ORDENAMIENTO POR MEZCLA

Merge Sort

Ordenamiento por mezcla

- Conceptualmente, el ordenamiento por mezcla funciona de la siguiente manera:

1. *Si la longitud de la lista es 0 o 1, entonces ya está ordenada. En otro caso:*
2. *Dividir la lista desordenada en dos sublistas de aproximadamente la mitad del tamaño.*
3. *Ordenar cada sublista recursivamente aplicando el ordenamiento por mezcla.*
4. *Mezclar las dos sublistas en una sola lista ordenada.*

- El algoritmo de ordenación por mezcla sigue de cerca el paradigma de divide y vencerás.

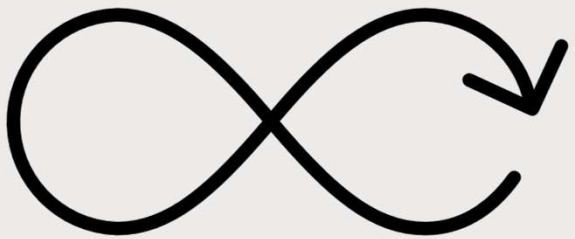
  Intuitivamente, funciona de la siguiente manera:

- **Dividir**: Divide la secuencia de **n** elementos para clasificar en dos subsecuencias de **n/2** elementos cada una.

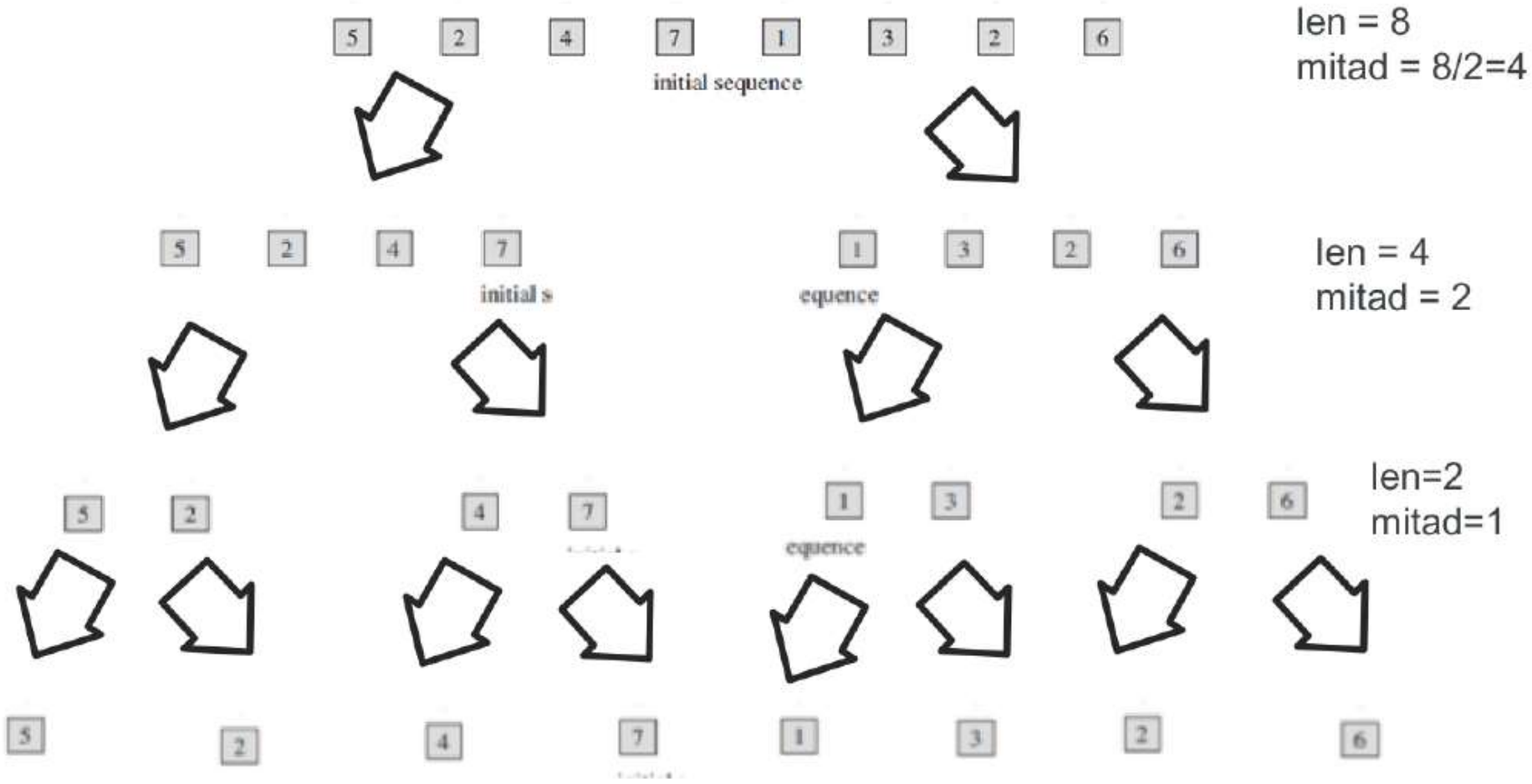
- **Conquistar**: Ordena las dos subsecuencias de forma recursiva utilizando merge sort.

