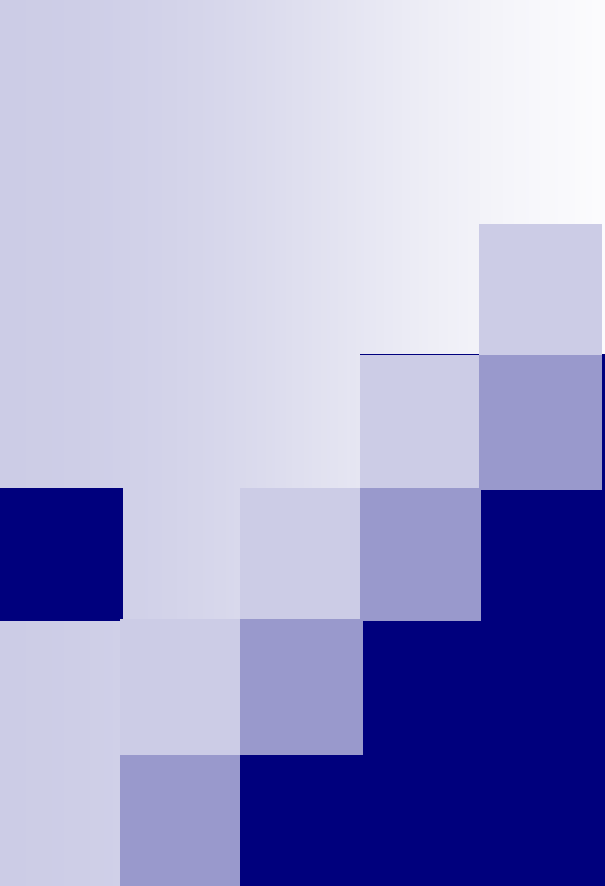
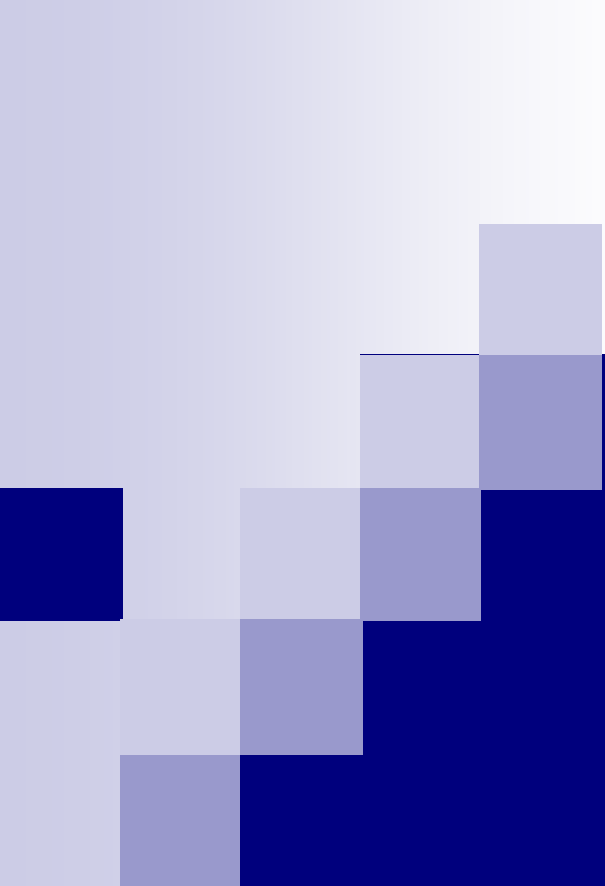




Tema 2

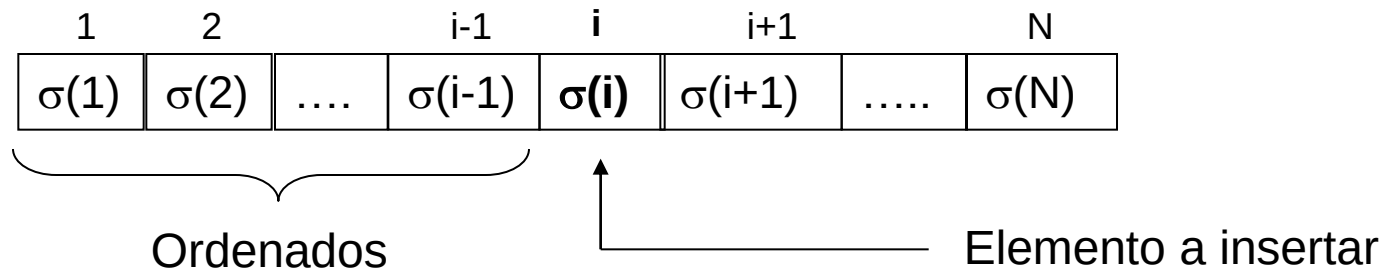
Algoritmos de ordenación



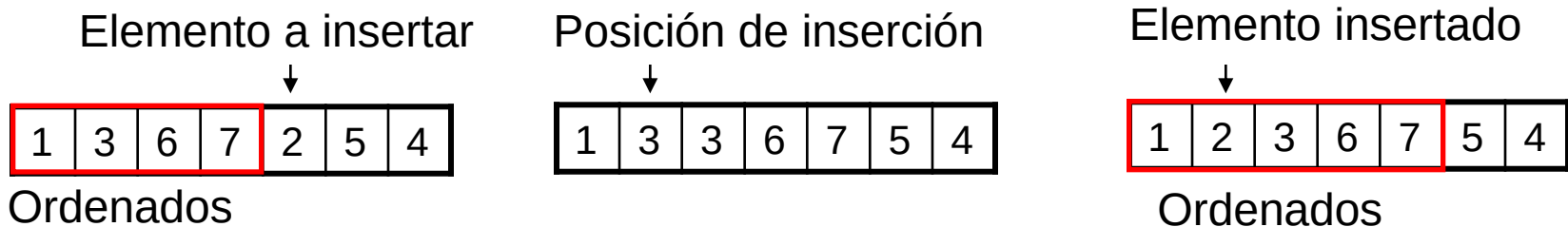
2.1 Algoritmos locales de ordenación

InsertSort

- La idea de InsertSort consiste en tener los $i-1$ primeros elementos de la tabla ordenados entre sí al inicio de la iteración i .



- En la iteración i se coloca el elemento $\sigma(i)$ en la posición correspondiente entre 1 e i de tal forma que pasan a estar ordenados entre sí los i primeros elementos de la tabla.



InsertSort

```
InsertSort(Tabla T, ind P, ind U)
```

```
  para i de P+1 a U;
```

```
    A=T[i];
```

```
    j=i-1;
```

```
    mientras (j ≥ P && T[j]>A);
```

```
        T[j+1]=T[j];
```

```
        j--;
```

```
    T[j+1]=A;
```

Operación Básica (CDC)

■ Observaciones:

- El trabajo de bucle interno depende de la entrada
- El trabajo sobre una entrada σ será:

$$n_{IS}(\sigma) = \sum_{i=2}^N n_{IS}(\sigma, i)$$

- $1 \leq n_{IS}(\sigma, i) \leq i-1$

InsertSort: casos mejor y peor

- Tenemos que $1 \leq n_{IS}(\sigma, i) \leq i-1$; por tanto se tiene que $\forall \sigma \in \Sigma_N$:

$$\sum_{i=2}^N 1 \leq \sum_{i=2}^N n_{IS}(\sigma, i) \leq \sum_{i=2}^N i-1 \Rightarrow N-1 \leq n_{IS}(\sigma) \leq \frac{N(N-1)}{2}$$

■ Caso peor

- Paso 1: Por lo anterior $\forall \sigma \in \Sigma_N, n_{IS}(\sigma) \leq N(N-1)/2$
 - Paso 2: $n_{IS}([N, N-1, N-2, \dots, 1]) = N(N-1)/2$
- $\left. \vphantom{\begin{matrix} \square \\ \square \end{matrix}} \right\} \Rightarrow W_{IS}(N) = N(N-1)/2$

■ Caso mejor

- Paso 1: Por lo anterior $\forall \sigma \in \Sigma_N, n_{IS}(\sigma) \geq N-1$
 - Paso 2: $n_{IS}([1, 2, 3, \dots, N]) = N-1$
- $\left. \vphantom{\begin{matrix} \square \\ \square \end{matrix}} \right\} \Rightarrow B_{IS}(N) = N-1$

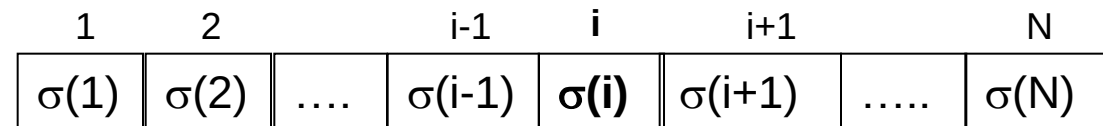
InsertSort: caso medio

- El caso medio para IS es

$$\begin{aligned} A_{IS}(N) &= \sum_{\sigma \in \Sigma_N} p(\sigma) n_{IS}(\sigma) = \frac{1}{N!} \sum_{\sigma \in \Sigma_N} n_{IS}(\sigma) = \frac{1}{N!} \sum_{\sigma \in \Sigma_N} \sum_{i=2}^N n_{IS}(\sigma, i) = \\ &= \sum_{i=2}^N \frac{1}{N!} \sum_{\sigma \in \Sigma_N} n_{IS}(\sigma, i) = \sum_{i=2}^N \boxed{A_{IS}(N, i)} \end{aligned}$$

Nº medio de operaciones que realiza
IS en la iteración i

- Estado de la tabla en la iteración i



Ordenados

Elemento a insertar

InsertSort: caso medio

- Observación: al abordar IS la entrada $\sigma(i)$, ésta puede acabar en las posiciones

$i, i-1, i-2, \dots, j, \dots, 2, 1$

haciendo respectivamente

$1, 2, 3, \dots, i-j+1, \dots, i-1$ e $i-1$

cdcs respectivamente

Posición Final	CDC perdidas ($\sigma(i) < \sigma(j)$)	CDC ganadas ($\sigma(i) > \sigma(j)$)	Total CDC
i	0	1 ($\sigma(i) > \sigma(i-1)$)	$1 = i - i + 1$
$i-1$	1 ($\sigma(i) < \sigma(i-1)$)	1 ($\sigma(i) > \sigma(i-2)$)	$2 = i - (i-1) + 1$
$i-2$	2 ($\sigma(i) < \sigma(i-1), \sigma(i) < \sigma(i-2)$)	1 ($\sigma(i) > \sigma(i-3)$)	$3 = i - (i-2) + 1$
$\dots j \dots$	$i-j$	1 ($\sigma(i) > \sigma(j-1)$)	$i-j+1$
3	$i-3$ ($\sigma(i) < \sigma(i-1), \dots, \sigma(i) < \sigma(3)$)	1 ($\sigma(i) > \sigma(2)$)	$i-2 = i-3+1$
2	$i-2$ ($\sigma(i) < \sigma(i-1), \dots, \sigma(i) < \sigma(2)$)	1 ($\sigma(i) > \sigma(1)$)	$i-1 = i-2+1$
1	$i-1$ ($\sigma(i) < \sigma(i-1), \dots, \sigma(i) < \sigma(1)$)	0	$i-1$

InsertSort: caso medio

- Esto es,

$$n_{IS}(i \rightarrow j) = i - j + 1 \text{ si } 1 < j \leq i \text{ y } n_{IS}(i \rightarrow 1) = i - 1,$$

donde $n_{IS}(i \rightarrow j)$ es el número de CDC necesarias para insertar el elemento $\sigma(i)$ en la posición j .

