DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Algoritmos de ordenación
Análisis y Diseño de Algoritmos

Algoritmos de ordenación

- Algoritmos básicos: **$\Theta(n^2)$**
  - Ordenación por inserción
  - Ordenación por selección
  - Ordenación por intercambio directo (burbuja)
- Algoritmos más eficientes
  - Mergesort
  - Quicksort
  - Heapsort
  - Shellsort
- Algoritmos para casos especiales
  - Binsort (ordenación por urnas)
  - ...

Introducción

- Supongamos un subconjunto de n elementos $X = \{e_1,\ldots,e_n\}$ de un conjunto referencial Y, $X \subseteq Y$.
- Dentro de Y se define una relación de orden total $\leq$ (con las propiedades reflexiva, simétrica y transitiva).
- El objetivo de la ordenación es encontrar una secuencia de los elementos de X $\langle e_{\sigma(1)},\ldots,e_{\sigma(n)}\rangle$ tal que se verifique: $e_{\sigma(1)} \leq \ldots \leq e_{\sigma(n)}$

Introducción

Observaciones

- Los datos pueden ser simples o complejos.
- El orden se establece de acuerdo al campo **clave**.
- Los conjuntos de datos pueden tener duplicados.
- Si se mantiene el orden relativo de los datos con clave repetida en el conjunto de datos original, el algoritmo se dice que es **estable**.

Introducción

Tipos de algoritmos de ordenación

- Algoritmos de ordenación interna:
  En memoria principal (acceso aleatorio)

- Algoritmos de ordenación externa:
  En memoria secundaria (restricciones de acceso).

Aplicaciones

Aplicaciones evidentes:

- Mostrar los ficheros de un directorio.
- Mostrar un listado ordenado del contenido de una base de datos (p.ej. listado alfabético).
- Ordenar los resultados de una búsqueda en Internet (p.ej. Google PageRank).

Problemas que resultan más sencillos de resolver con los datos ordenados:

- Realizar una búsqueda (p.ej. búsqueda binaria).
- Encontrar la mediana.
- Encontrar el par más cercano.
- Identificar "outliers" / anomalías.
- Detectar duplicados.

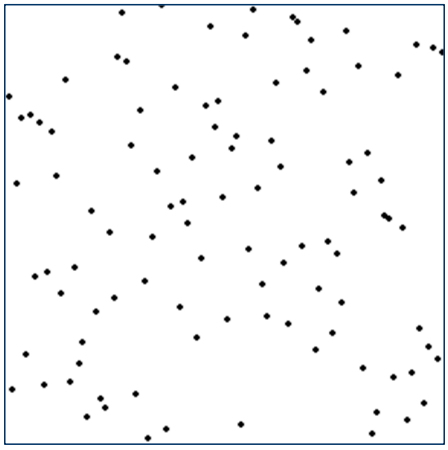
Aplicaciones

Otras aplicaciones (puede que no tan evidentes) :

- Compresión de datos.
- Informática gráfica.
- Planificación de tareas.
- Balanceado de carga en un ordenador paralelo.
- Biología computacional.
- Simulación (p.ej. sistemas de partículas).
- Árbol generador minimal [MST: Minimum Spanning Tree]
- Gestión de la cadena de suministro [SCM: Supply Chain Management]
- Recomendaciones de libros en Amazon.com
- …

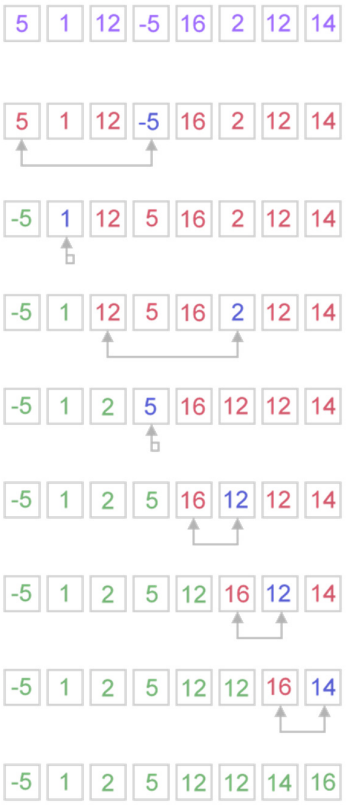
Ordenación por selección

En cada iteración, se selecciona el menor elemento del subvector no ordenado y se intercambia con el primer elemento de este subvector:



Ordenación por selección

```
void selectionSort (double v[])
{
  double tmp;
  int i, j, pos_min;
  int N = v.length;

  for (i=0; i<N-1; i++) {

      // Menor elemento del vector v[i..N-1]
      pos_min = i;

      for (j=i+1; j<N; j++)
          if (v[j]<v[pos_min])
             pos_min = j;

      // Coloca el mínimo en v[i]

      tmp = v[i];
      v[i] = v[pos_min];
      v[pos_min] = tmp;
  }
}
```



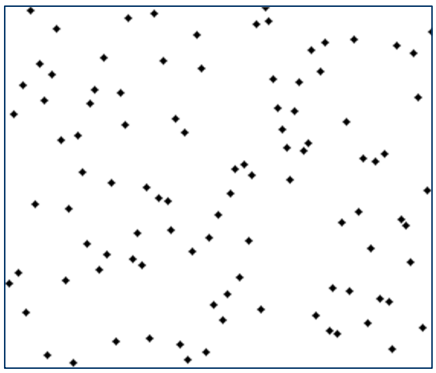
Ordenación por selección

```
void selectionSort (double v[])
{
  double tmp;
  int i, j, pos_min;
  int N = v.length;

  for (i=0; i<N-1; i++) {

      // Menor elemento del vector v[i..N-1]
      pos_min = i;

      for (j=i+1; j<N; j++)
          if (v[j]<v[pos_min])
             pos_min = j;

      // Coloca el mínimo en v[i]

      tmp = v[i];
      v[i] = v[pos_min];
      v[pos_min] = tmp;
  }
}
```