- **Combinar**: Combina las dos subsecuencias ordenadas para producir la respuesta ordenada.

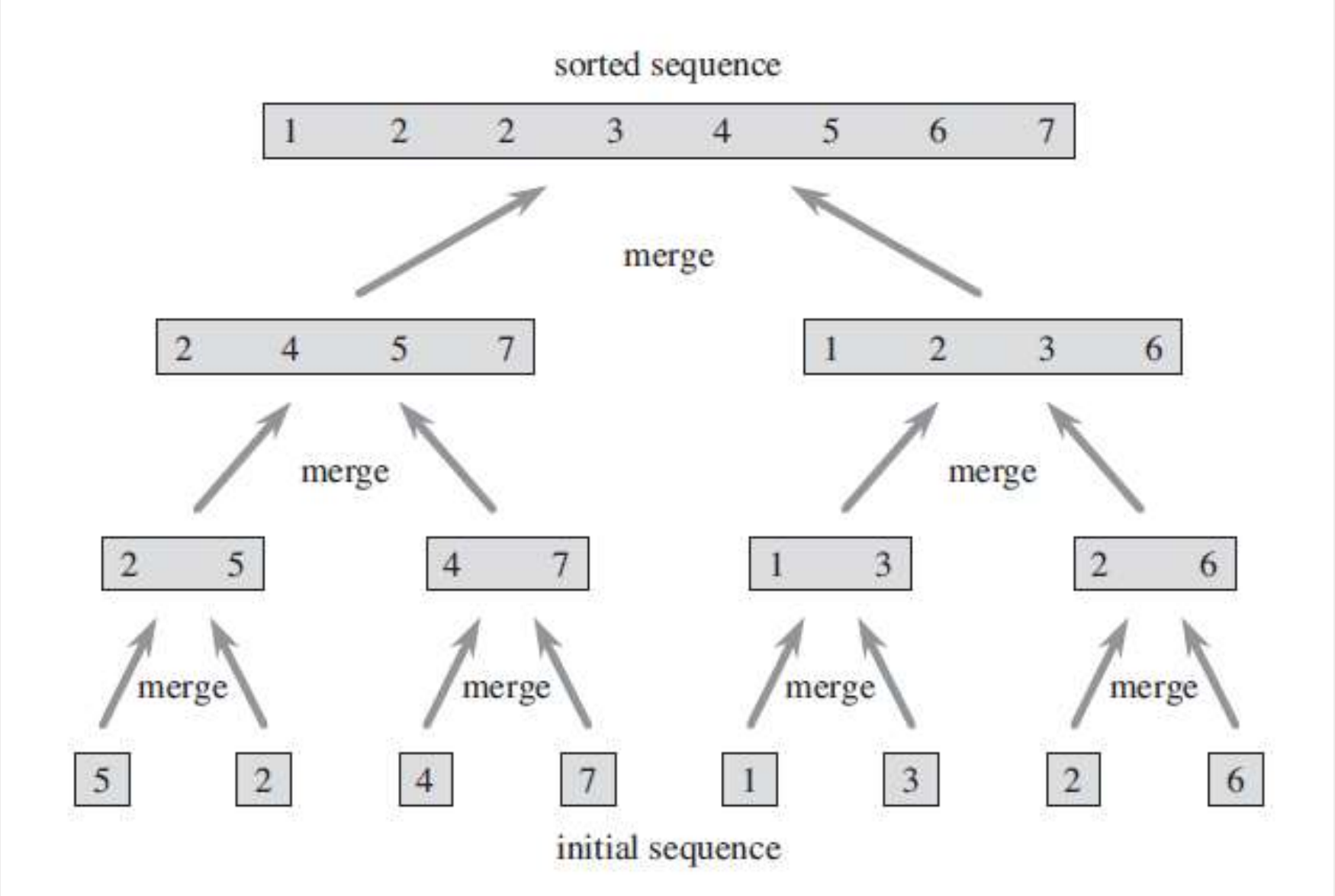
- La recursividad "toca fondo" cuando la secuencia que se va a ordenar tiene longitud de 1, en tal caso ya no hay trabajo por hacer, ya que cada secuencia de longitud 1 ya está ordenada.