- Q: con qué probabilidad acabará $\sigma(i)$ en la posición j ?
- Si las σ son equiprobables, es razonable pensar que $P(\sigma(i) \text{ acabe en } j) = 1/i$ para todo j entre 1 e i
- De aquí se deduce que ***el trabajo medio $A_{IS}(N, i)$ de IS sobre la entrada i -ésima de una tabla de N elementos será $i/2 + O(1)$***
- Y por tanto ***$A_{IS}(N) = N^2/4 + O(N)$***
- En más detalle ...

InsertSort: caso medio

- Recordamos que $A_{IS}(N, i) = \sum_{j=1}^i p(j) n_{IS}(i \rightarrow j)$,
donde $p(j)$ es la probabilidad de que el elemento $\sigma(i)$ termine en la posición j .
- Asumimos por **equiprobabilidad** que $p(j) = 1/i$ ($j=1, 2, \dots, i$) , con lo que se tiene:

$$A_{IS}(N, i) = \frac{1}{i} \sum_{j=1}^i n_{IS}(i \rightarrow j) = \frac{1}{i} \left[\left(\sum_{j=1}^i (i-j) \right) + (i-1) \right] = \frac{i-1}{2} + \frac{i-1}{i}$$

- Dado que $A_{IS}(N) = \sum_{i=2}^N A_{IS}(N, i)$, sustituyendo lo anterior resulta:

$$A_{IS}(N) = \sum_{i=2}^N \left(\frac{i-1}{2} + \frac{i-1}{i} \right) = \frac{1}{2} \sum_{i=1}^{N-1} i + \sum_{i=1}^{N-1} \frac{i-1}{i} = \frac{N^2}{4} + O(N)$$

InsertSort: caso medio

- Sabemos que

$$W_{IS}(N) = N^2/2 + O(N)$$

$$A_{IS}(N) = N^2/4 + O(N)$$

- Conclusión: IS es algo mejor que SS y que BS, pero no mucho más en el caso medio e igual en el caso peor
- Q: hemos llegado al límite de eficacia en ordenación?

Cotas inferiores

- ¿Cuánto podemos mejorar un algoritmo de ordenación por cdcs?
- Es decir ¿existe una $f(N)$ universal tal que $n_A(\sigma) \geq f(N)$ para cualquier A ?
- Obviamente se tiene que $n_A(\sigma) \geq N$, por tanto $n_A(\sigma) = \Omega(N)$.
- ¿Existe algún algoritmo que alcance esa cota mínima?
- Si existe ¿cómo es ese algoritmo y en que condiciones la alcanza ?
- Herramienta: medida del desorden de una tabla

¿Cómo medir el desorden de una tabla?

- Las operaciones que realiza un algoritmo dependen del orden de la tabla.
- ¿Como medir el orden o desorden que hay en una tabla?.
- **Definición:**
 - Decimos que dos índices $i < j$ están en inversión si $\sigma(i) > \sigma(j)$
- Ejemplo:
 - $\sigma = (3 \ 2 \ 1 \ 5 \ 4)$
 1 2 3 4 5
 - Inversiones de $\sigma = ((1,2), (1,3), (2,3), (4,5)) \Rightarrow 4$ inversiones.
 - $\text{Inv}(\sigma) = 4$
 - En la práctica decimos que “3 está en inversión con 2” en vez de que “los índices 1 y 2 están en inversión”

¿Cómo medir el desorden de una tabla?

■ Observaciones

1. $\text{inv}([1,2,3,\dots,N-1,N])=0$
2. $\text{inv}([5,4,3,2,1])=10$
3. $\text{inv}([N,N-1,N-2,\dots,2,1])=N+(N-1)+\dots+2+1=N^2/2-N/2$

Obs: No puede haber ninguna permutación **con más inversiones** que $\sigma = [N,N-1,N-2,\dots,2,1]$ ya que

$$\begin{aligned}\text{inv}([\sigma(1), \sigma(2), \dots, \sigma(N)]) &= \text{inv}(\sigma(1)) + \text{inv}(\sigma(2)) + \dots + \text{inv}(\sigma(N-1)) + \text{inv}(\sigma(N)) \leq \\ &\quad \underbrace{\quad}_{\wedge I} \quad \underbrace{\quad}_{\wedge I} \quad \underbrace{\quad}_{\wedge I} \\ &\quad N-1 \quad \quad N-2 \quad \quad 1 \\ &\leq (N-1) + (N-2) + \dots + 1 = N^2/2 - N/2\end{aligned}$$

Cotas inferiores para algoritmos locales

- **Definición:** Un algoritmo de ordenación por comparación de clave (CDC) es **local** si **por cada CDC** que realiza el algoritmo **se deshace a lo sumo una inversión**.
- InsertSort, BurbujaSort y (moralmente) SelectSort son locales
- **Obs:** Si A es un algoritmo local, el número mínimo de CDC que realizará A será el número de inversiones que tenga la tabla a ordenar σ , es decir $n_A(\sigma) \geq \text{inv}(\sigma)$.
- **Caso peor: Si A es local, $W_A(N) \geq N^2/2 - N/2$**

$$W_A(N) \geq n_A([N, N-1, N-2, \dots, 2, 1]) \geq \text{inv}([N, N-1, N-2, \dots, 2, 1]) = N^2/2 - N/2$$

- **Consecuencia:** IS, BS y SS son **óptimos** en el caso **peor** entre los algoritmos locales

Cotas inferiores en el caso medio

- **Definición:** Si $\sigma \in \Sigma_N$ definimos su traspuesta, σ^t como $\sigma^t(i) = \sigma(N-i+1)$.
- Ejemplo $\sigma = [3, 2, 1, 5, 4]$ entonces $\sigma^t = [4, 5, 1, 2, 3]$
- Observaciones:
 - $(\sigma^t)^t = \sigma$
 - $\text{inv}([3, 2, 1, 5, 4]) + \text{inv}([4, 5, 1, 2, 3]) = 3 + 7 = 10 = (5 \cdot 4) / 2$
 - $\text{inv}([5, 4, 3, 2, 1]) + \text{inv}([1, 2, 3, 4, 5]) = 10 + 0 = 10 = (5 \cdot 4) / 2$
- **Proposición:** Si $\sigma \in \Sigma_N$
$$\text{inv}(\sigma) + \text{inv}(\sigma^t) = N(N-1)/2$$

Demo: Si $1 \leq i < j \leq N$, o bien (i, j) es inversión de σ o $(N-j+1, N-i+1)$ lo es de σ^t

Cotas inferiores en el caso medio

- ***Si A es local $A_A(N) \geq N^2/4 + O(N)$***

$$\begin{aligned} A_A(N) &= \frac{1}{N!} \sum_{\sigma \in \Sigma_N} n_A(\sigma) \overset{\text{A local}}{\geq} \frac{1}{N!} \sum_{\sigma \in \Sigma_N} \text{inv}(\sigma) = \frac{1}{N!} \sum_{\sigma, \sigma^t} (\text{inv}(\sigma) + \text{inv}(\sigma^t)) = \\ &= \frac{1}{N!} \frac{N(N-1)}{2} \sum_{\sigma, \sigma^t} 1 = \frac{1}{N!} \frac{N(N-1)}{2} \frac{N!}{2} = \frac{N^2}{4} + O(N) \end{aligned}$$

- InsertSort es **óptimo** para el caso medio entre los algoritmos locales.

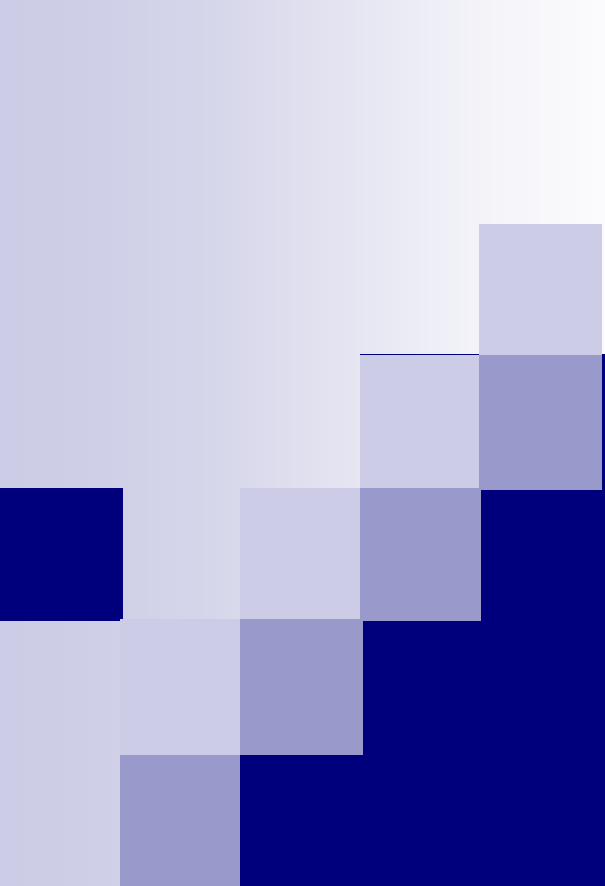
Los algoritmos locales de ordenación son poco eficaces.