8
5
2
6
9
3
1
4
0
7

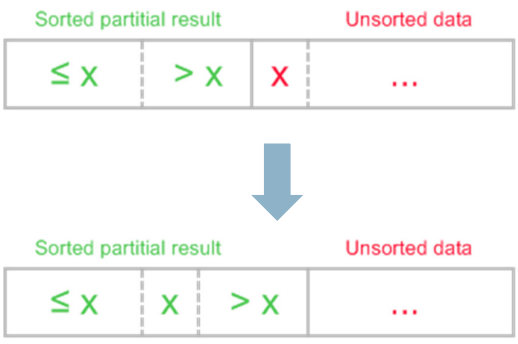
Ordenación por inserción

En cada iteración, se inserta un elemento del subvector no ordenado en la posición correcta dentro del subvector ordenado.



Ordenación por inserción

En cada iteración, se inserta un elemento del subvector no ordenado en la posición correcta dentro del subvector ordenado.



Ordenación por inserción

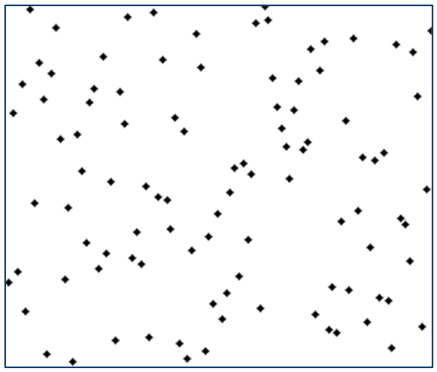
```
void insertionSort (double v[])
{
  double tmp;
  int i, j;
  int N = v.length;

  for (i=1; i<N; i++) {

      tmp = v[i];

      for (j=i; (j>0) && (tmp<v[j-1]); j--)
          v[j] = v[j-1];

      v[j] = tmp;
  }
}
```

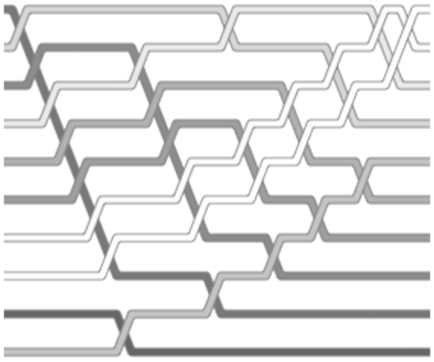
7	-5	2	16	4
7	-5	2	16	4
?	7	2	16	4
-5	7	2	16	4
-5	7	2	16	4
-5	?	7	16	4
-5	2	7	16	4
-5	2	7	16	4
-5	2	7	?	16
-5	2	?	7	16
-5	2	4	7	16
-5	2	4	7	16

Método de la burbuja

Ordenación por intercambio directo

IDEA: Los elementos más ligeros ascienden





Método de la burbuja

```
void bubbleSort (double v[])
{
  double tmp;
  int i,j;
  int N = v.length;

  for (i = 0; i < N - 1; i++)
      for (j = N - 1; j > i; j--)
          if (v[j] < v[j-1]) {
              tmp = v[j];
              v[j] = v[j-1];
              v[j-1] = tmp;
           }
}
```

5 1 12 -5 16	unsorted
5 1 12 -5 16	5 > 1, swap
1 5 12 -5 16	5 < 12, ok
1 5 12 -5 16	12 > -5, swap
1 5 -5 12 16	12 < 16, ok
1 5 -5 12 16	1 < 5, ok
1 5 -5 12 16	5 > -5, swap
1 -5 5 12 16	5 < 12, ok
1 -5 5 12 16	1 > -5, swap
-5 1 5 12 16	1 < 5, ok
-5 1 5 12 16	-5 < 1, ok
-5 1 5 12 16	sorted

Método de la burbuja

Mejora: Usar un centinela y terminar cuando en una iteración del bucle interno no se produzcan intercambios.

```
void bubbleSort(double[] v)
{
  boolean swapped = true;
  int i, j = 0;
  double tmp;
  while (swapped) {
        swapped = false;
        j++;
        for (i = 0; i < v.length - j; i++)
            if (v[i] > v[i + 1]) {
                tmp = v[i];
                v[i] = v[i + 1];
                v[i + 1] = tmp;
                swapped = true;
             }
  }
}
```

6 1 2 3 4 5	unsorted
6 1 2 3 4 5	6 > 1, swap
1 6 2 3 4 5	6 > 2, swap
1 2 6 3 4 5	6 > 3, swap
1 2 3 6 4 5	6 > 4, swap
1 2 3 4 6 5	6 > 5, swap
1 2 3 4 5 6	1 < 2, ok
1 2 3 4 5 6	2 < 3, ok
1 2 3 4 5 6	3 < 4, ok
1 2 3 4 5 6	4 < 5, ok
1 2 3 4 5 6	sorted

Análisis de eficiencia

Ordenación por selección

$N^2/2$ comparaciones y N intercambios.