Ejemplo
Generación del árbol



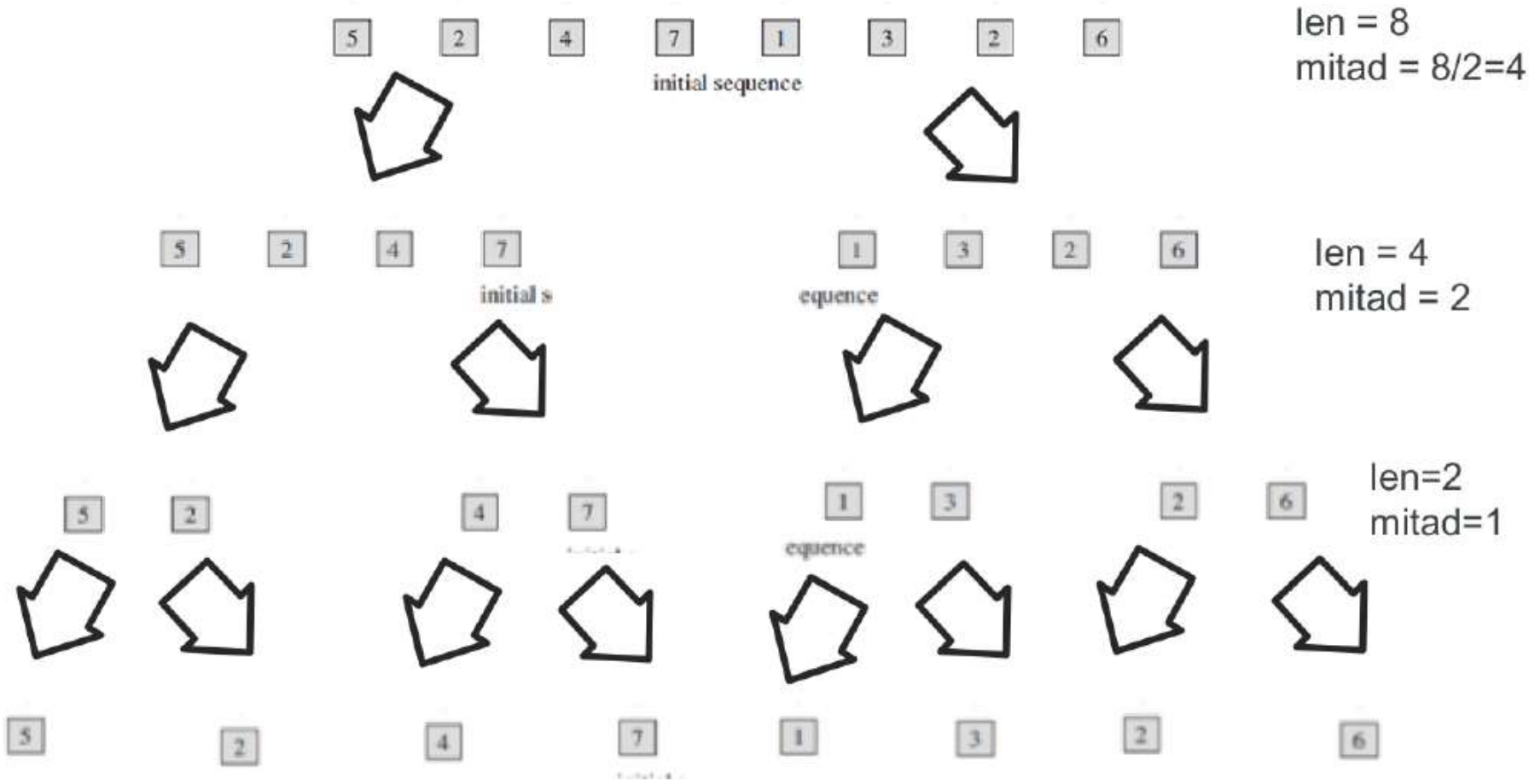
Ejemplo
Combinación de secuencias



Función Merge sort

- El procedimiento MERGE-SORT(A, p, r) ordena los elementos en el subarreglo A[p..r].
- Si p ≥ r, el subarreglo tiene como máximo un elemento y, por lo tanto, ya está ordenado.
- De lo contrario, el paso de división simplemente calcula un índice *q* que divide A[p..r] en dos subarreglos:
  - *A[p..q], que contiene* $\lceil n/2 \rceil$ *elementos.*
  - *A[q+1...r] que contiene* $\lceil n/2 \rceil$ *elementos.*

```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```



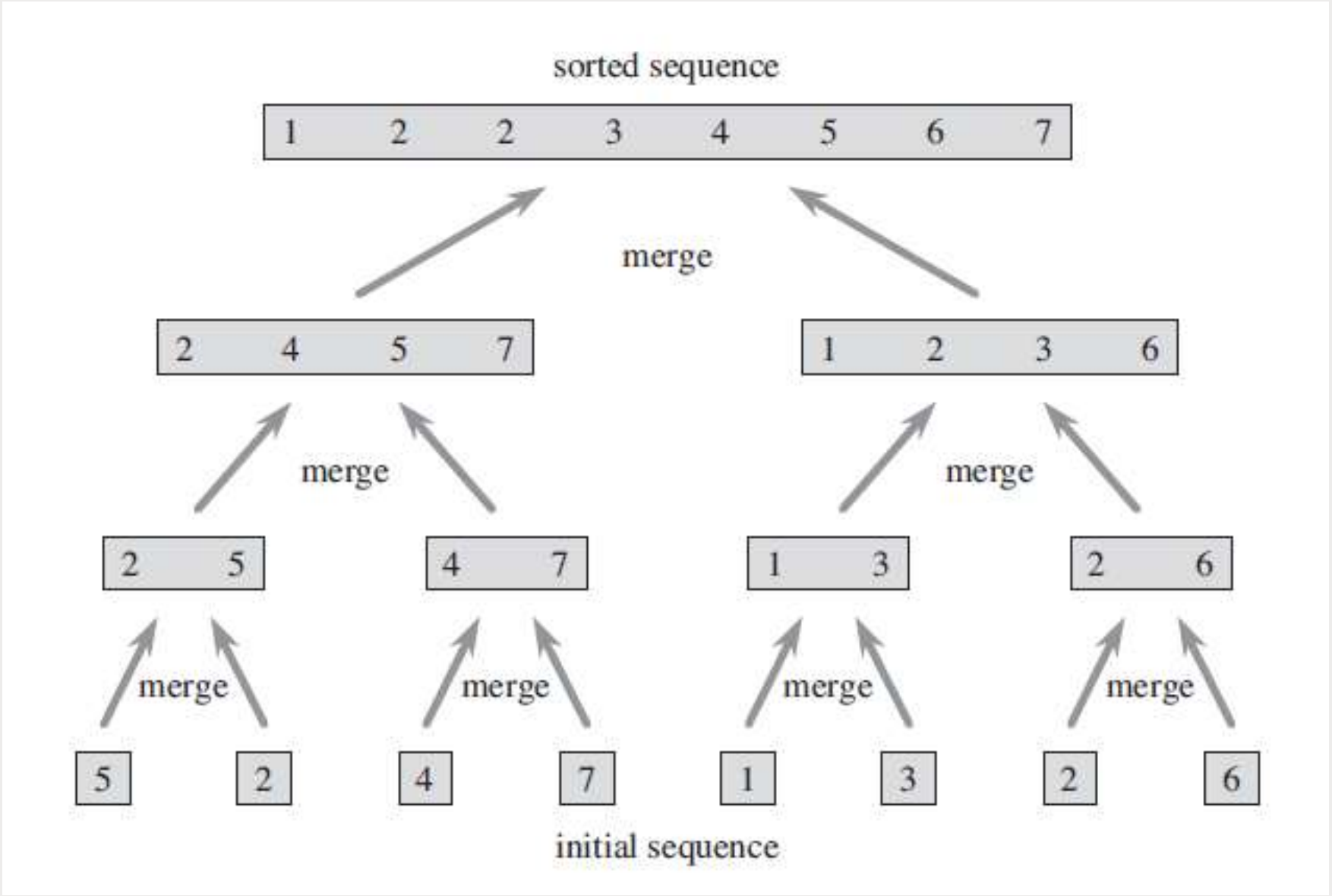
Si len<=1 entonces se termina la recursividad

Descripción general del pseudocódigo merge

- Supongamos que tenemos dos pilas de cartas boca arriba sobre una mesa.