En esta sección hemos aprendido...

- El algoritmo de ordenación InsertSort, así como el cálculo de sus casos mejor, peor y medio.
- El concepto de algoritmo de ordenación local, así como sus cotas inferiores para los casos peor y medio.

Herramientas y técnicas a trabajar ...

- Funcionamiento de InsertSort
- Evolución de algoritmos locales
- Casos peor, medio y mejor de InsertSort y algoritmos similares y variantes
- Detección y cuenta de inversiones en permutaciones
- Rendimiento de algoritmos locales en permutaciones concretas
- Problemas a resolver (al menos!!) :
32, 33, 34, 35, 37, 39, 40, 44, 45, 46



2.2 Algoritmos recursivos de ordenación

Métodos divide y vencerás (DyV)

- La idea de los algoritmos divide y vencerás es la siguiente:
 - Partir la tabla T en dos subtablas T_1 y T_2
 - Ordenar T_1 y T_2 recursivamente.
 - Combinar T_1 y T_2 ya ordenados en T también ordenada.
- Pseudocódigo general de algoritmos DyV

```
DyVSort(tabla T)
  si  $\text{dim}(T) \leq \text{dimMin}$  :
    directSort(T);
  else :
    Partir( $T, T_1, T_2$ );
    DyVSort( $T_1$ );
    DyVSort( $T_2$ );
    Combinar( $T, T_1, T_2$ );
```

- Una primera opción es implementar una función **Partir** sencilla y una función **Combinar** complicada.
- Resultado: **MergeSort**.

MergeSort

```
status MergeSort(tabla T, ind P, ind U)
```

```
  si P>U:
```

```
    devolver ERROR;
```

```
  si P==U:  //tabla con un elemento
```

```
    devolver OK;
```

```
  else:
```

```
    M=⌊(P+U)/2⌋;      // "partir"
```

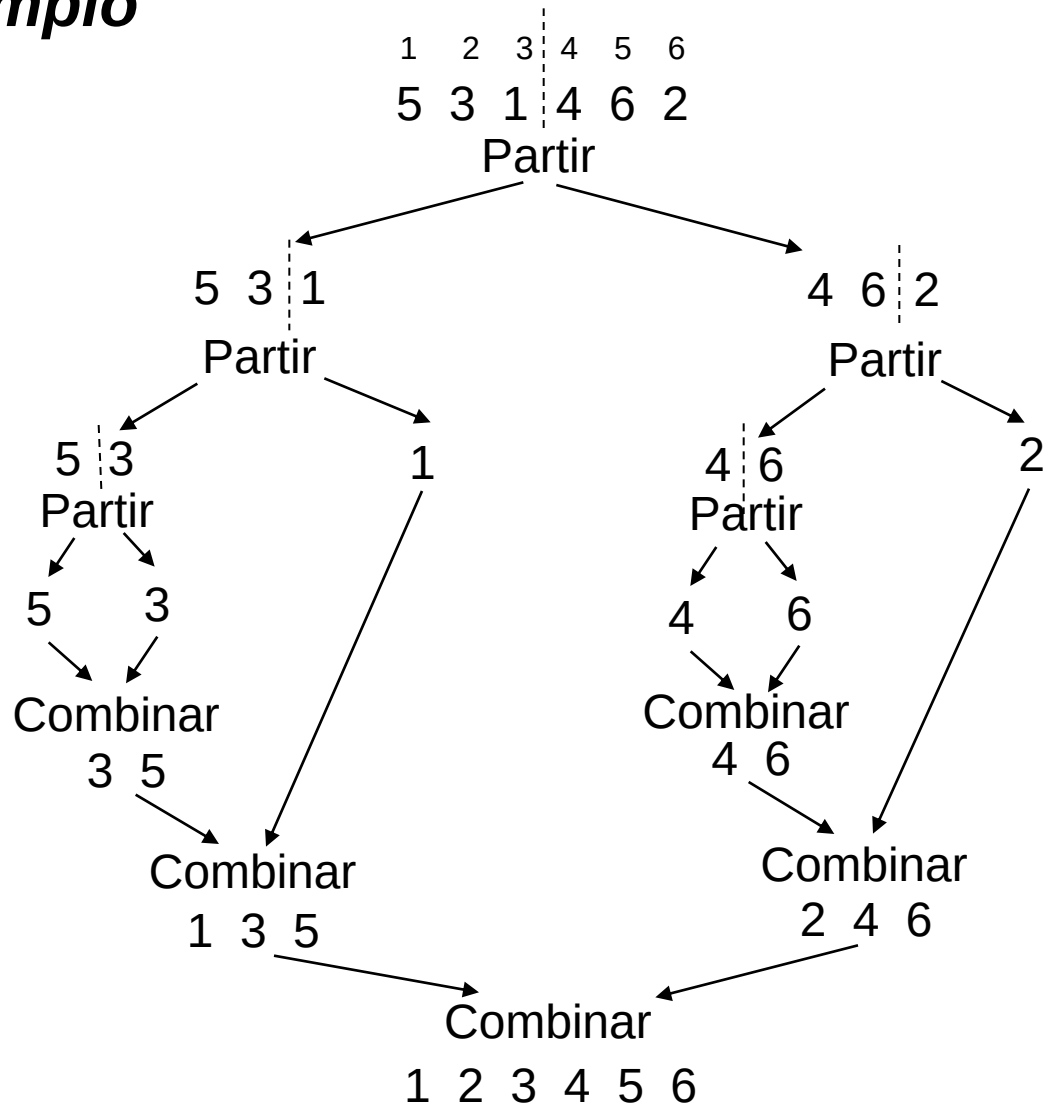
```
    MergeSort(T,P,M);
```

```
    MergeSort(T,M+1,U);
```

```
  devolver Combinar(T,P,M,U); ← Va a requerir memoria dinámica
```

MergeSort

■ *Ejemplo*



MergeSort: Combinar

```
status Combinar(Tabla T, ind P, ind U, ind M)
```

```
    T'=TablaAux(P,U); ← Tabla auxiliar con
```

índices de P a U

```
    Si T'==NULL: devolver Error;
```

```
    i=P;j=M+1;k=P;
```

```
    mientras i≤M y j≤U:
```

```
        → si T[i]<T[j]: T'[k]=T[i];i++;
```

```
        else: T'[k]=T[j];j++;
```

```
        k++;
```

```
    si i>M: // copiar resto de tabla derecha
```

```
        mientras j≤U:
```

```
            T'[k]=T[j];j++;k++;
```

```
    else si: j>U: // copiar resto de tabla izquierda
```

```
        mientras i≤M:
```

```
            T'[k]=T[i];i++;k++;
```

```
    Copiar(T',T,P,U); ←
```

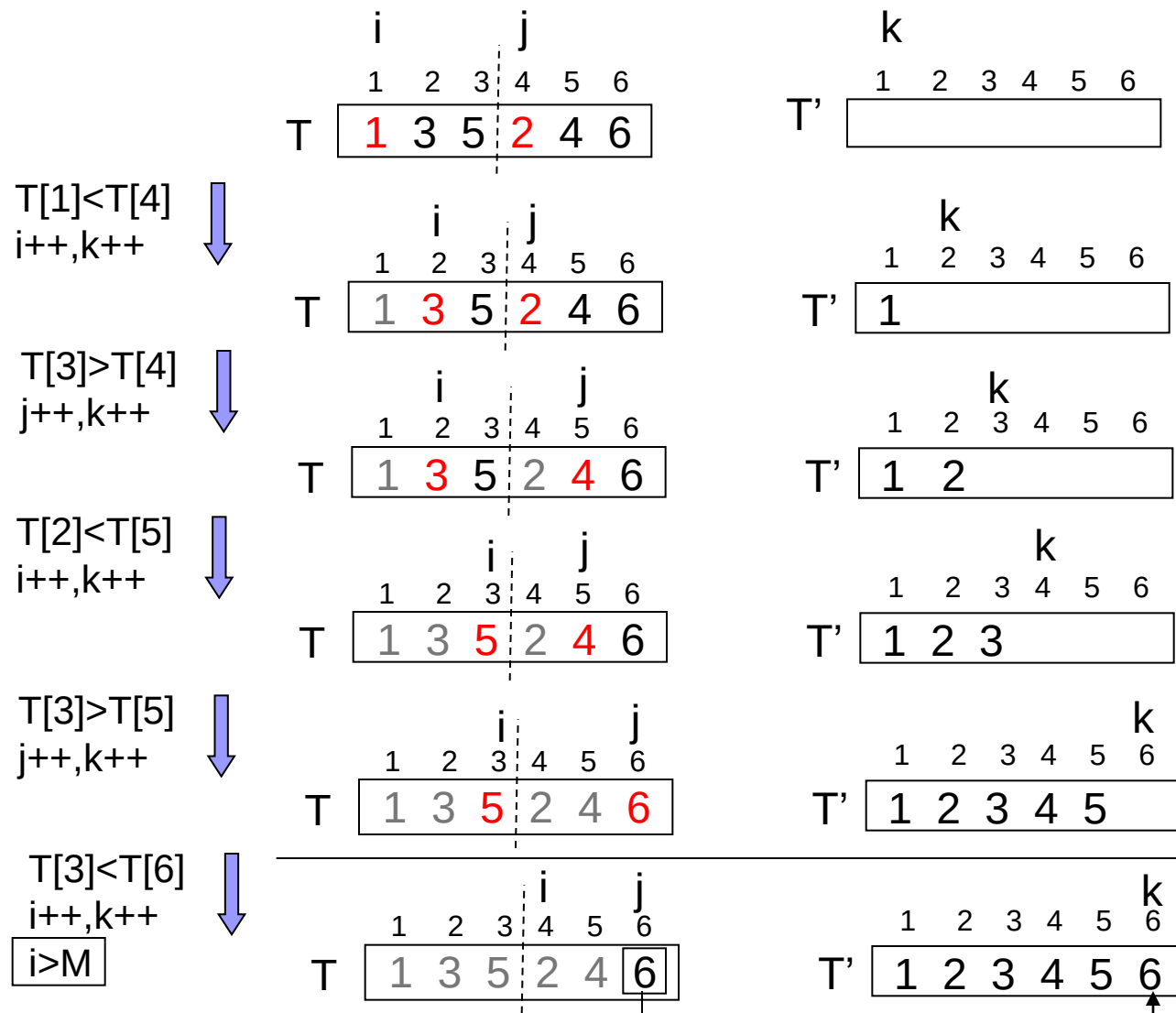
Copia T' en T
entre los índices P y U

```
    Free(T');
```

```
    devolver T;
```