Ordenación por inserción

$N^2/4$ comparaciones y $N^2/8$ intercambios en media

- El doble en el peor caso.
- Casi lineal para conjuntos casi ordenados.

Ordenación por burbuja

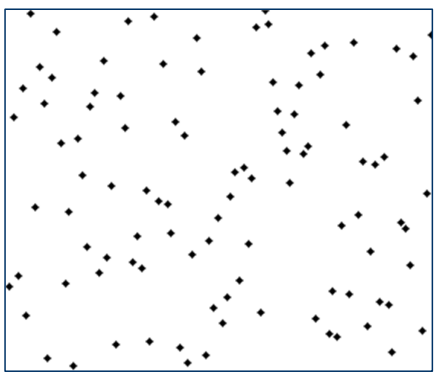
$N^2/2$ comparaciones y $N^2/2$ intercambios en media (y en el peor caso).

- Lineal en su versión mejorada si el vector está ordenado.

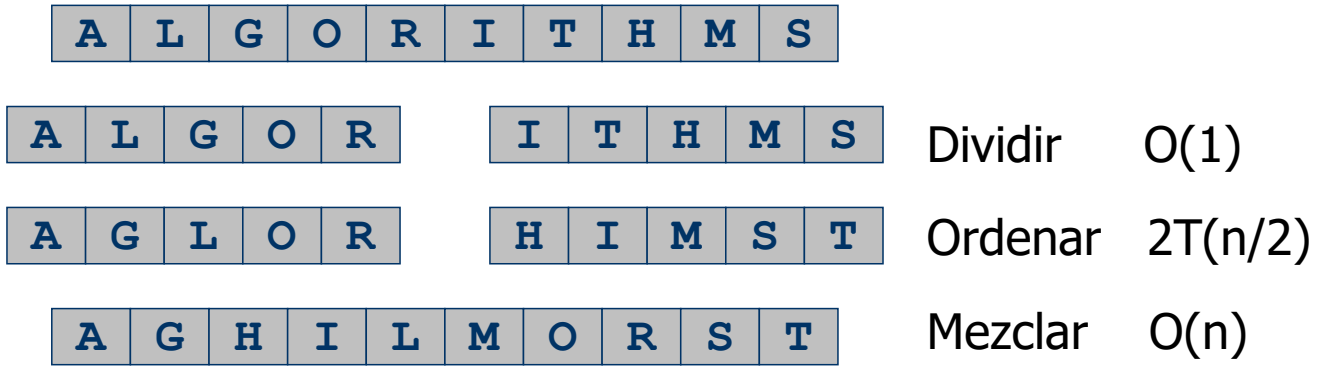
Mergesort

Algoritmo de ordenación "divide y vencerás":

1. Dividir nuestro conjunto en dos mitades.
2. Ordenar recursivamente cada mitad.
3. Combinar las dos mitades ordenadas: O(n).



Mergesort

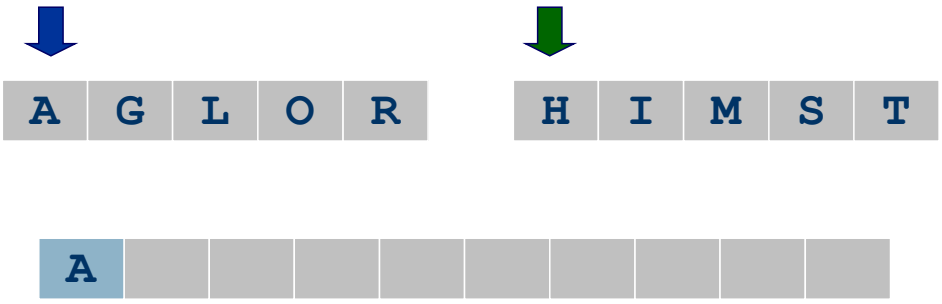


Mergesort

¿Cómo combinar eficientemente dos listas ordenadas?

Usando un array auxiliar y un número lineal de comparaciones:

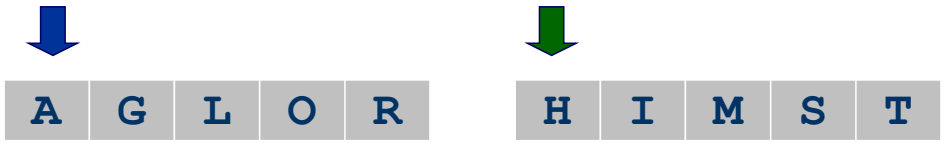
- Controlar la posición del elemento más pequeño en cada mitad.
- Añadir el más pequeño de los dos a un vector auxiliar.
- Repetir hasta que se hayan añadido todos los elementos.

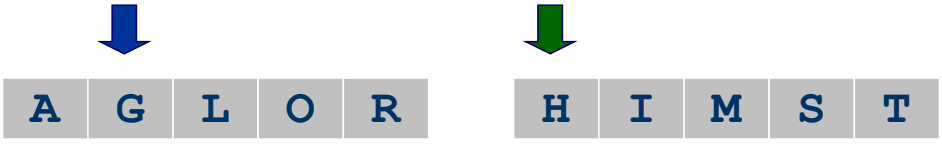


Optimización: "In-place merge" (Kronrud, 1969)

Cantidad constante de almacenamiento extra.

