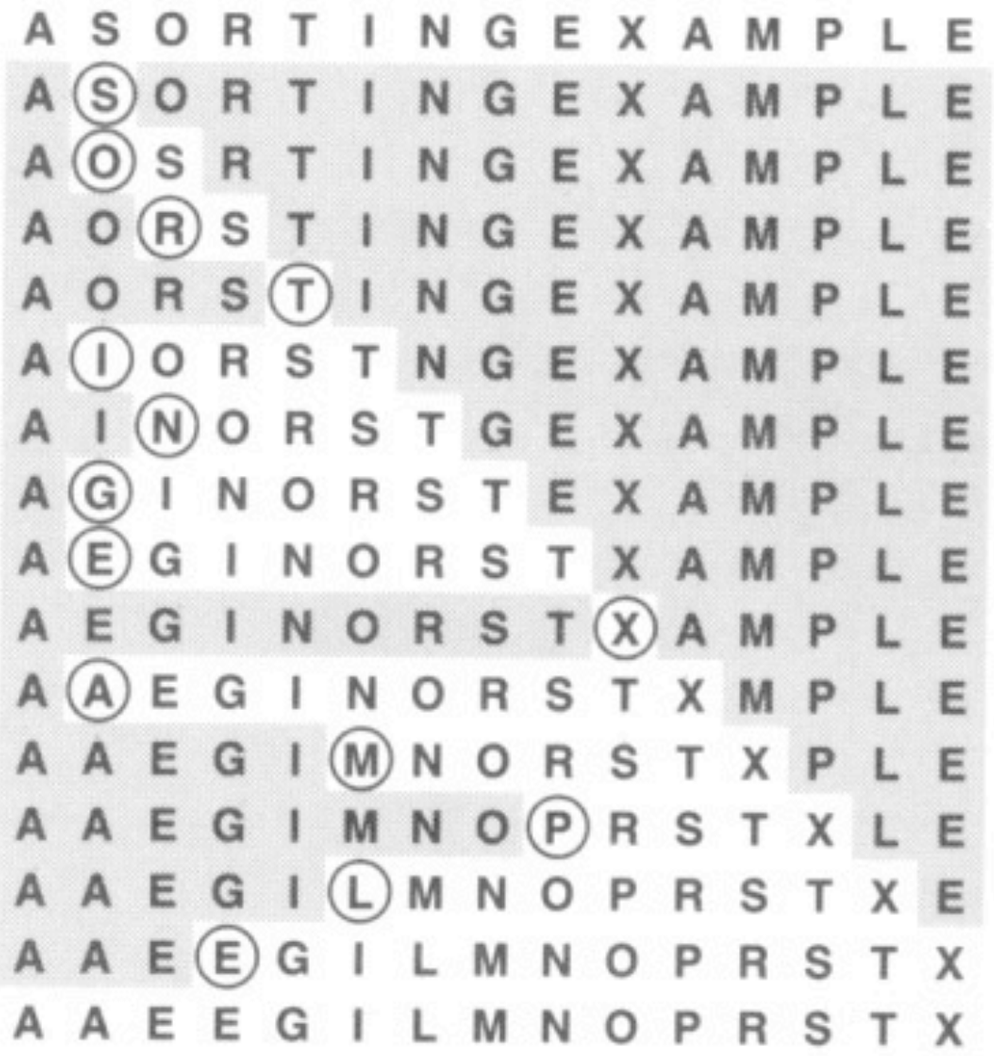
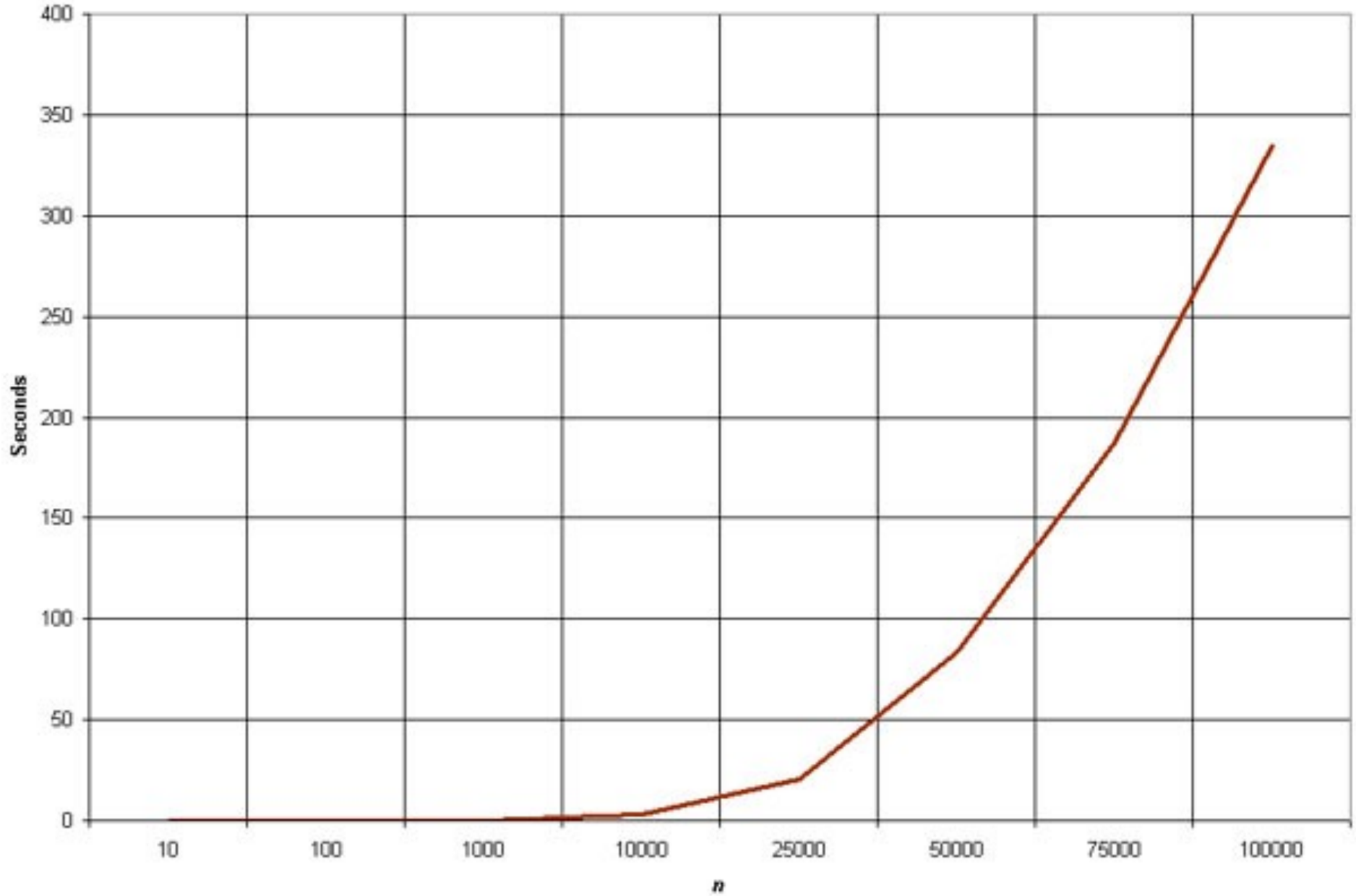