Operación
Básica de
Combinar
y de MS

Combinar: Ejemplo



Finalmente copiamos T' en T y liberamos T'

MergeSort: Rendimiento

■ Observaciones

1. OB: en Combinar $T[i] < T[j]$
2. $n_{MS}(\sigma) = n_{MS}(\sigma_i) + n_{MS}(\sigma_d) + n_{Combinar}(\sigma, \sigma_i, \sigma_d)$
3. $\text{tamaño}(\sigma_i) = \lceil N/2 \rceil$; $\text{tamaño}(\sigma_d) = \lfloor N/2 \rfloor$
4. $\lfloor N/2 \rfloor \leq n_{Combinar}(\sigma, \sigma_i, \sigma_d) \leq N-1$

■ Con estas observaciones se tiene:

$$W_{MS}(N) \leq W_{MS}(\lceil N/2 \rceil) + W_{MS}(\lfloor N/2 \rfloor) + N-1;$$

$$W_{MS}(1) = 0.$$

■ Primer ejemplo de *desigualdad recurrente*.

CG: $T(N) \leq T(N_1) + T(N_2) + \dots + T(N_k) + f(N)$, con $N_i < N$

CB: $T(1) = X$ (X constante).

MergeSort: Rendimiento, caso peor

- ¿ Cómo se resuelve una desigualdad recurrente ?
 - **Paso 1:** Se obtiene una solución particular, por ejemplo sobre un caso particular fácil de calcular.
 - Por ejemplo en el caso de MS, se toma $N=2^k$
 - *En el caso de MS, tomando $N=2^k$ se tiene la desigualdad recurrente*

$$W_{MS}(N) \leq 2W_{MS}(N/2)+N-1 \text{ y } W_{MS}(1)=0$$
 - *Desarrollando en cadena la expresión anterior se obtiene*
$$W_{MS}(N) \leq N \lg(N) + O(N).$$
 - **Paso 2:** Se demuestra que la expresión obtenida en el paso 1 es válida para todo N mediante el método de ***demostración por inducción.***

Nota importante: Se recomienda ver el desarrollo anterior para ambos pasos en clase o en los apuntes de la asignatura.

MergeSort: Rendimiento, casos peor y medio

- Por un razonamiento similar al del caso peor se tiene que $B_{MS}(N) \geq B_{MS}(\lceil N/2 \rceil) + B_{MS}(\lfloor N/2 \rfloor) + \lfloor N/2 \rfloor$ y

$$B_{MS}(1)=0$$

- Tomando, de nuevo $N=2^k$ se obtiene la ecuación recurrente $B_{MS}(N) \geq 2B_{MS}(N/2)+N/2$ y $B_{MS}(1)=0$

- Resolviendo la desigualdad anterior se obtiene:
 $B_{MS}(N) \geq (1/2)N \lg(N)$

- Para estimar el caso medio observamos que:

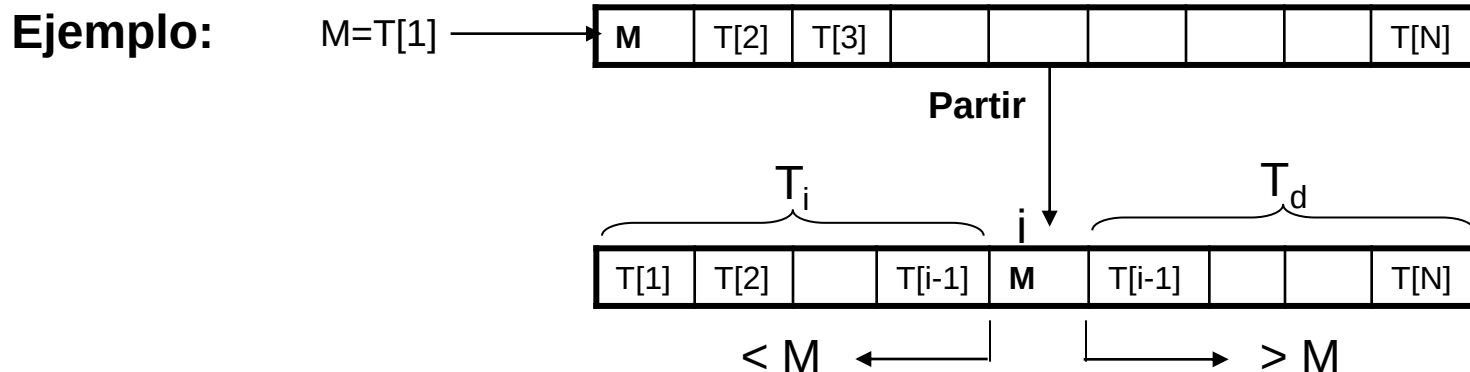
$$(1/2)N \lg(N) \leq B_{MS}(N) \leq A_{MS}(N) \leq W_{MS}(N) \leq N \lg(N) + O(N), \text{ con}$$

lo cual se tiene que: $A_{MS}(N) = \Theta(N \lg(N))$

- El rendimiento es bueno, pero el algoritmo necesita **memoria dinámica** y además tiene **costes ocultos en la recursión**

QuickSort

- En Quicksort (QS), se parte de una función **Partir** complicada que hace innecesaria una función **Combinar**.
- La idea de **Partir** en QS consiste en elegir un elemento $M=T[m]$ de la tabla a ordenar (pivote).
- Tras **Partir** los elementos de la tabla quedan ordenados respecto a M
=>no hace falta Combinar.



QuickSort: pseudocódigos

status QS(tabla T, ind P, ind U)

si $P > U$:

devolver ERROR;

si $P == U$:

devolver OK;

else:

$M = \text{Partir}(T, P, U)$;

si $P < M - 1$:

QS(T, P, M - 1);

si $M + 1 < U$:

QS(T, M + 1, U);

devolver OK;

ind Partir(tabla T, ind P, ind U)

$M = \text{Medio}(T, P, U)$; $\leftarrow$ Pivote

$k = T[M]$;

swap($T[P]$, $T[M]$);

$M = P$;

para i de P + 1 a U:

si $T[i] < k$:

$M++$;

swap($T[i]$, $T[M]$);

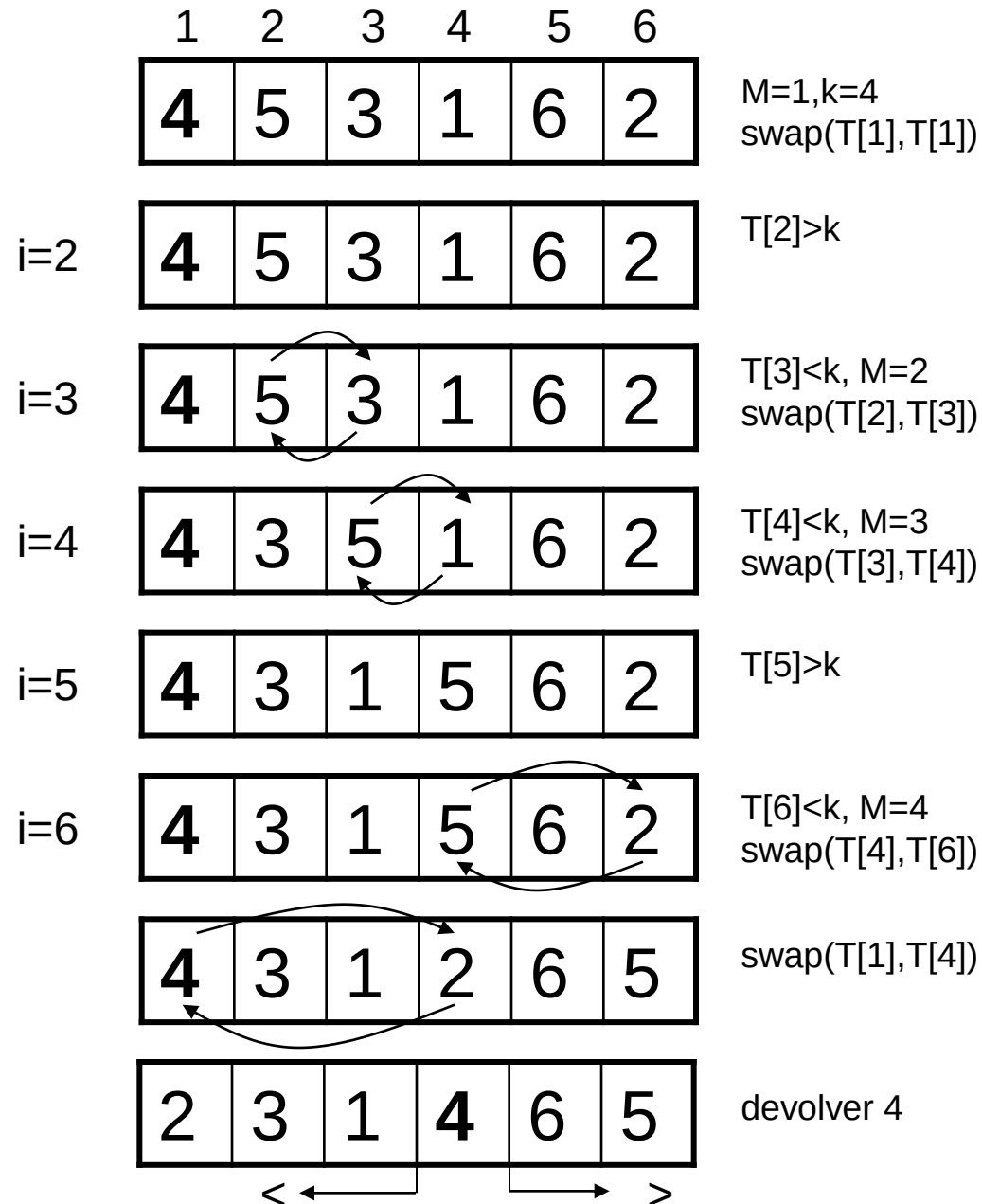
swap($T[P]$, $T[M]$);

devolver M;

