Análisis de algoritmos de ordenación

- Basados en comparaciones
  - Algoritmos directos
    - Inserción Directa
    - Selección Directa
    - Intercambio/Burbuja
  - Algoritmos rápidos
    - Mezclas ("MergeSort")
    - Partición ("QuickSort")
    - Montículos ("HeapSort")
- No basados en comparaciones
  - Conteo o apartados ("CountingSort")
  - Residuos ("RadixSort")
  - Cubetas ("BucketSort")

http://www.sorting-algorithms.com/

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
etsinf Escuela Técnica Superior de Ingeniería Informática

Algoritmo de selección directa

- El algoritmo de ordenación por selección consiste en:
  - Seleccionar el mínimo elemento del array e intercambiarlo con el primero.
  - Seleccionar el mínimo en el resto del array e intercambiarlo con el segundo.
  - Y así sucesivamente...

- Algoritmo de selección directa (*Selection Sort*):
  http://www.youtube.com/watch?v=boOwArDShLU

Algoritmo de selección directa

- La estrategia de ordenación por selección directa se puede sintetizar en:
  - Para todo `i` desde `0` hasta `n-2` hacer:
  1. Encontrar la posición del mínimo en el subarray que va de `i` a `n-1`.
  2. Intercambiar el mínimo con `v[i]`.

  - Los elementos del subarray `v[0..i-1]` están ordenados entre sí y además tienen valores inferiores a los del subarray `v[i..n-1]`.

- La operación de encontrar el mínimo en el subarray es un recorrido.

Algoritmo de selección directa

- **Ejemplo:** ordenar el array {16, 54, 7, 98, 2, 66, 30, 14}

  {**2,** 54, 7, 98, 16, 66, 30, 14} ← selecciona 2 e intercambia con 16

  {**2, 7,** 54, 98, 16, 66, 30, 14} ← selecciona 7 e intercambia con 54

  {**2, 7, 14,** 98, 16, 66, 30, 54} ← selecciona 14 e intercambia con 54

  {**2, 7, 14, 16,** 98, 66, 30, 54} ← selecciona 16 e intercambia con 98

  {**2, 7, 14, 16, 30,** 66, 98, 54} ← selecciona 30 e intercambia con 98

  {**2, 7, 14, 16, 30, 54,** 98, 66} ← selecciona 54 e intercambia con 66

  {**2, 7, 14, 16, 30, 54, 66, 98**} ← selecciona 66 e intercambia con 98

Algoritmo de selección directa

- Implementación

```
static void selDirecta(int v[]){
    for (int i=0; i<v.length-1; i++) {
   // calcular la posición del mínimo de v[i:v.length-1]
        int pMin = i;
        for(int j=i+1; j<v.length; j++)
            if (v[j]<v[pMin]) pMin = j;
                // en pMin está la posición
                // del mínimo de v[i:v.length-1]
        // intercambiar v[i] con v[pMin]
        int aux = v[pMin];
        v[pMin] = v[i];
        v[i] = aux;          // desde 0 a i están ordenados
   }
            // array ordenado desde 0 hasta v.length-1
}
```

Algoritmo de selección directa

- Costes
- La talla del problema es el número de elementos a ordenar, n.
- La estructura se basa en dos recorridos anidados, y el coste no presenta variación frente a diferentes instancias del problema.

$$T(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} (n-i-1) = \frac{n(n-1)}{2} \in \Theta(n^2)$$

Algoritmo de inserción directa

- El algoritmo divide el array en una parte ordenada y otra no ordenada:
  - Inicialmente, la parte ordenada consta de un único elemento (el que ocupa la primera posición).
  - Los elementos son insertados uno a uno desde la parte no ordenada a la ordenada.
    - Seleccionar el elemento `v[1]` del array y situarlo de manera ordenada con el que ocupa la posición `0`.
    - Seleccionar el elemento `v[2]` del array y situarlo de manera ordenada entre los que ocupan las posiciones `0` y `1`.
    - Repetir situando el elemento `v[i]` de manera ordenada en el subarray comprendido entre las posiciones `0` e `i-1`.
  - Finalmente, la parte ordenada abarca todo el vector.