- Cada pila esta ordenada, con las tarjetas más pequeñas en la parte superior.
- Deseamos combinar las dos pilas en una sola pila de salida ordenada, que debe estar boca abajo sobre la mesa.
- Nuestro paso básico consiste en elegir la más pequeña de las dos cartas en la parte superior de las pilas boca arriba, quitarla de su pila (que expone una nueva carta superior), y colocando esta carta boca abajo en la pila de salida.
- Repetimos este paso hasta que una pila de entrada esté vacía, momento en el que simplemente tomamos la pila de entrada restante y la colocamos boca abajo en la pila de salida.

- Implementando la idea anterior, pero con un giro adicional que evita tener que comprobar si alguna pila está vacía en cada paso básico.
- Colocamos en la parte inferior de cada pila una carta centinela, que contiene un valor especial que usamos para simplificar nuestro código.
- Aquí, usamos ∞ como valor centinela, de modo que siempre que una carta con ∞ esté expuesta, no puede ser la carta más pequeña a menos que ambas pilas expongan sus cartas de centinela.
- Pero una vez que eso sucede, todas las cartas no centinelas ya se han colocado en la pila de salida.
- Sabemos que exactamente $r - p + 1$ cartas serán colocadas en la pila de salida, y entonces podemos detenernos.

Ejemplo


sorted sequence
1 2 2 3 4 5 6 7
merge
2 4 5 7
1 2 3 6
merge
merge
2 5
4 7
1 3
2 6
merge
merge
merge
merge
5
2
4
7
1
3
2
6
initial sequence

Merge

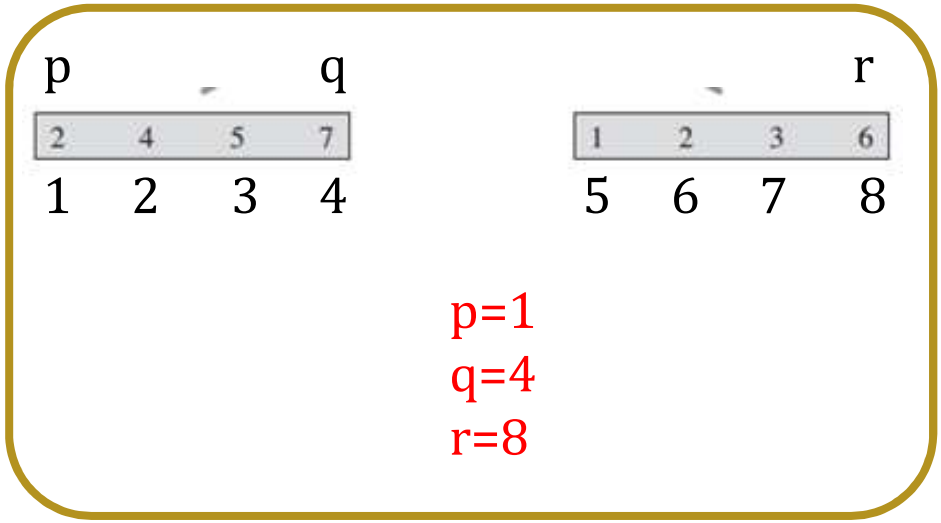
- La operación clave del algoritmo del ordenamiento por mezcla es la combinación de las dos secuencias en el paso "combinar".