QuickSort: Elección del pivote

- El pivote se puede elegir de varias maneras:
 - El primer elemento (devolver P).
 - El último elemento (devolver U).
 - La posición que está en mitad de la tabla (devolver $(P+U)/2$).
 - Una posición aleatoria entre el primer y ultimo elemento de la tabla (devolver $\text{aleat}(P,U)$).
- **No se garantiza** que el valor del pivote sea aproximadamente el valor medio de la tabla

Ejemplo



QS: Rendimiento en el caso peor

- Observaciones

1. OB: en Partir “si $T[i] < k$ ”
2. $n_{QS}(\sigma) = n_{QS}(\sigma_i) + n_{QS}(\sigma_d) + n_{Partir}(\sigma)$
3. $n_{Partir}(\sigma) = N-1$ (Si Medio devuelve P, $n_{Medio}(\sigma) = 0$)

- Entonces, si σ_i tiene k elementos, σ_d tiene $N-1-k$ elementos y por tanto

$$\begin{aligned} n_{QS}(\sigma) &\leq N-1 + W(k) + W(N-1-k) \\ &\leq N-1 + \max_{k=1, \dots, N-1} \{ W(k) + W(N-1-k) \} \end{aligned}$$

- Esto es,

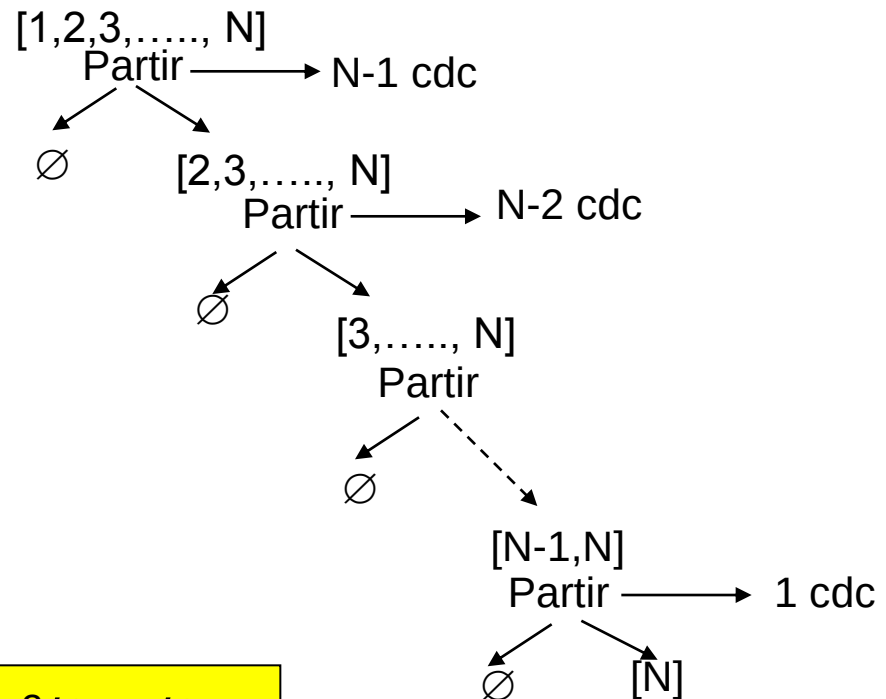
$$W(N) \leq N-1 + \max_{k=1, \dots, N-1} \{ W(k) + W(N-1-k) \}$$

- Y se puede demostrar por inducción que

$$W(N) \leq N^2/2 - N/2$$

QS: Rendimiento en el caso peor

- Pero además $W_{QS}(N) \geq N^2/2 - N/2$.
Consideramos $T=[1,2,3,\dots, N]$



- Por tanto se tiene:
$$n_{QS}([1,2,3,\dots,N]) = (N-1) + (N-2) + \dots + 1 = N^2/2 - N/2$$

Luego

$$W_{QS}(N) = N^2/2 - N/2$$

QS: Rendimiento en el caso medio

- De nuevo tenemos $n_{QS}(\sigma) = n_{QS}(\sigma_i) + n_{QS}(\sigma_d) + N - 1$.
- Aproximamos $n_{QS}(\sigma_i) \cong A_{QS}(i-1)$ y $n_{QS}(\sigma_d) \cong A_{QS}(N-i)$ con lo que obtenemos $n_{QS}(\sigma) \cong A_{QS}(i-1) + A_{QS}(N-i) + N - 1$.
- Y obtenemos la igualdad recurrente aproximada

$$A_{QS}(N) = (N - 1) + \frac{1}{N} \sum_{i=1}^N [A_{QS}(i-1) + A_{QS}(N-i)]$$

$$A(1) = 0$$

- Se puede demostrar

$$A_{QS}(N) = 2N \log(N) + O(N)$$

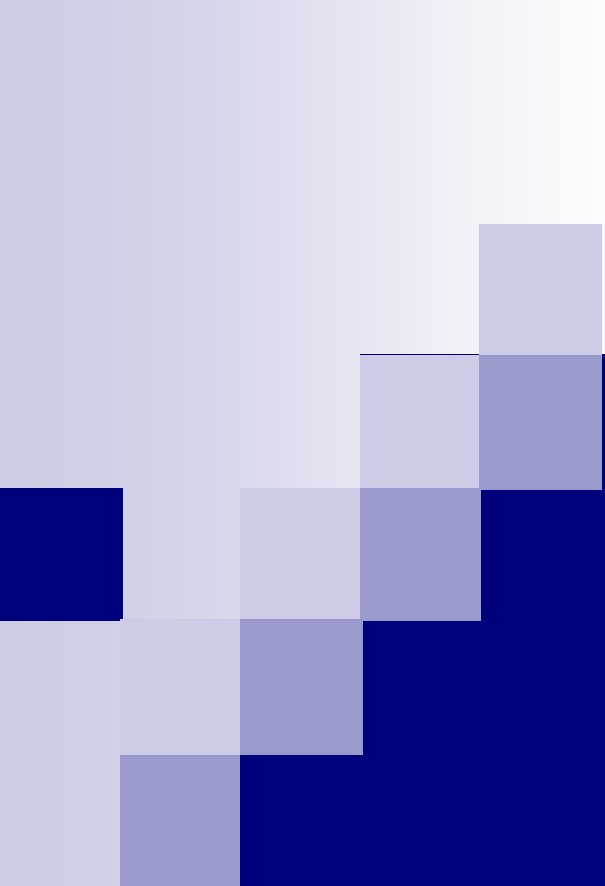
Nota: Se recomienda seguir la demostración de lo anterior en la pizarra o en los apuntes de la asignatura.

En esta sección hemos aprendido...

- Los algoritmos Quick y MergeSort
- Sus ecuaciones de rendimiento en los casos peor y medio
- Cómo resolverlas
- Cómo escribir la ecuación de rendimiento de un algoritmo recursivos
- Cómo efectuar estimaciones de ecs. recurrentes
 - Intuyendo soluciones particulares en algunos casos
 - Desplegando para estimar una solución general o particular
 - Estimando el caso general por inducción

Herramientas y técnicas a trabajar ...

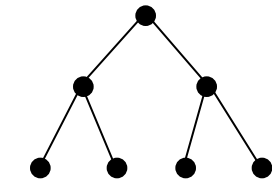
- Funcionamiento de MergeSort y QuickSort
- Casos peor, medio y mejor de MS y QS y variantes
- Estimación del crecimiento de funciones en desigualdades recurrentes
- Determinación de ecuaciones de rendimiento de algoritmos recurrentes y su resolución
- Problemas a resolver (al menos!!) :
63, 64, 65, 66, 67, 68, 69, 70, 71, 74, 75, 76,
78, 79



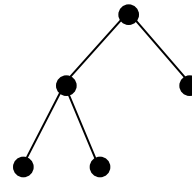
2.3 HeapSort

HeapSort

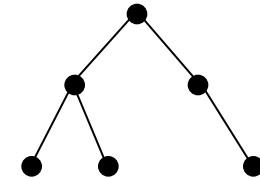
- **Definición:** Un heap (montón) es un árbol binario quasicompleto (es decir, solo tiene huecos en los elementos más a la derecha del último nivel).