- Algoritmo de inserción directa (*Insertion Sort*)

http://www.youtube.com/watch?v=gTxFxgvZmQs&feature=related

Algoritmo de inserción directa

- La estrategia de ordenación por inserción directa se puede describir como sigue:
  – Para todo `i` desde `1` hasta `n-1` hacer:

1. Insertar el elemento `v[i]` de manera ordenada en el subarray `v[0..i-1]`:

    1. Se busca secuencialmente en `v[0..i-1]` el primer elemento menor o igual a `v[i]`. Sea `j` la posición de dicho elemento (o `-1` si no se encuentra).
    2. Se desplazan una posición hacia la derecha todos los elementos desde `j+1` hasta `i-1`.
    3. Se asigna el que había en `v[i]` a `v[j+1]`.

    - Los pasos 1 y 2 se pueden hacer de manera combinada.

Algoritmo de inserción directa

- **Ejemplo:** ordenar el array {16, 7, 54, 98, 2, 66, 30, 14}

{16, 7, 54, 98, 2, 66, 30, 14} ← inicialmente, [0..0] está ordenado

{7, 16, 54, 98, 2, 66, 30, 14} ← ordenados [0..1], inserta 7

{7, 16, 54, 98, 2, 66, 30, 14} ← ordenados [0..2], sin inserciones

{7, 16, 54, 98, 2, 66, 30, 14} ← ordenados [0..3], sin inserciones

{2, 7, 16, 54, 98, 66, 30, 14} ← ordenados [0..4], inserta 2

{2, 7, 16, 54, 66, 98, 30, 14} ← ordenados [0..5], inserta 66

{2, 7, 16, 30, 54, 66, 98, 14} ← ordenados [0..6], inserta 30

{2, 7, 14, 16, 30, 54, 66, 98} ← ordenados [0..7], inserta 14

Algoritmo de inserción directa

- Implementación

```
static void insDirecta(int v[]){
    for(int i=1; i<=v.length-1; i++) {
        int x = v[i]; // elemento a insertar
        int j = i-1;  // inicio de la parte ordenada
    // buscar en la parte ordenada,
    // desplazando a derecha los elementos mayores que x
        while( j>=0 && v[j]>x ) {
            v[j+1] = v[j];
            j--;
        }
        v[j+1] = x; // asignar x en la parte ordenada
    }
}
```

Algoritmo de inserción directa

- Costes
- La talla del problema es n, el número de elementos a ordenar.
- La estructura del algoritmo es un recorrido y una búsqueda combinadas.
- El bucle interno, el que hace la búsqueda, no siempre se repite completamente (cuando encuentra la posición correcta se detiene). Es decir, se ahorran algunos pasos.
- En este caso se puede escoger como instrucción crítica la comparación (`v[j]>x`).

Algoritmo de inserción directa

- Costes
- Caso mejor: Cuando el array está ordenado el bucle interno no itera ninguna vez, la comparación siempre se evalúa a falso. En este caso, el algoritmo ejecuta tantos pasos como el bucle externo.

$$T^m(n) = \sum_{i=1}^{n-1} 1 = n - 1 \in \Theta(n) \Rightarrow T(n) \in \Omega(n)$$

- Caso peor: El número de veces que se ejecuta el bucle interno es el máximo posible, es decir, `i` veces en cada iteración del bucle externo. Se trata del caso en el que el array está ordenado al revés de como se quiere ordenar.