Figure 6.3
Insertion sort example

Insertion sort



Algoritmos de ordenamiento

- En general su tiempo de cálculo es proporcional a:
  - el número de operaciones de **comparación**
  - el número de operaciones de **intercambio**
  - ambos
- Con entradas aleatorias el tiempo de cálculo varía por una constante (siendo los tres de complejidad $O(n^2)$).

Algoritmos de ordenamiento

- **TAREA: REVISAR ESTAS PROPIEDADES**
- **Propiedad 1:** *Bubble sort toma cerca de* ***$N^2/2$*** *comparaciones y* ***$N^2/2$*** *intercambios en promedio y en el peor caso.*
- **Propiedad 2:** *Selection sort toma cerca de* ***$N^2/2$*** *comparaciones y* ***N*** *intercambios.*
- **Propiedad 3:** *Insertion sort toma cerca de* ***$N^2/4$*** *comparaciones y* ***$N^2/4$*** *movimientos en promedio, y el doble en el peor caso.*

Shell sort

Idea:

Mejorar el algoritmo de insertion sort moviendo más de un lugar a la vez.

•arreglar el conjunto de elementos en una tabla y arreglar las columnas.

•repetir el proceso cada vez con menos columnas pero más largas.

•al final quedará una sola columna

Ejemplo:

13 14 94 33 82 25 59 94 65 23 45 27 73 25 39 10

con un tamaño de paso de 5,3 y 1

Ejemplo shell sort

13 14 94 33 82 25 59 94 65 23 45 27 73 25 39 10

Si comenzamos con un tamaño de paso de 5, podríamos visualizar esto dividiendo la lista de números en una tabla con 5 columnas. Esto quedaría así:

13 14 94 33 82
25 59 94 65 23
45 27 73 25 39
10

Entonces ordenamos cada columna, lo que nos da

10 14 73 25 23
13 27 94 33 39
25 59 94 65 82
45

10 14 73 25 23 13 27 94 33 39 25 59 94 65 82 45

Shell sort

Original	32	95	16	82	24	66	35	19	75	54	40	43	93	68	
After 5-sort	32	35	16	68	24	40	43	19	75	54	66	95	93	82	6 swaps
After 3-sort	32	19	16	43	24	40	54	35	75	68	66	95	93	82	5 swaps
After 1-sort	16	19	24	32	35	40	43	54	66	68	75	82	93	95	15 swaps

Shell sort

- ¿qué secuencia de incrementos usar para optimalidad?
    - problema dificil y ampliamente estudiado
    - en práctica funcionan bien secuencias que decrecen geométricamente (cada incremento la mitad del anterior), entonces el número de incrementos es logarítmico en el tamaño de la entrada.