Mergesort



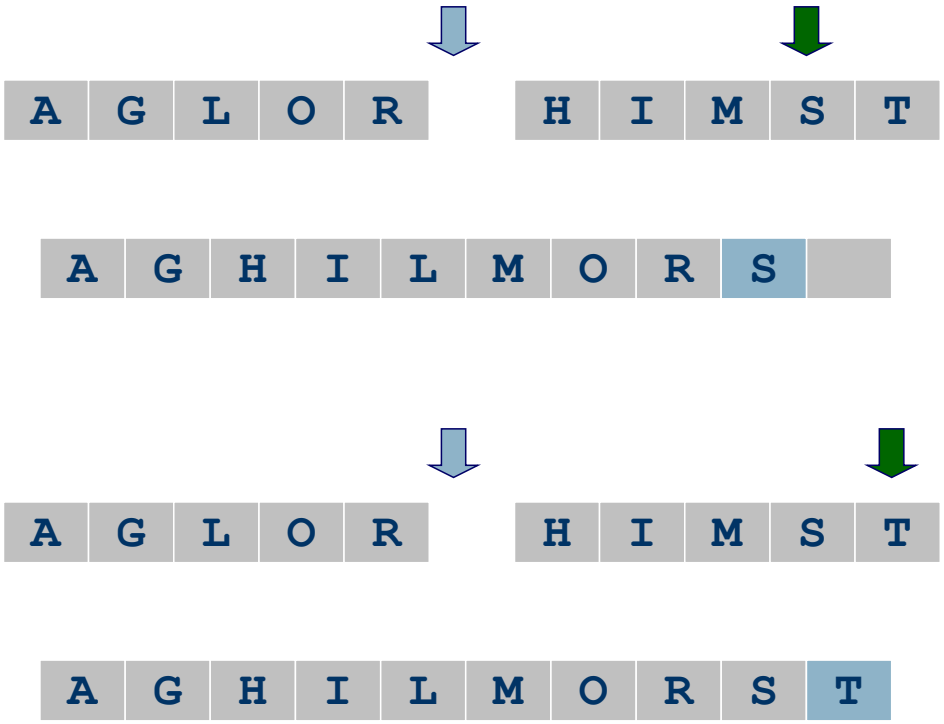


Mergesort

Mergesort

Mergesort

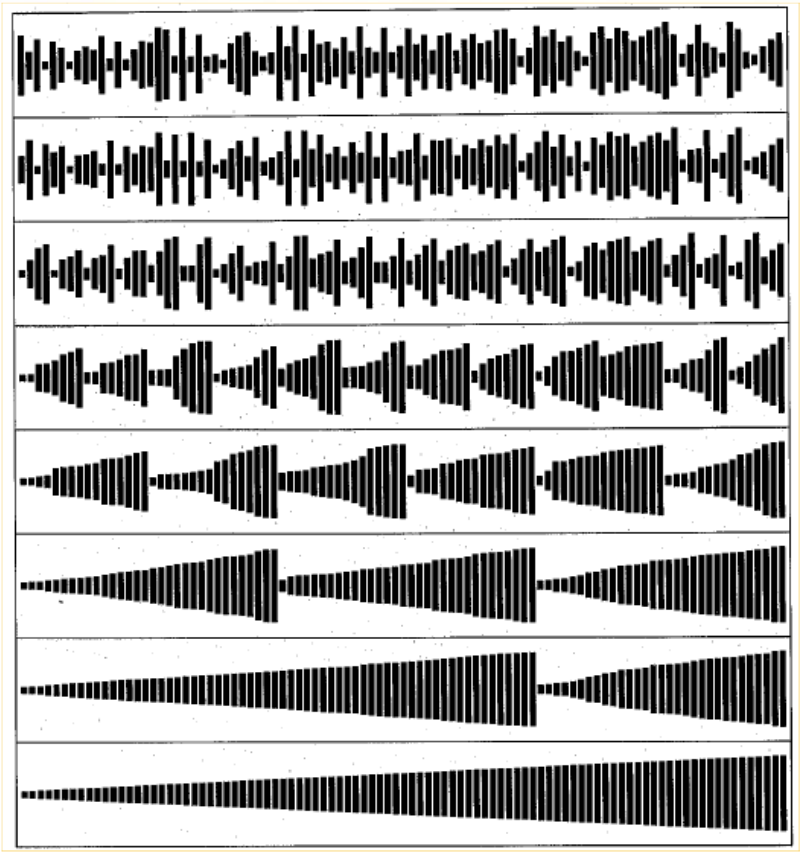
Mergesort



Mergesort

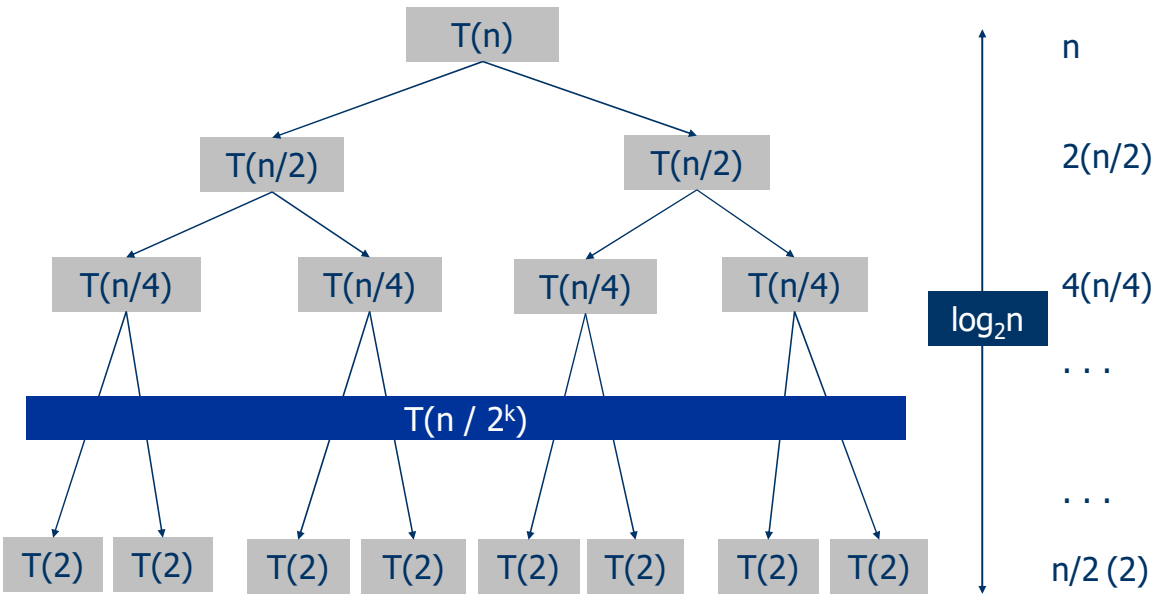
```
double[] merge (double L[],
                double R[])
{
  double data[] = new
         double[L.length+R.length]

  int i = 0;
  int j = 0;
  int k = 0;   // Mezcla...

  while ((i<L.length)
          && (j<R.length)) {
     if (L[i] <= R[j]) {
        data[k] = L[i];
        i++;
     } else {
        data[k] = R[j];
        j++;
     }
     k++;
  }

  // Elementos restantes de L[]
  while (i < L.length) {
     data[k] = L[i];
     i++;
     k++;
  }

  // Elementos restantes de R[]
  while (j < R.length) {
     data[k] = R[j];
     j++;
     k++;
  }

  return data;
}
```

Mergesort



Mergesort: Eficiencia

$$T(n) = \begin{cases} 1, & n = 1 \\ 2T(n/2) + n, & n > 1 \end{cases}$$



Mergesort: Eficiencia

$$T(n) = \begin{cases} 1, & n = 1 \\ 2T(n/2) + n, & n > 1 \end{cases}$$

Expandiendo la recurrencia:

$$\begin{aligned} \frac{T(n)}{n} &= \frac{2T(n/2)}{n} && +1 \\ &= \frac{T(n/2)}{n/2} && +1 \\ &= \frac{T(n/4)}{n/4} && +1+1 \\ &\cdots \\ &= \frac{T(n/n)}{n/n} && \underbrace{+1+\cdots+1}_{\log_2 n} \\ &= \log_2 n \end{aligned}$$

Mergesort: Eficiencia

$$T(n) = \begin{cases} 1, & n = 1 \\ 2T(n/2) + n, & n > 1 \end{cases}$$

Usando el método de la ecuación característica:

$$t_i = T(2^i) \qquad t_i = 2t_{i-1} + 2^i \qquad \Rightarrow \qquad t_i = c_1 2^i + c_2 i 2^i$$

$$T(n) = c_1 2^{\log_2(n)} + c_2 \log_2(n) 2^{\log_2(n)} = c_1 n + c_2 n \log_2(n)$$

$$T(n) \text{ es } O(n \log_2 n)$$

Quicksort

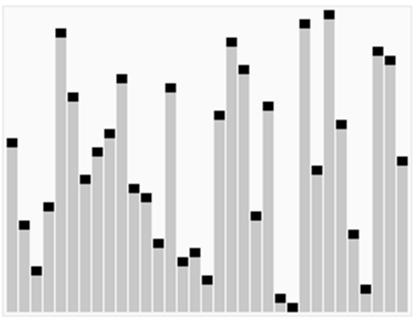
1. Se toma un elemento arbitrario del vector, al que denominaremos pivote (p).

2. Se divide el vector de tal forma que todos los elementos a la izquierda del pivote sean menores que él, mientras que los que quedan a la derecha son mayores que él.

3. Ordenamos, por separado, las dos zonas delimitadas por el pivote.

Quicksort

```
void quicksort (double v[], int izda, int dcha)
{
  int pivote;  // Posición del pivote

  if (izda<dcha) {

     pivote = partir (v, izda, dcha);

     quicksort (v, izda, pivote-1);

     quicksort (v, pivote+1, dcha);
  }
}
```



Uso:

```
quicksort (vector, 0, vector.length-1);
```

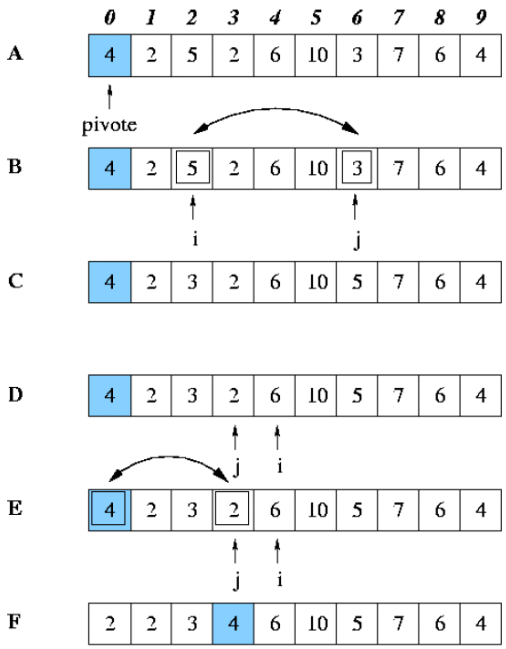
Quicksort

Obtención del pivote

Mientras queden elementos mal colocados con respecto al pivote:

- Se recorre el vector, de izquierda a derecha, hasta encontrar un elemento situado en una posición i tal que v[i] > p.
- Se recorre el vector, de derecha a izquierda, hasta encontrar otro elemento situado en una posición j tal que v[j] < p.
- Se intercambian los elementos situados en las casillas i y j (de modo que, ahora, v[i] < p < v[j]).