$$T^p(n) = \sum_{i=1}^{n-1} i \in \Theta(n^2) \Rightarrow T(n) \in O(n^2)$$

Algoritmo de intercambio directo o burbuja

- Algoritmo de ordenación sencillo e ineficiente. Consistente en:
  - Recorrer el array comparando pares de elementos consecutivos.
  - Intercambiándolos si no están en el orden correcto.
- Algoritmo de intercambio directo (*Bubble Sort*)

  http://www.youtube.com/watch?v=1JvYAXT_064&feature=related

Algoritmo de intercambio directo o burbuja

- **Ejemplo**: ordenar el array {16, 54, 7, 98, 2, 66, 30, 14}
- **Primera iteración:**
  {**16,54,**7,98,2,66,30,14} → {**16,54,**7,98,2,66,30,14} (sin intercambio)
  {16,**54,7,**98,2,66,30,14} → {16,**7,54,**98,2,66,30,14} (intercambio 7 - 54)
  {16,7,**54,98,**2,66,30,14} → {16,7,**54,98,**2,66,30,14} (sin intercambio)
  {16,7,54,**98,2,**66,30,14} → {16,7,54,**2,98,**66,30,14} (intercambio 2 - 98)
  {16,7,54,2,**98,66,**30,14} → {16,7,54,2,**66,98,**30,14} (intercambio 66 - 98)
  {16,7,54,2,66,**98,30,**14} → {16,7,54,2,66,**30,98,**14} (intercambio 30 - 98)
  {16,7,54,2,66,30,**98,14**} → {16,7,54,2,66,30,**14,98**} (intercambio 14 - 98)

- **Segunda iteración:**
  {**16,7,**54,2,66,30,14,98} → {**7,16,**54,2,66,30,14,98} (intercambio 7 - 16)
  {7,**16,54,**2,66,30,14,98} → {7,**16,54,**2,66,30,14,98} (sin intercambio)
  {7,16,**54,2,**66,30,14,98} → {7,16,**2,54,**66,30,14,98} (intercambio 2 - 54)
  {7,16,2,**54,66,**30,14,98} → {7,16,2,**54,66,**30,14,98} (sin intercambio)
  {7,16,2,54,**66,30,**14,98} → {7,16,2,54,**30,66,**14,98} (intercambio 30 - 66)
  {7,16,2,54,30,**66,14,**98} → {7,16,2,54,30,**14,66,**98} (intercambio 14 – 66)

Algoritmo de intercambio directo o burbuja

- Implementación

```
static void burbuja (int[] v) {
    for (int i=1; i<=v.length-1; i++)
        for (int j=0; j<v.length-i; j++)
            // comparar pares de elementos consecutivos
            if ( v[j] > v[j+1] ) {
            // si par desordenado, entonces intercambio
                int x = v[j];
                v[j] = v[j+1];
                v[j+1] = x;
            }
}
```

Algoritmo de intercambio directo o burbuja

- Costes

- Los dos bucles se ejecutan `n-1` veces:

$$T(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{n-i-1} 1 = \sum_{i=1}^{n-1} (n-i) = n(n-1) - [n(\frac{n+1}{2}) - n)] \in \Theta(n^2)$$

- Una mejora consiste en añadir una bandera o flag que indique si se ha producido algún intercambio durante el recorrido:
  - Si no se ha producido ninguno, el array se encuentra ordenado y se puede acabar.
  - Con esta mejora su coste sigue siendo cuadrático.

Algoritmo de intercambio directo o burbuja

- Implementación (2)

```
static void burbuja(int[] v){
  int x;
  for(int i=0;i<v.length-1;i++)
   for(int j=v.length-1;j>i;j--)
      if ( v[j-1] > v[j] ) {
        x = v[j-1];
        v[j-1] = v[j];
        v[j] = x;
    }
}
```

16	54	7	98	2	66	30	14
16	54	7	98	2	66	14	30
16	54	7	98	2	14	66	30
16	54	7	98	2	14	66	30
16	54	7	2	98	14	66	30
16	54	2	7	98	14	66	30
16	2	54	7	98	14	66	30
2	16	54	7	98	14	66	30

Algoritmo de mezcla natural

- Problema: dados 2 arrays a y b con número diferente de elementos pero ordenados, fusionarlos en un array nuevo c que quede ordenado.