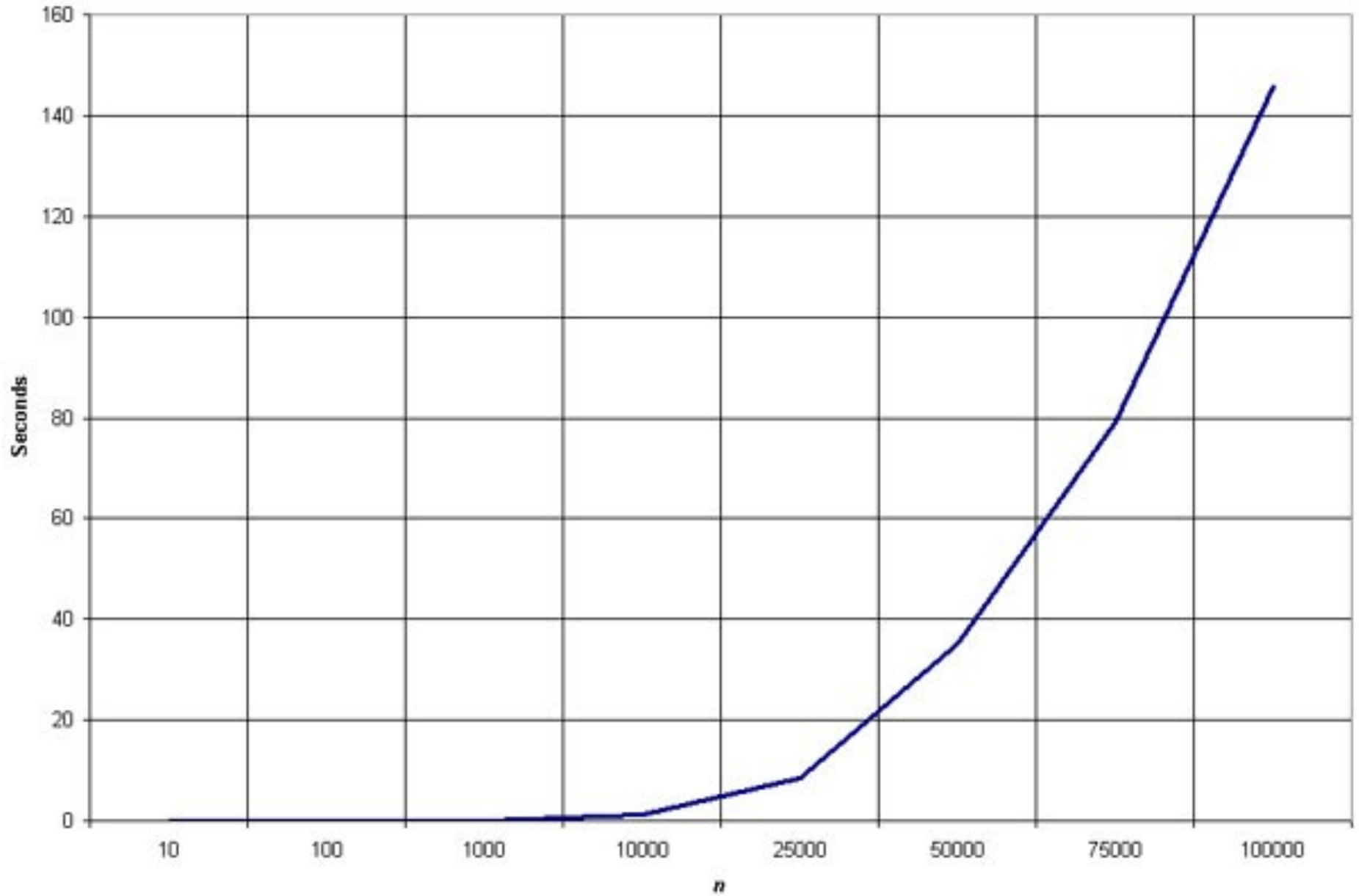
Shell sort

- Algoritmo difícil de analizar
- Más difícil de implementar que los vistos anteriormente.
- Mejora el desempeño de insertion sort
- Su desempeño depende de la secuencia de incrementos.