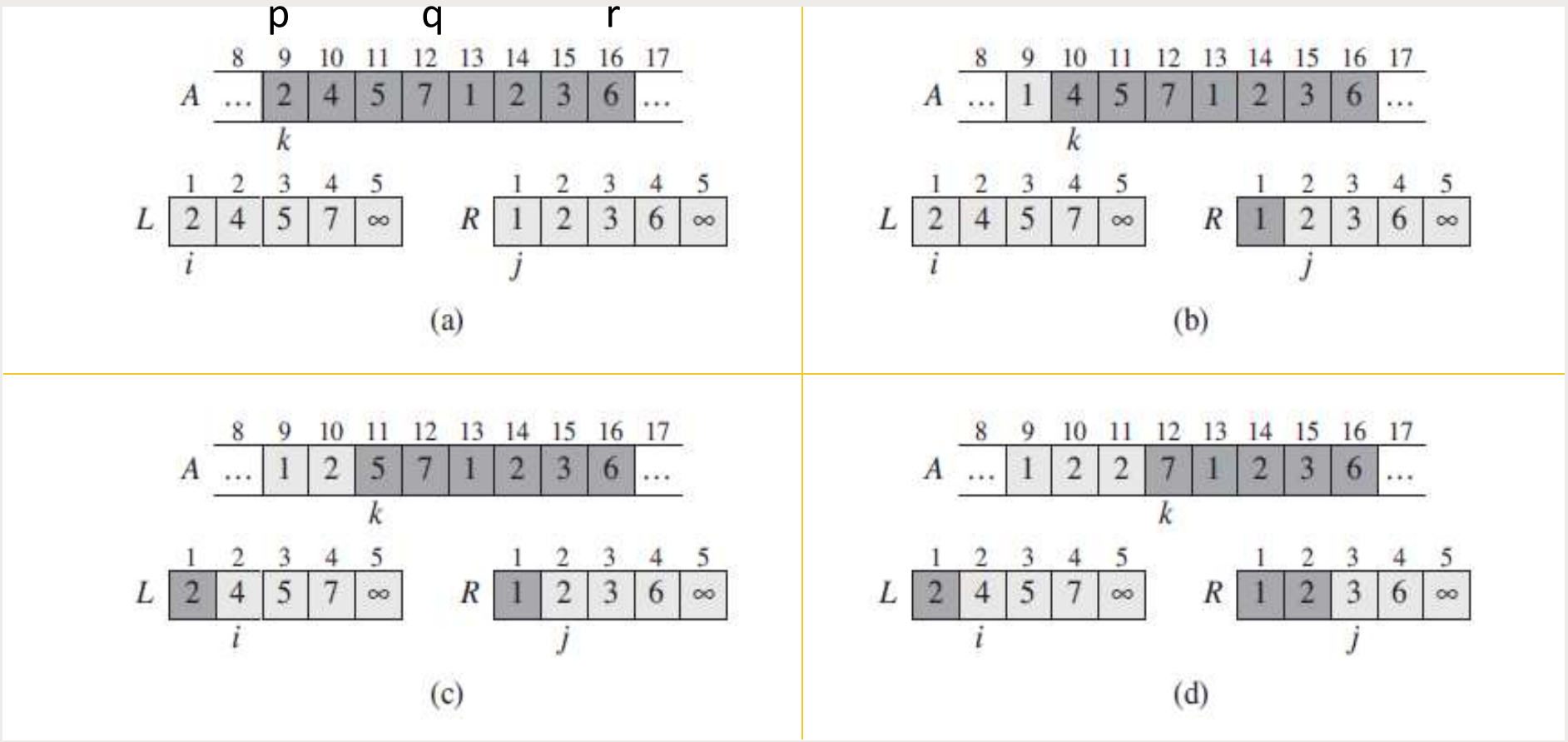
- Las combinamos llamando a un procedimiento auxiliar MERGE(A, p, q, r), donde:

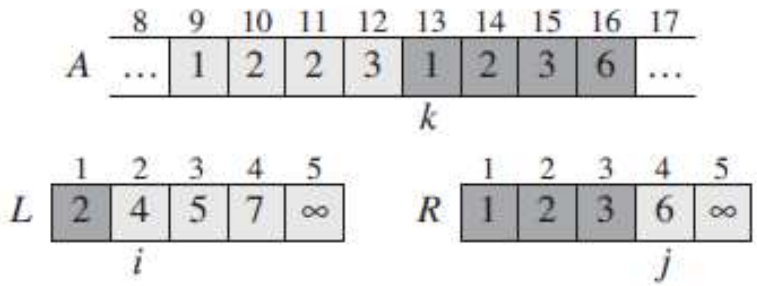
  - ***A*** *es un arreglo.*
  - ***p***, ***q*** *y* ***r*** *son índices en el arreglo.*
  - *Los subarreglos* ***a[p...q]*** *y* ***A[q+1...r]*** *ya están ordenados.*
  - *Combina los subarreglos para formar un único subarreglo ordenado que reemplaza al subarreglo actual* ***A[p...r]***.

Pseudocódigo MERGE

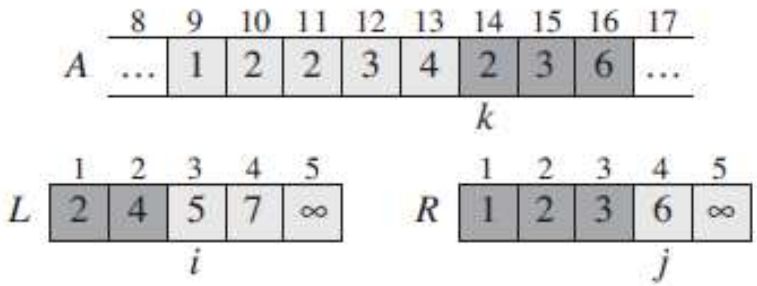
```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
 4  for i = 1 to n1
 5      L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7      R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```