Quicksort



Quicksort

```
int partir (double v[],
      int primero, int ultimo)
{
  // Valor del pivote
  double pivote = v[primero];

  // Variable auxiliar
  double temporal;

  int izda = primero+1;
  int dcha = ultimo;

  do {  // Pivotear…

    while ((izda<=dcha)
          && (v[izda]<=pivote))
          izda++;
    while ((izda<=dcha)
          && (v[dcha]>pivote))
          dcha--;

    if (izda < dcha) {
       temporal = v[izda];
       v[izda] = v[dcha];
       v[dcha] = temporal;
       dcha--;
       izda++;
    }

  } while (izda <= dcha);

  // Colocar el pivote en su sitio
  temporal = v[primero];
  v[primero] = v[dcha];
  v[dcha] = temporal;

  // Posición del pivote
  return dcha;
}
```


Quicksort: Eficiencia

En el peor caso

$T(n) = T(1) + T(n-1) + c \cdot n \in O(n^2)$

Tamaño de los casos equilibrado

(idealmente, usando la mediana como pivote)

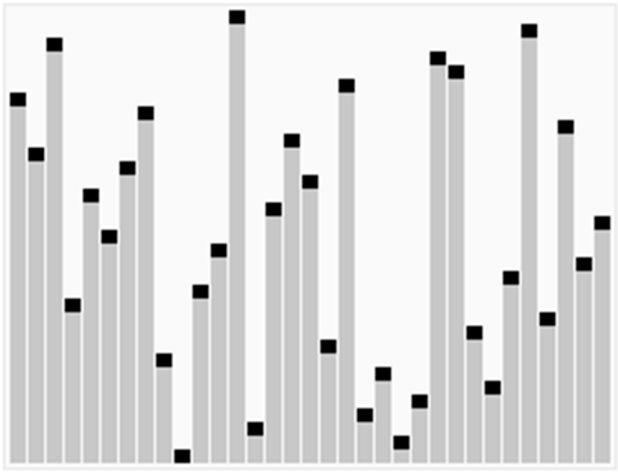
$T(n) = 2T(n/2) + c \cdot n \in O(n \log n)$

Promedio

$$T(n) = \frac{1}{n}\sum_{i=1}^{n-1}\left[T(n-i)+T(i)\right]+cn = \frac{2}{n}\sum_{i=1}^{n-1}T(i)+cn \in O(n\log n)$$

Heapsort

Algoritmo de ordenación de la familia del algoritmo de ordenación por selección basado en la construcción de un árbol parcialmente ordenado ("heap").



Heapsort O(n log n)

Heapsort: Eficiencia

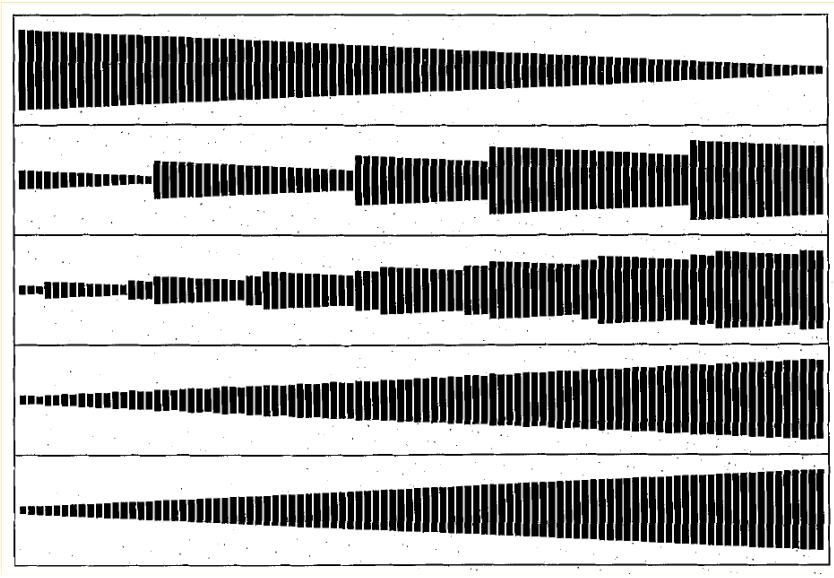
1. Construcción inicial del heap:
   n operaciones de inserción O(log n)
   sobre un árbol parcialmente ordenado $\Rightarrow$ O(n log n).

2. Extracción del menor/mayor elemento del heap:
   n operaciones de borrado O(log n)
   sobre un árbol parcialmente ordenado $\Rightarrow$ O(n log n).

Por tanto, el heapsort es O(n log n).