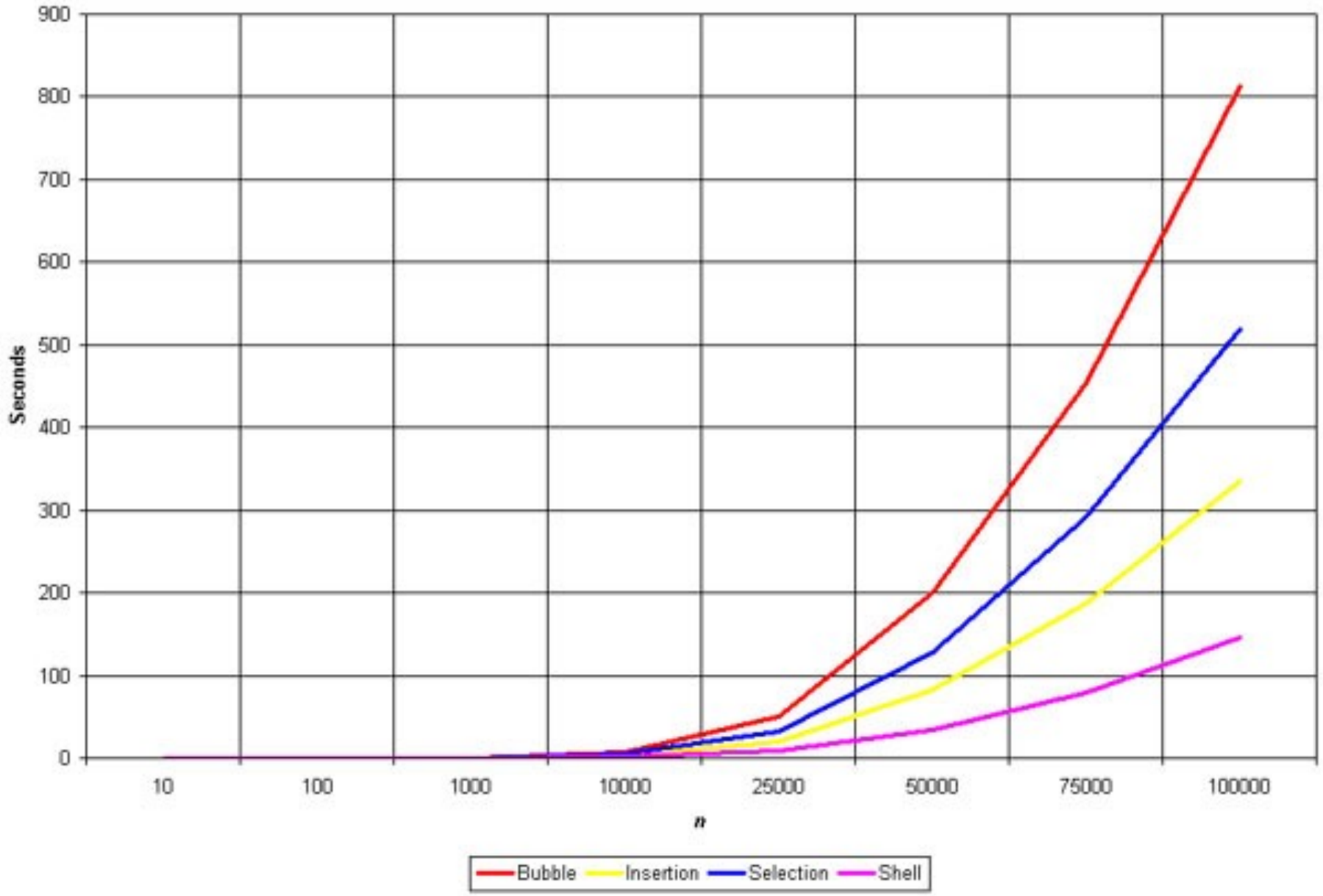
Shell sort

- ***Ventajas:***
    - Eficiente para conjuntos de elementos medianos (típicamente menores a 1000 elementos)
- ***Desventajas:***
    - Complejo para analizar y no se acerca a la eficiencia de merge, heap y quick sorts.

Shell sort


160
140
120
100
80
60
40
20
0
Seconds
10
100
1000
10000
25000
50000