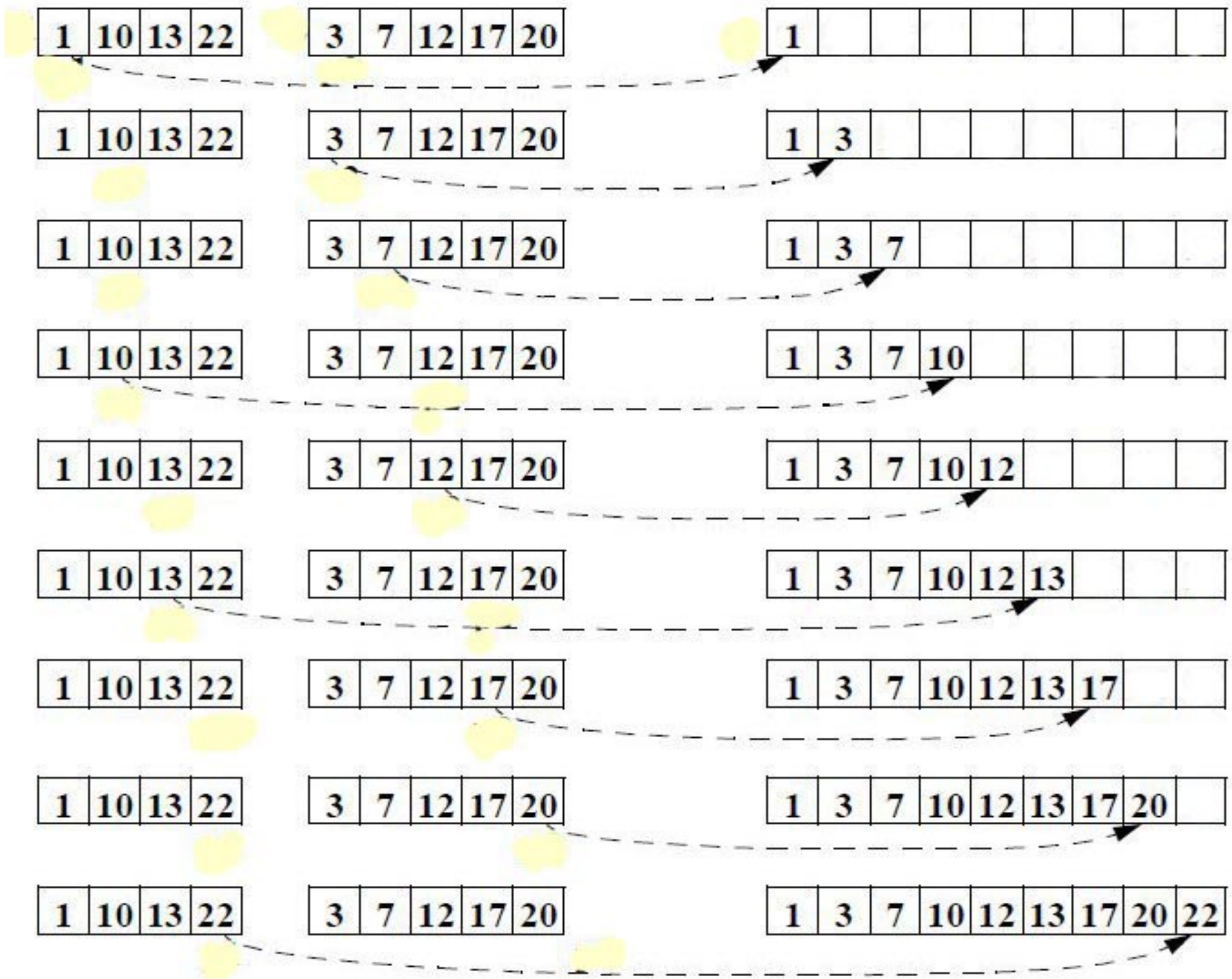
- El algoritmo que resuelve este problema en dos fases es el siguiente:

  1. Un bucle que compara los elementos de los 2 arrays y los copia de manera ordenada en el array destino. Este bucle acaba cuando se alcanza el final de uno de los arrays.