Árbol binario completo



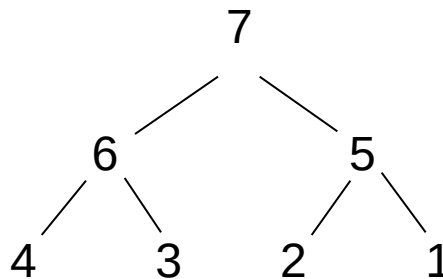
Árbol binario quasicompleto (heap)



Árbol binario (no heap)

- **Definición:** Un Max-heap es un heap tal que $\forall$ subárbol T' de T se tiene $\text{info}(T') > \text{info}(T'_i), \text{info}(T'_d)$

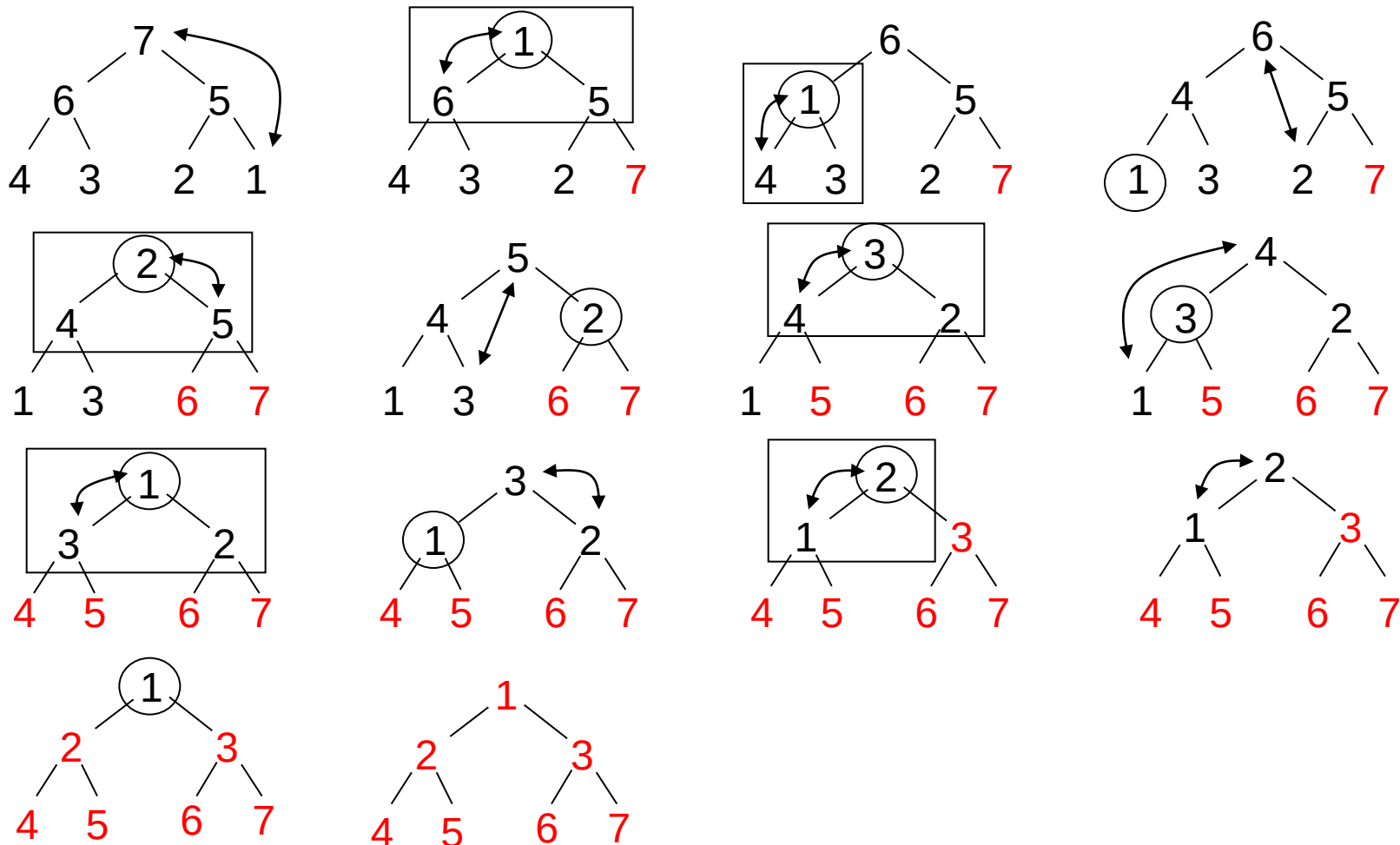
Ejemplo:



- **Observación:** Un max-heap es fácil de ordenar.

Ordenación de max-heap.

1. Se intercambia el nodo de la raíz con el nodo inferior derecho
2. Se mantiene la condición de max-heap con el nodo recién colocado en la raíz.

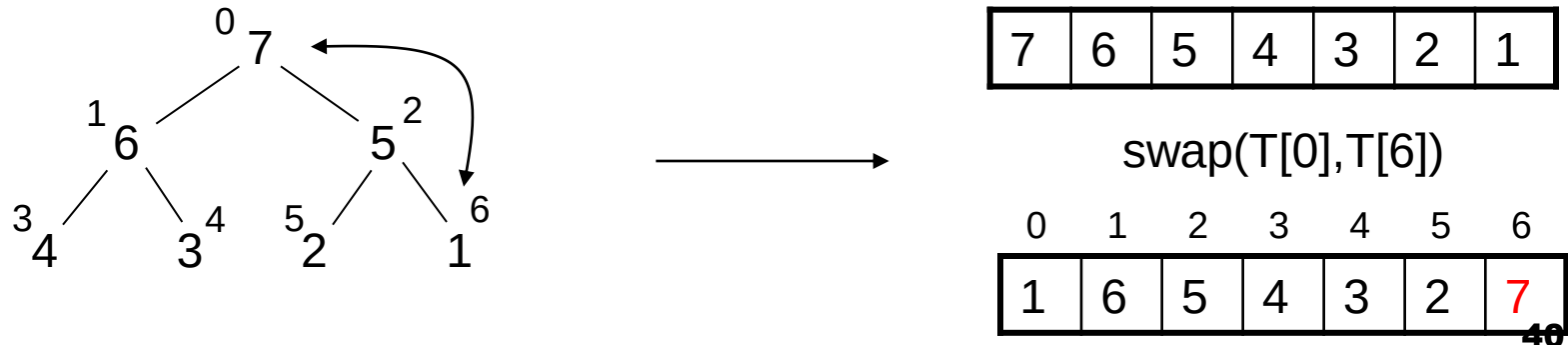
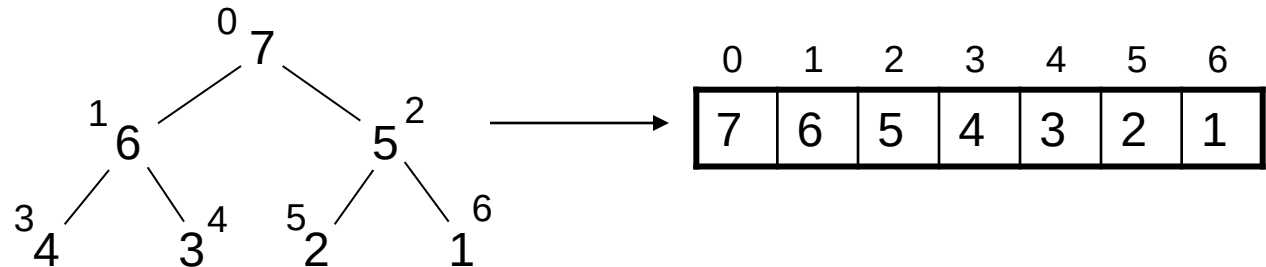


HeapSort: Ordenación en tablas

■ Observaciones:

1. Si recorremos el árbol resultante al final de arriba a abajo y de izquierda a derecha se tiene una tabla ordenada
2. Los nodos de un heap se pueden colocar en una tabla de tal forma que el método sea in-place.

Ejemplo:



HeapSort: Posiciones en tablas que contienen max-heaps

- Padre $\rightarrow$ Hijo izquierdo y Padre $\rightarrow$ Hijo derecho

P	H _I	H _D
0	1	2
1	3	4
2	5	6
...	...	...



Padre	Hijo Izq	Hijo der
j	$2j+1$	$2j+2$

- Hijo $\rightarrow$ Padre

H	P
1	0
2	0
3	1
4	1
5	2
6	2
...	...

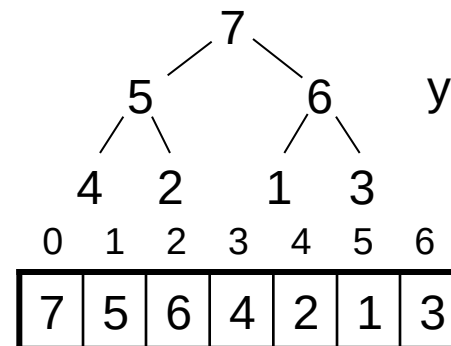
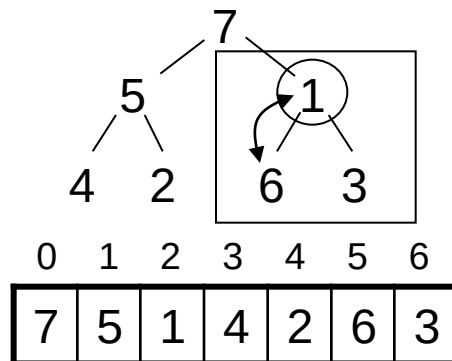
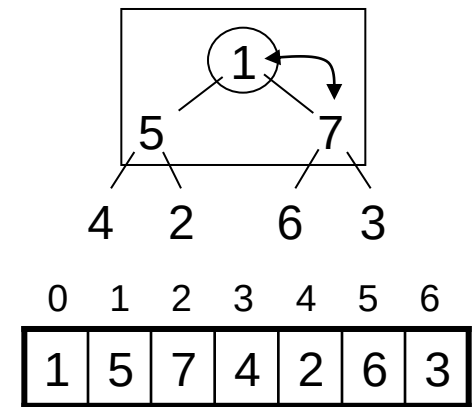
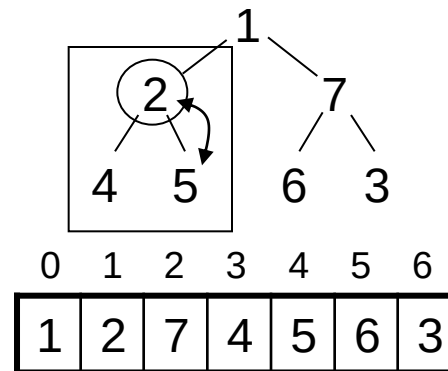
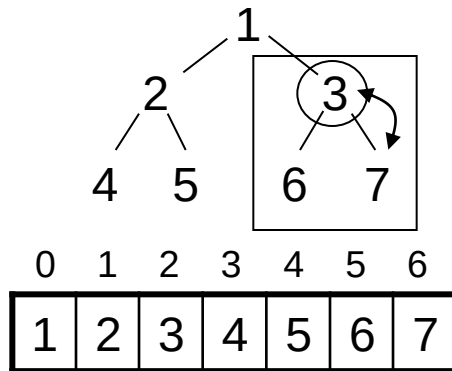


Hijo	Padre
j	$\lfloor (j-1)/2 \rfloor$

HeapSort: Creación del heap

- El proceso anterior permite ordenar un heap
- ¿Cómo podemos crear un heap a partir de una tabla dada?
 - Se mantiene la condición de heap en todos los nodos internos (nodos que tienen al menos un hijo), de derecha a izquierda y de abajo arriba.