75000
100000
n

Complejidad de diferentes algoritmos de ordenamiento

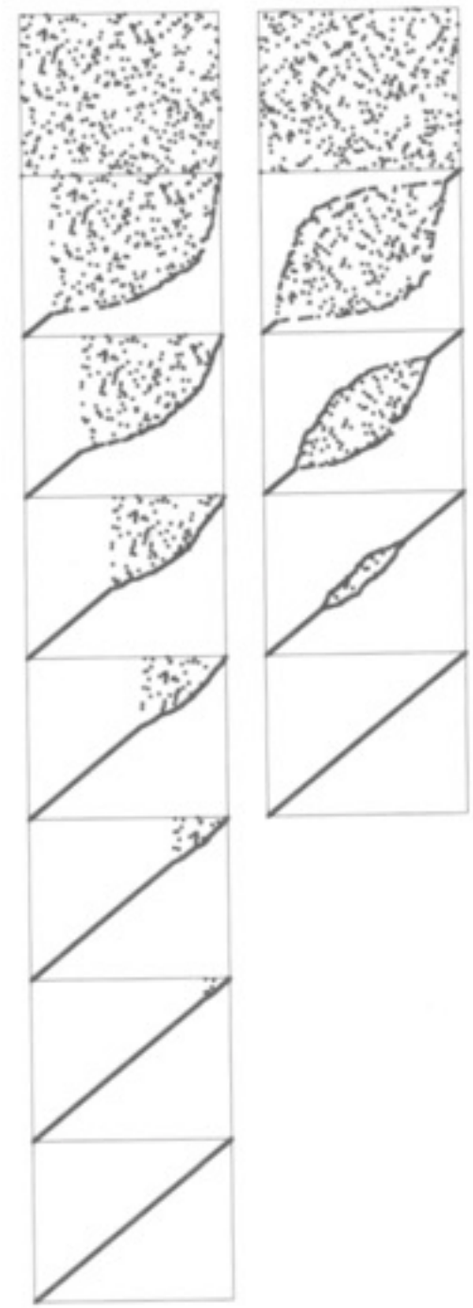


Algoritmos de ordenamiento de complejidad $O(n^2)$

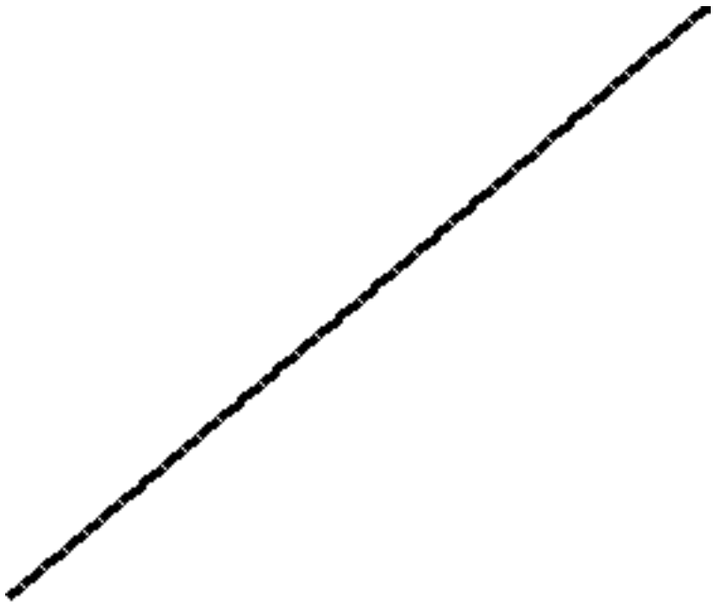
- fáciles de implementar
- útiles para prototipaje rápido
- cuando los elementos a ordenar son muy pocos
- cuando se requiere ordenar pocas veces.