  2. Un bucle que copia, sin comparar nada, los restantes elementos de uno de los arrays al final del array destino.

Algoritmo de mezcla natural

- Ejemplo de ejecución.



Algoritmo de mezcla natural

- Implementación

```
/** a y b están ordenados
  * c.length es a.length + b.length
  */
static void mezclaNatural(int a[], int b[], int[] c){
   int i=0, l=a.length, j=0, m=b.length, k=0;

   while ( i < l && j < m ) {
        if (a[i] < b[j]) { c[k] = a[i]; i++; }
        else { c[k] = b[j]; j++; }
        k++;
   }

   for (int r=i; r < l; r++)  { c[k] = a[r]; k++; }
   for (int r=j; r < m; r++)  { c[k] = b[r]; k++; }
}
```

Algoritmo de mezcla natural

- El tamaño **`n`** del problema viene determinado por la suma de los tamaños de los arrays a mezclar.

- Para contar los pasos del algoritmo, se pueden contar las veces que se ejecuta el incremento de la variable índice de acceso al array destino.

- Entonces, se ejecuta **`l+m`** veces, donde **`l`** es **`a.length`** y **`m`** es **`b.length`**.

$$T(\mathbf{l},\mathbf{m}) = \mathbf{l+m} \in \Theta(\mathbf{l+m})$$

Es decir, $T(\mathbf{n}) \in \Theta(\mathbf{n})$

Algoritmo MergeSort

- Algoritmo de ordenación que consiste en:
  - Dividir el array en dos partes iguales.
  - Ordenar por separado cada una de las partes (mediante llamadas recursivas).
  - Mezclar ambas partes, manteniendo la ordenación.

- Algoritmo de mezcla directa (MergeSort)

  http://www.youtube.com/watch?v=HA6ghMlYuO4

Algoritmo MergeSort

- Implementación

```
static void mergesort (int[] v, int ini, int fin) {
    if ( ini<fin ) {
        int mitad = (fin+ini)/2;
        mergesort(v, ini, mitad);
        mergesort(v, mitad+1, fin);
    // versión de mezcla natural especializada:
    // mezcla en v[ini..fin] de
    // v[ini..mitad] y v[mitad+1..fin]
        mezclaNatural2(v, ini, mitad, fin);
    }
}
```

Algoritmo MergeSort

- Costes
- Coste de `mezclaNatural2`: $\Theta(n)$

- Coste de `mergesort`:

$$T(n) = \begin{cases} c_1 & 0 \le n \le 1 \\ 2T(n/2) + c_2 n & n > 1 \end{cases}$$

Algoritmo MergeSort

- Despliegue de recurrencias

$$T(n) = \begin{cases} c_1 & 0 \le n \le 1 \\ 2T(n/2) + c_2 n & n > 1 \end{cases}$$

$$T(n) \le 2T\left(\frac{n}{2}\right) + c_2 n \qquad\qquad T\left(\frac{n}{2}\right) \le 2T\left(\frac{n}{4}\right) + c_2 \frac{n}{2}$$

$$T(n) \le 2\left[2T\left(\frac{n}{4}\right) + c_2 \frac{n}{2}\right] + c_2 n = 4T\left(\frac{n}{4}\right) + 2c_2 n$$

$$T(n) \le 4\left[2T\left(\frac{n}{8}\right) + c_2 \frac{n}{4}\right] + 2c_2 n = 8T\left(\frac{n}{8}\right) + 3c_2 n$$

Algoritmo MergeSort

- Despliegue de recurrencias
- Si n >= 2^i, se tiene que:

$$T(n) \le 2^i T\left(\frac{n}{2^i}\right) + ic_2 n$$

- Si n = 2^k, se obtiene T(1) en la parte derecha:

$$T(n) \le 2^k T(1) + kc_2 n$$

- Si n = $2^k \Leftrightarrow k = \log_2 n$, y como $T(1) \le c_1$:

$$T(n) \le c_1 n + c_2 n \log n \Rightarrow T(n) \in \Theta(n \log n)$$