Ejemplo:



ya es un heap

HeapSort: Pseudocódigo

— HeapSort —

```
HeapSort(tabla T, int N)
  CrearHeap(T,N);
  OrdenarHeap(T,N);
```

— CrearHeap —

```
CrearHeap(tabla T, int N)
  si N==1:
    volver ;
  para i de  $\lfloor (N-2)/2 \rfloor$  a 0 :
    heapify(T,N,i);
```

— OrdenarHeap —

```
OrdenarHeap(tabla T, int N)
  para i de N-1 a 1 :
    swap(T[0],T[i]);
    heapify(T,i,0);
```

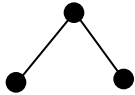
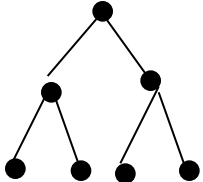
— heapify —

```
heapify(tabla T,int N,ind i)
  mientras (  $2*i+1 \leq N-1$  ) :
    ind=max(T,N,i,izq(i),der(i));
    if (ind  $\neq$  i) :
      swap( A[i], A[ind] ) ;
      i = ind ;
    else :
      return;
```

Donde la función $\text{max}(T,N,i,\text{izq}(i),\text{der}(i))$ devuelve el índice del elemento de la tabla T que contiene el valor mayor entre i, izq(i), der(i)

Profundidad de Heaps

- **Observación:** Si T es un heap con N nodos, $\text{prof}(T) = \lfloor \log(N) \rfloor$

Nº de nodos N	Ejemplo de Heap	Prof.
1		0
2, 3		1
4, 5, 6, 7		2
...		...

HeapSort: Rendimiento

■ Observaciones:

- $n_{\text{HeapSort}}(T) = n_{\text{CrearHeap}}(T) + n_{\text{OrdenarHeap}}(T)$
- El número máximo de cdc que CrearHeap y OrdenarHeap realizan sobre un nodo es $\text{prof}(T)$
- $\text{prof}(T) = \lfloor \log(N) \rfloor$ pues T es quasi-completo

■ $n_{\text{CrearHeap}}(T) \leq N \lfloor \log(N) \rfloor$ y $n_{\text{OrdenarHeap}}(T) \leq N \lfloor \log(N) \rfloor$

■ $W_{\text{HS}}(N) = O(N \log(N))$

■ $n_{\text{CrearHeap}}([1, 2, \dots, N]) = N \log(N)$

$W_{\text{HS}}(N) = \Theta(N \log(N))$

■ El método seguido no es recursivo

■ Es el método de ordenación más eficaz hasta ahora!

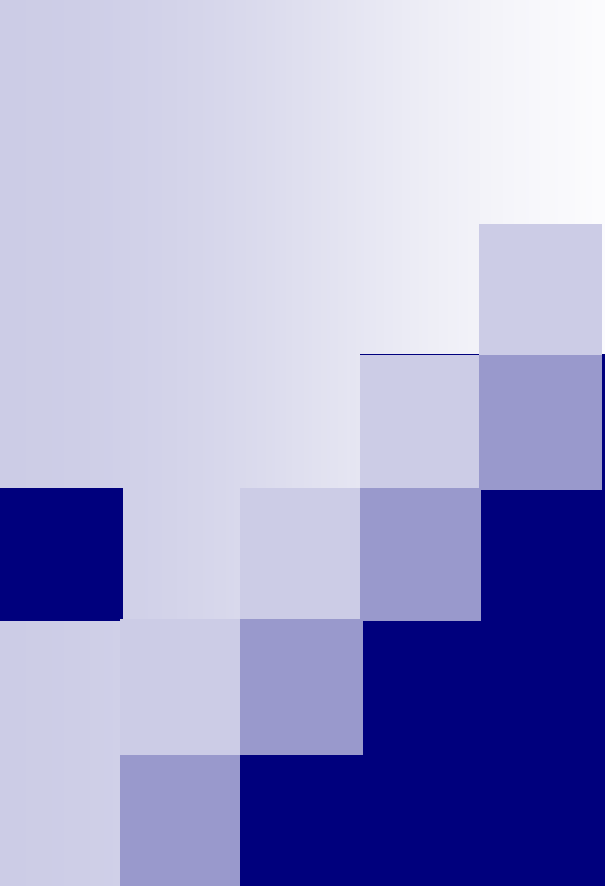


En esta sección hemos aprendido...

- El concepto de Maxheap y su construcción
- El algoritmo HeapSort y su rendimiento

Herramientas y técnicas a trabajar

- La construcción de Maxheaps
- La aplicación del algoritmo HeapSort
- Problemas a resolver (al menos!!) :
83, 90



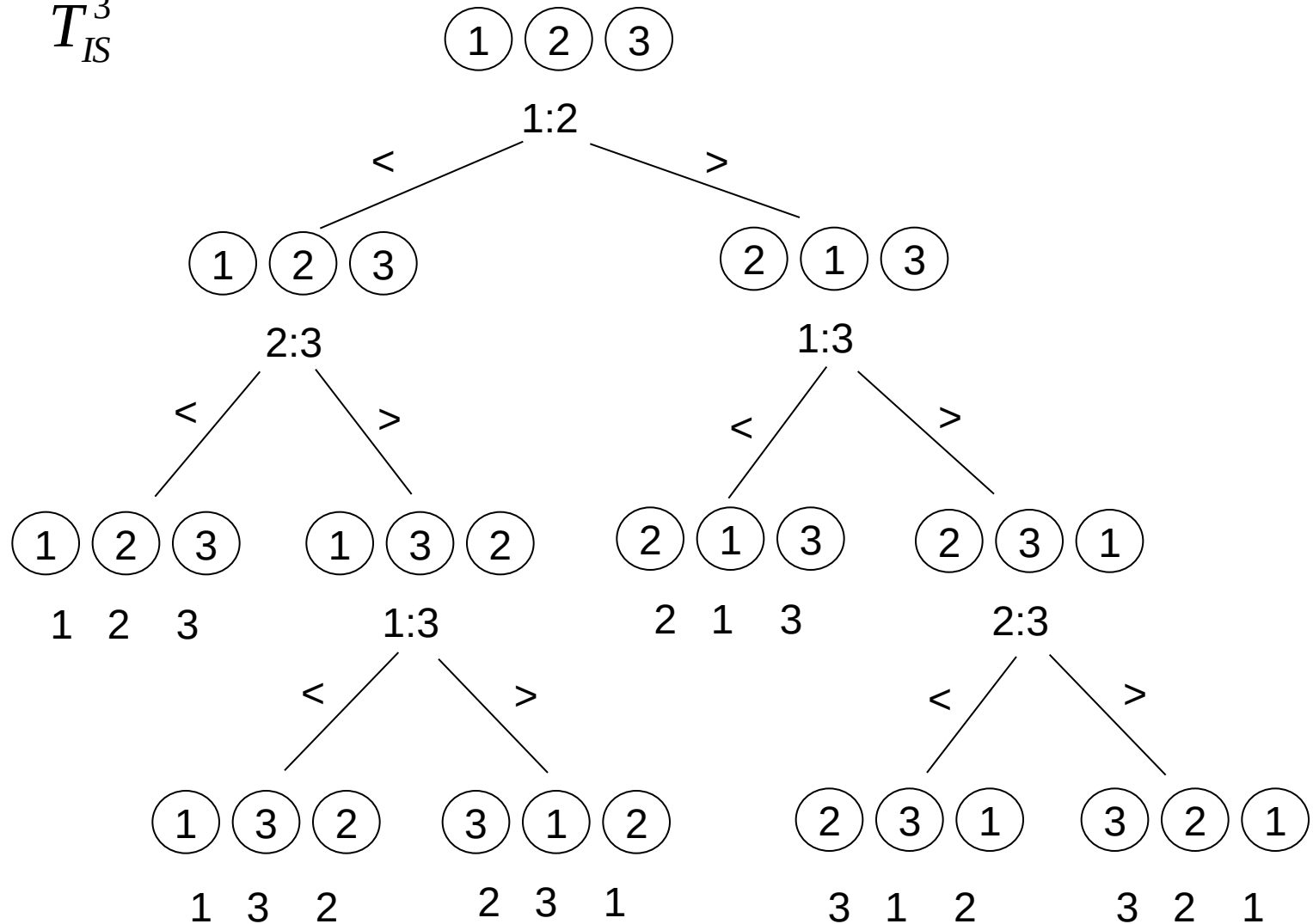
2.4 Árboles de decisión para algoritmos de ordenación

Cotas inferiores para algoritmos de ordenación por cdcs

- Hasta ahora el mejor algoritmo de ordenación por cdcs es HeapSort
- Ningún algoritmo de ordenación va a tener un coste mejor que $\Theta(N)$
- **Pregunta:** ¿hay algoritmos de ordenación de coste mejor que $\Theta(N \log(N))$?
- **Respuesta: NO** al menos si trabajamos sobre cdcs
- Herramienta: **árboles de decisión**