Shellsort

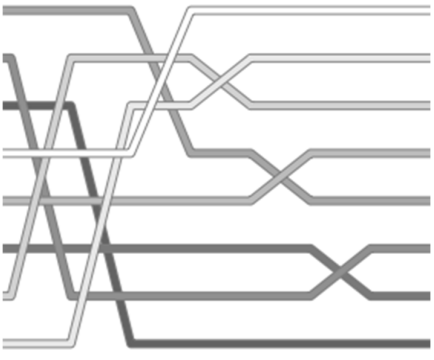
Mejora del algoritmo de ordenación por inserción: Compara elementos separados por varias posiciones y, en varias pasadas, de saltos cada vez menores, ordena el vector (Donald Shell, 1959).



Shellsort

Mejora del algoritmo de ordenación por inserción: Compara elementos separados por varias posiciones y, en varias pasadas de saltos cada vez menores, ordena el vector (Donald Shell, 1959).

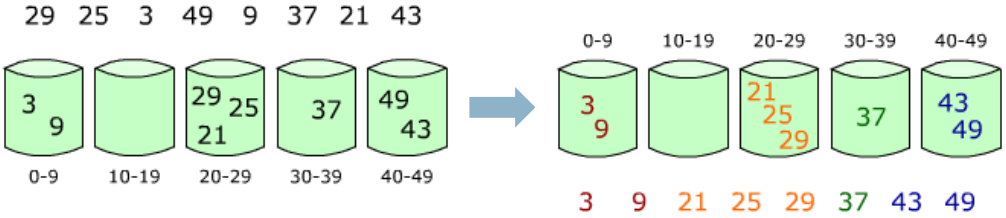
Inserción $O(n^2)$



Shellsort $O(n\log^2 n)$

Bucket sort & Binsort

Se divide el array en un conjunto de "urnas" y cada urna se ordena por separado: $O(n^2)$

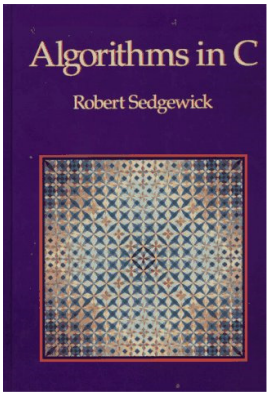


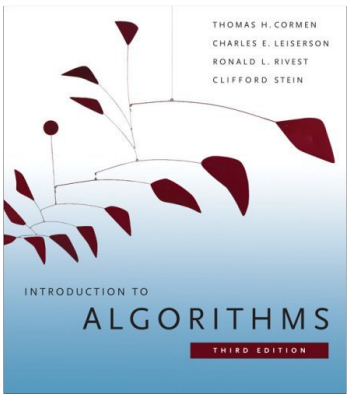
Bajo ciertas condiciones (claves entre 0 y N-1 sin duplicados), la ordenación se puede hacer en O(n):

```
void binSort (int[] v)
{
  for (int i = 0; i < v.length; i++)
       while (v[i] != i) {
              int tmp = v[v[i]];
              v[v[i]] = v[i];
              v[i] = tmp;
       }
}
```


Otros algoritmos

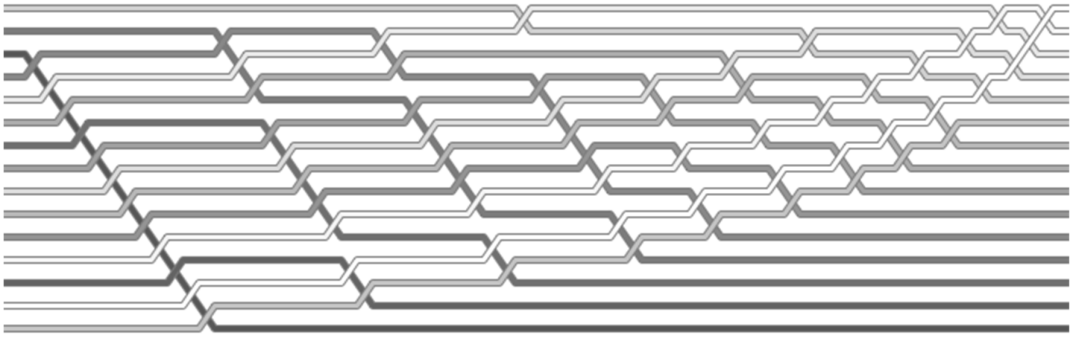
- Radix sort
- Histogram sort
- Counting sort = ultra sort = math sort
- Tally sort
- Pigeonhole sort
- Bead sort
- Timsort
- Smoothsort
- …