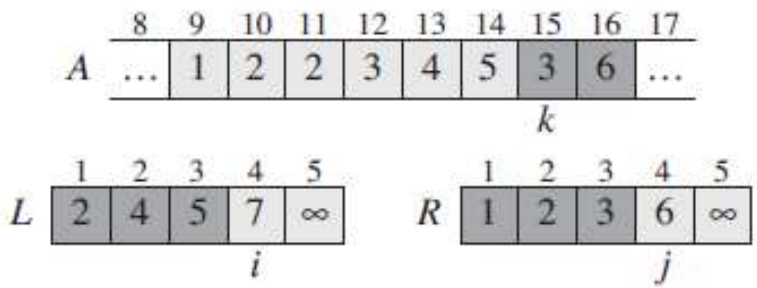

p
q
r
A
L
R
i
j
k
(a)
(b)
(c)
(d)


8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

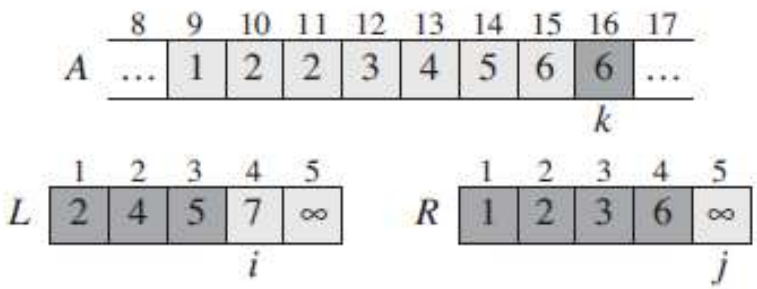
(e)


8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

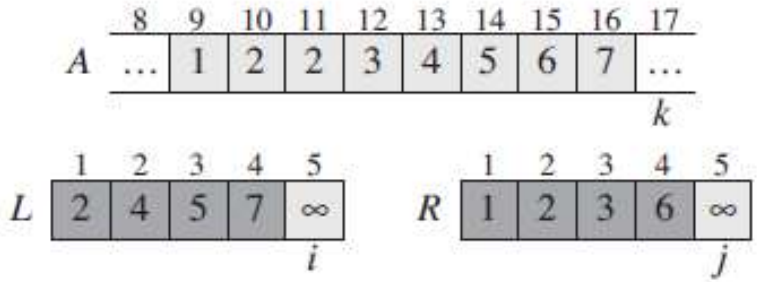
(f)


8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 5 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

(g)


8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 5 6 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

(h)


8 9 10 11 12 13 14 15 16 17
A … 1 2 2 3 4 5 6 7 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

(i)

Ordenamiento por mezcla

```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1 .. n1 + 1] and R[1 .. n2 + 1]
 4  for i = 1 to n1
 5      L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7      R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```

Complejidad algorítmica del algoritmo de ordenamiento por mezcla

$$O(n \lg n)$$