Árbol de decisión: Ejemplo InsertSort

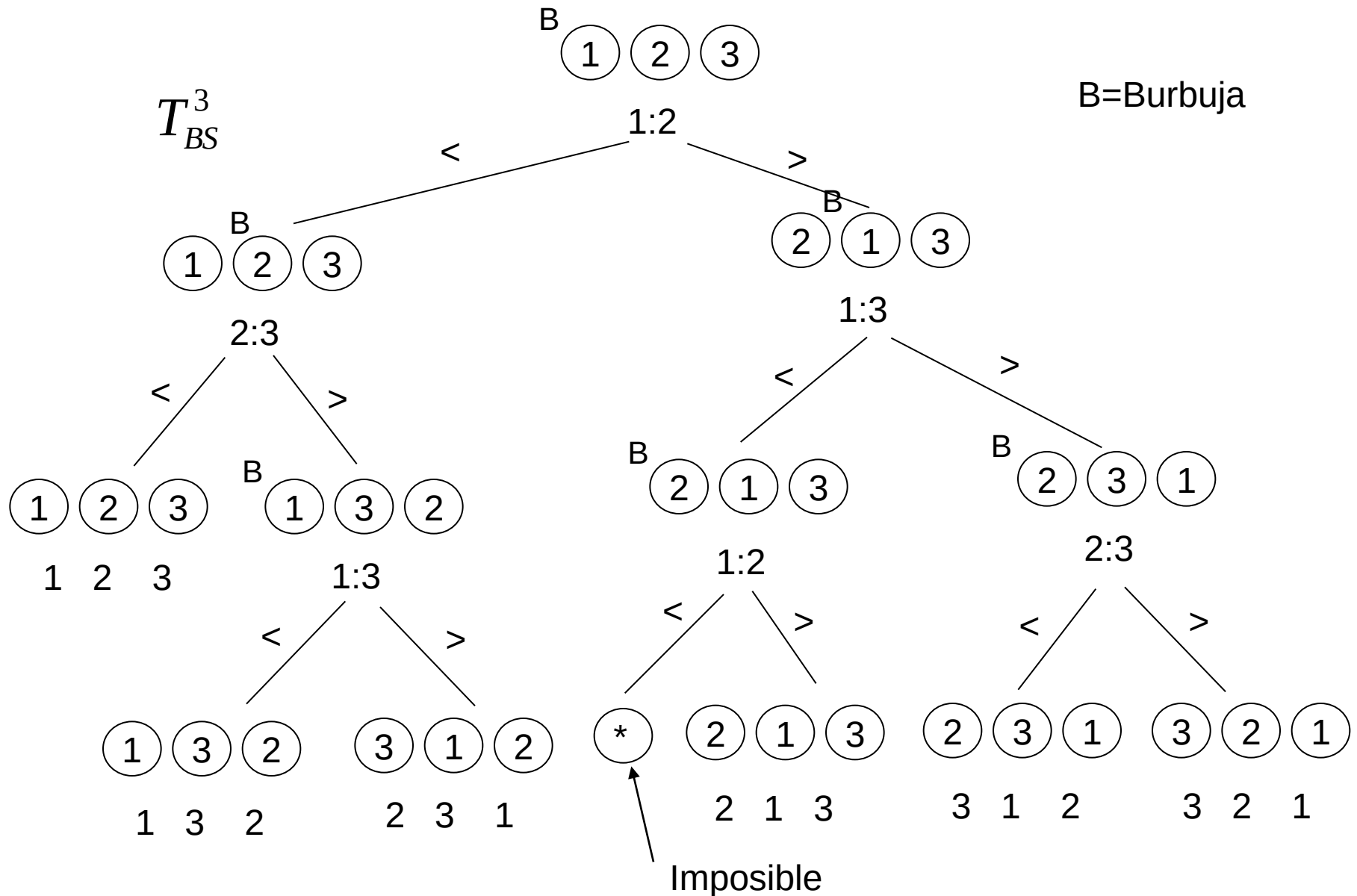
T_{IS}^3



Árbol de decisión: definición

- Si **A** es un algoritmo de ordenación por comparación de clave y **N** es un tamaño de tabla, se puede construir su **árbol de decisión** T_A^N sobre Σ_N cumpliendo las siguientes 4 condiciones:
 1. Contiene nodos de la forma $\boxed{i:j}$ ($i < j$) que indica la cdc entre los elementos **inicialmente** en las posiciones **i** y **j**.
 2. El subárbol izquierdo de $i:j$ en T_A^N contiene el resto del trabajo (cdcs) que realiza el algoritmo **A** si $i < j$.
 3. El subárbol derecho de $i:j$ en T_A^N contiene el el resto del trabajo (cdcs) que realiza el algoritmo **A** si $i > j$.
 4. A cada $\sigma \in \Sigma_N$ le corresponde una única **hoja** H_σ en T_A^N y los **nodos entre la raíz y la hoja** H_σ son las **sucesivas cdc** que realiza el algoritmo **A** al recibir la permutación σ para ordenarla.

Ejemplo de árbol de decisión: BurbujaSort



Árbol de decisión: consecuencias

1. El número de hojas en T_A^N es $N! = |\Sigma_N|$.
2. $n_A(\sigma) = n^\circ$ de cdc = profundidad de la hoja H_σ en T_A^N

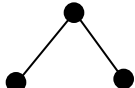
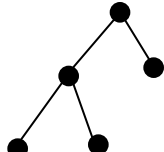
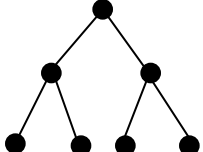
$$n_A(\sigma) = \text{prof}_{T_A^N}(H_\sigma)$$

3. Por tanto:

$$W_A(N) = \max_{\sigma \in \Sigma_N} n_A(\sigma) = \max_{\sigma \in \Sigma_N} \text{prof}_{T_A^N}(H_\sigma)$$

Cota inferior en el caso peor I

- ¿Cuál es la **profundidad mínima** de un árbol binario de H hojas?

Nº de hojas H	$AB^{\text{Minimo}}(H)$	Prof.
1		0
2		1
3		2
4		2
....		

Cota inferior en el caso peor II

- Parece que la Prof Mínima de un AB con H hojas es $\lceil \log(H) \rceil$
- Para el caso peor se tiene:

$$\begin{aligned} W_A(N) &= \max_{\sigma \in \Sigma_N} \text{prof}_{T_A^N}(H_\sigma) \geq \text{prof mín de AB con } N! \text{ hojas} \\ &= \lceil \log(N!) \rceil \end{aligned}$$

- Como sabemos que $\log(N!) = \Theta(N \log(N))$ se tiene que $W_A(N) = \Omega(N \log(N))$.
- HeapSort y MergeSort son **óptimos para el caso peor**.

Cota inferior en el caso medio I

- Tenemos

$$A_A(N) = \frac{1}{N!} \sum_{\sigma \in \Sigma_N} n_A(\sigma) = \frac{1}{N!} \sum_{H \in T_A^N} \text{prof}_{T_A^N}(H)$$

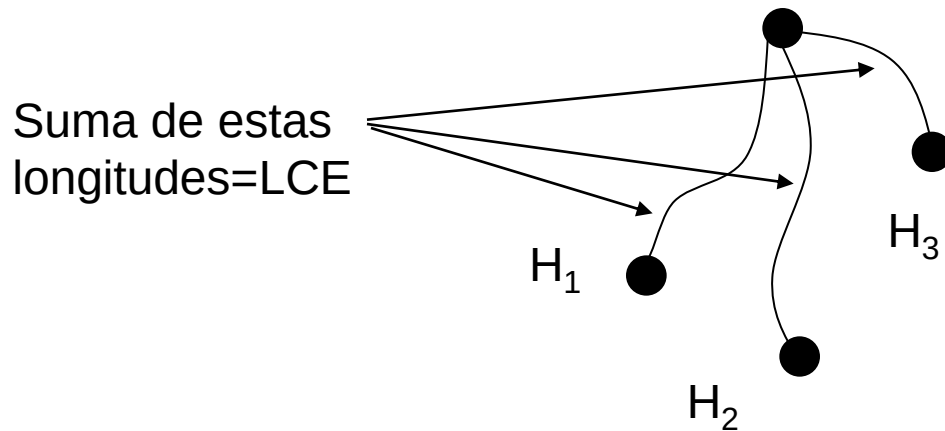
- Luego $A_A(N) \geq PM_{\min}(N)$ donde

$$PM_{\min}(k) = \min \{ \text{prof media}(T) : T \text{ AB con } k \text{ hojas} \}$$

- Pero $\text{prof media}(T) = \frac{1}{k} \sum_{H \in \text{hojas de } T} \text{prof}(H) = \frac{1}{k} LCE(T)$

- **LCE: Longitud de Caminos Externos**

Cota inferior en el caso medio II



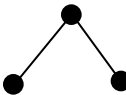
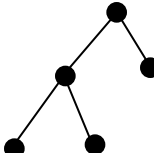
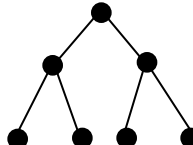
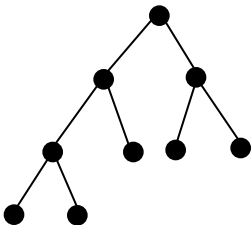
■ Tenemos

$$A_A(N) \geq \frac{1}{N!} LCE_{\min}(N!) \text{ con}$$

$$LCE_{\min}(k) = \min \{LCE(T) : T \text{ tiene } k \text{ hojas}\}$$

Cota inferior en el caso medio III

- Estimamos $LCE_{\min}(k)$

k	T Óptimo	$LCE_{\min}(k)$
1		0
2		$2(1+1)$
3		$5(2+2+1)$
4		$8(2+2+2+2)$
5		$12(3+3+2+2+2)$
....		

Cota inferior en el caso medio IV

- Se puede demostrar (ver apuntes o clase)

$$LCE_{\min}(k) = k \lceil \log(k) \rceil + k - 2^{\lceil \log(k) \rceil}$$

- Dado que

$$A_A(N) \geq \frac{1}{N!} LCE_{\min}(N!) =$$

$$\frac{1}{N!} \left(N! \lceil \log(N!) \rceil + N! - 2^{\lceil \log(N!) \rceil} \right) = \lceil \log(N!) \rceil + 1 - \frac{2^{\lceil \log(N!) \rceil}}{N!} =$$

$$\lceil \log(N!) \rceil = \Omega(N \log(N))$$

- Se sigue que

$$A_A(N) = \Omega(N \log(N))$$

- MS, QS y HS son **óptimos** para el **caso medio**.



En esta sección hemos aprendido...

- El concepto de **árbol de decisión** para un algoritmo de ordenación por cdc
- A **construir** un árbol de decisión para un algoritmo de ordenación por CDC.
- Las **cotas inferiores** para los algoritmos de ordenación por CDC
- **Cómo se obtienen** mediante el uso de árboles de decisión.



Herramientas y técnicas a trabajar

- La construcción de Árboles de Decisión para tablas de 3 elementos
- La construcción de Árboles de Decisión parciales para tablas de 4 elementos
- Problemas a resolver (al menos!!) :
92, 96, 97, 100, 101, 102, 103, 106