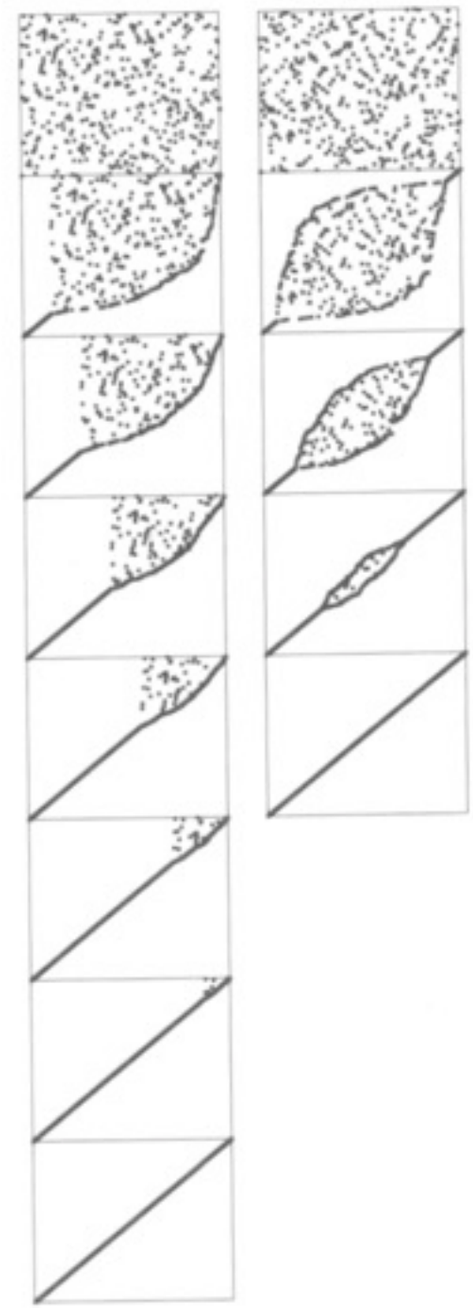
http://www.cs.ubc.ca/spider/harrison/Java/sorting-demo.html

Tarea, modificar bubble sort (izquierda) para que se comporte como el de la derecha.



Tarea, modificar bubble sort (izquierda) para que se comporte como el de la derecha.





Algoritmos de ordenamiento de complejidad $O(n^2)$

Nombre	Caso promedio	Peor caso	Memoria utilizada	Estabilidad	Método
Bubble sort	-	$O(n^2)$	O(1)	Si	Comparación e intercambio
Selection sort	$O(n^2)$	$O(n^2)$	O(1)	No	Selección
Insertion sort	O(n+d) donde d es el número de inversiones.	$O(n^2)$	O(1)	Si	Inserción
Shell sort	-	$O(n^2)$ depende de la secuencia de incremento	O(1)	No	Inserción

Información Extra de:
Algoritmos de ordenamiento

- métodos para ordenar ***archivos*** o ***elementos*** que contengan ***claves***.
- ordenar los elementos de tal modo que las claves estén ordenadas de acuerdo a una regla de orden dada.
- implementados con ***arreglos*** o con ***listas ligadas***.
  - ***sorting no-adaptativo:*** independiente a la estructura y orden de los datos de entrada.
  - ***sorting adaptativo:*** operaciones dependientes del orden de la entrada hasta el momento.

Información Extra de:
Algoritmos de ordenamiento

- un ordenamiento que se lleva a cabo ***enteramente*** en la ***memoria primaria*** se conoce como ***ordenamiento interno***.

- aquellos que involucran discos auxiliares para guardar resultados intermedios se conocen como ***ordenamientos externos***.