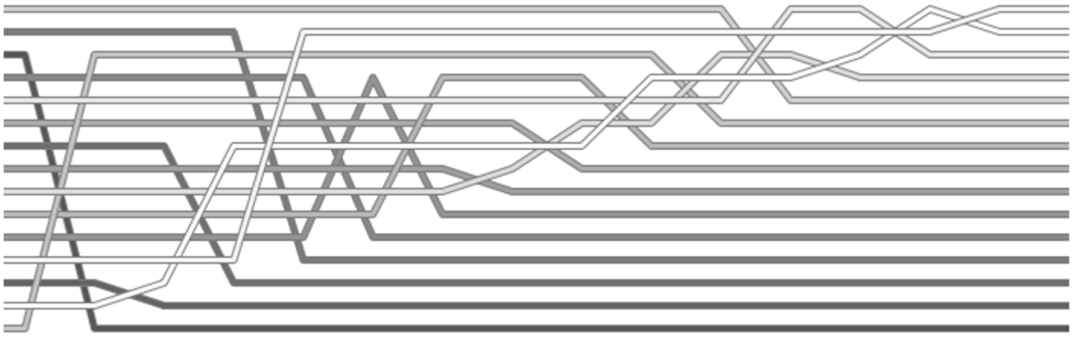


Resumen

Burbuja

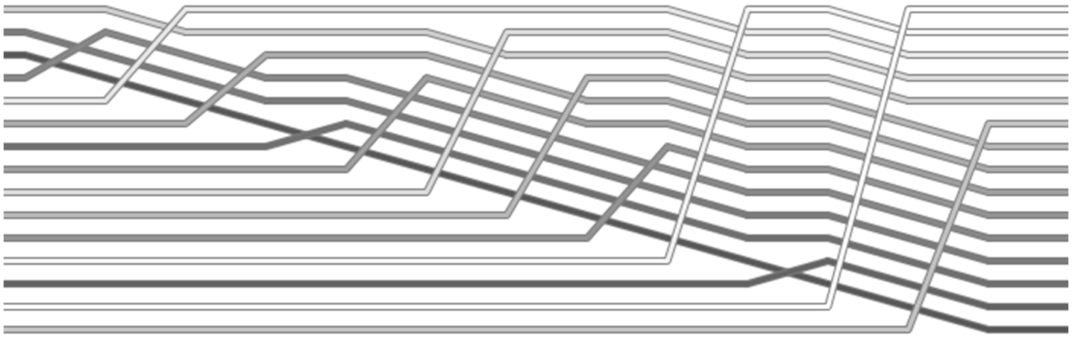


Selección

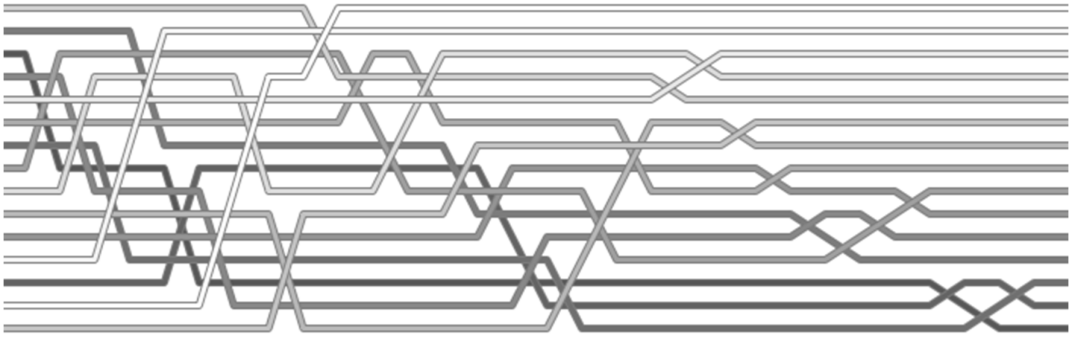


Resumen

Inserción

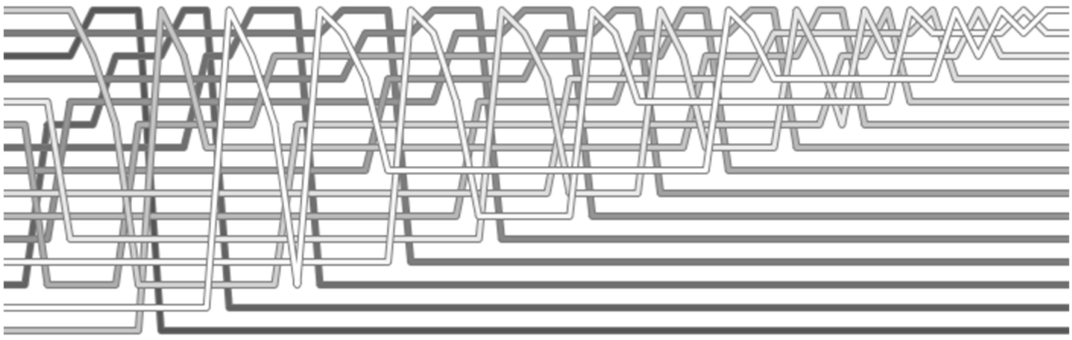


Shellsort

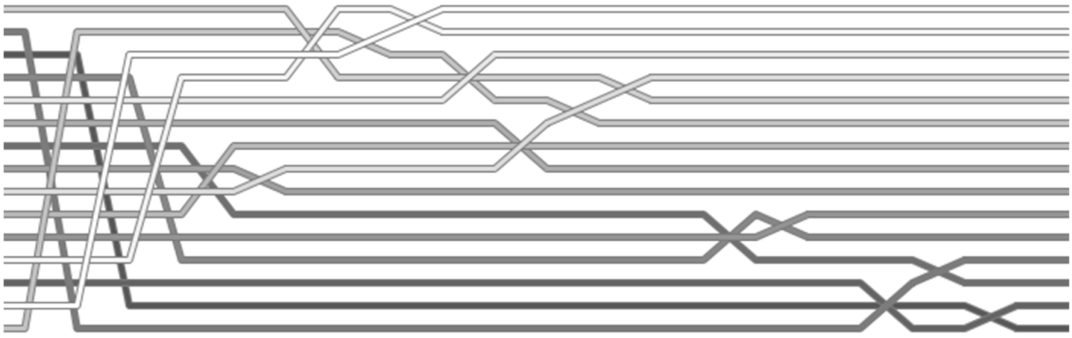


Resumen

Heapsort



Quicksort



Resumen

Timsort: Mergesort + Ordenación por inserción