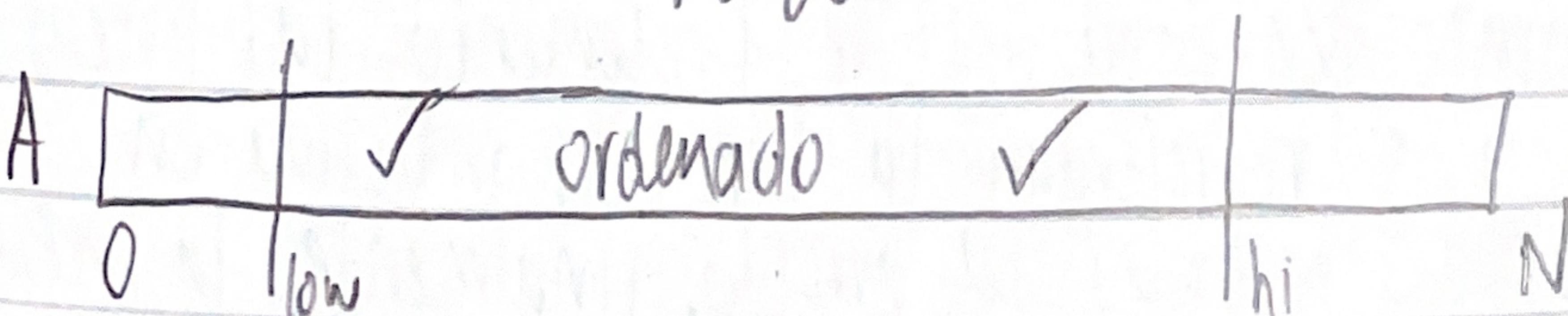
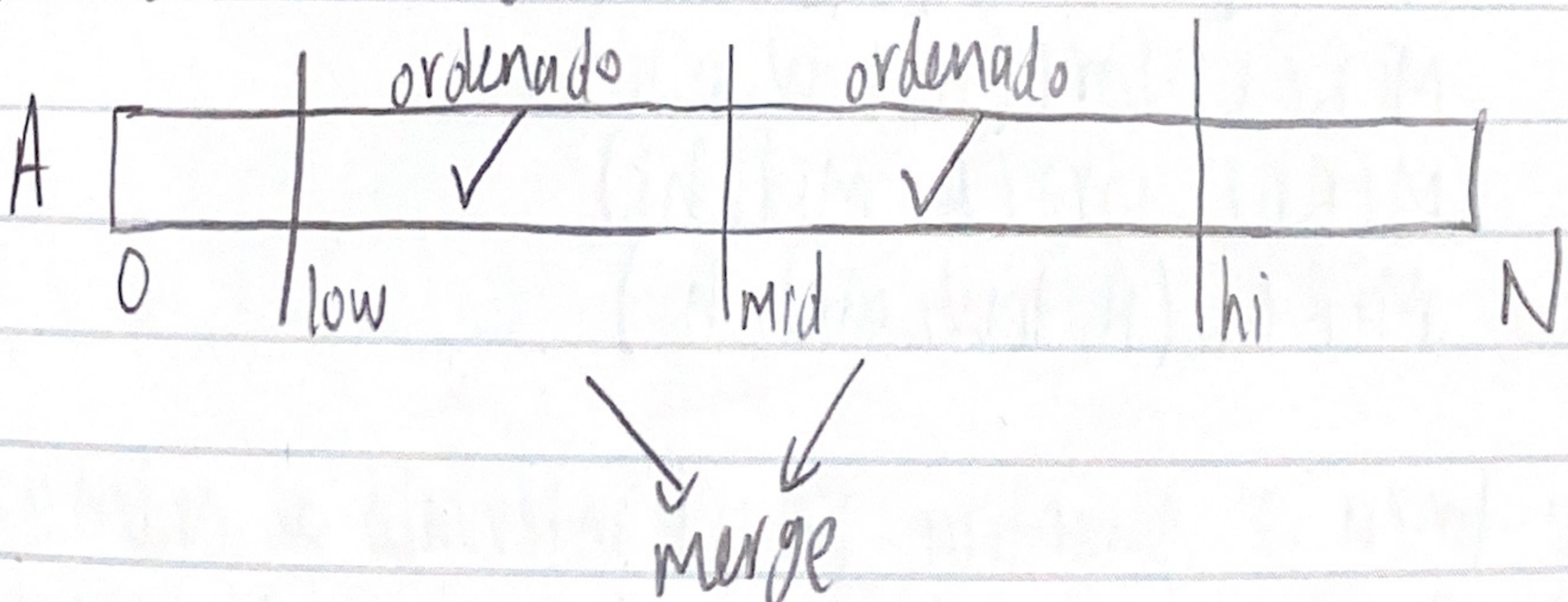


# Merge sort

El algoritmo merge sort hace uso de una hipótesis induktiva de la siguiente manera:

- Cada subarreglo se divide en dos mitades, resultando en problemas más pequeños que el original.
- Se ordenan estas dos mitades recurrentemente, para luego asumir induktivamente que ya se encuentran ordenadas ascendentemente.
- Se combinan ambas partes ordenadas del arreglo, resultando en un arreglo totalmente ordenado

De manera general, el ordenamiento se puede visualizar como sigue:



Según los límites definidos anteriormente, los valores iniciales de low y hi para ordenar el arreglo completo deberían ser  $0$  y  $N$ , respectivamente, donde  $N$  representa el tamaño del arreglo. Note que el valor hi no es inclusivo.

De esta manera, se va a implementar una

función  $\text{MERGE-SORT}(A, \text{low}, \text{hi})$  que ordena el subarreglo  $A[\text{low} \dots \text{hi}]$ . Esta función puede

ser usada para ordenar el arreglo completo  $A[0 \dots N]$  al ser llamada como  $\text{MERGE-SORT}(A, 0, N)$ .

A continuación se presenta un Pseudocódigo:

1 FUNC MERGE-SORT( $A$ ,  $\text{low}$ ,  $\text{hi}$ )

2 IF  $\text{low} + 1 < \text{hi}$  then

3      $\text{Mid} = \lfloor (\text{low} + \text{hi}) / 2 \rfloor$

4     MERGE-SORT( $A$ ,  $\text{low}$ ,  $\text{Mid}$ )

5     MERGE-SORT( $A$ ,  $\text{Mid}$ ,  $\text{hi}$ )

6     MERGE( $A$ ,  $\text{low}$ ,  $\text{Mid}$ ,  $\text{hi}$ )

La linea 2 garantiza que el intervalo de ordenamiento contenga más de 1 elemento. Las líneas 4 y 5 representan la hipótesis inductiva en las dos mitades del arreglo. Finalmente, la linea

6 combina las dos mitades preservando el orden. Esta última función que combina ambas mitades puede ser implementada como se muestra a continuación.

Nota: El arreglo original es modificado.  
in-place

```
1 FUNC MERGE(A, low, mid, hi):
2     COPY A into an auxiliar array.
3     i = 0
4     l = low //initial index of first half
5     r = mid //initial index of second half.
6     while i < hi do
7         IF l = mid then
8             A[i] := aux[r]; r := r+1
9         ELSE IF r = hi then
10            A[i] := aux[l]; l := l+1
11        ELSE:
12            IF aux[l] ≤ aux[r] then
13                A[i] := aux[l]; l := l+1
14            ELSE
15                A[i] := aux[r]; r := r+1
16            i := i+1
```

la linea 2 crea una copia del arreglo original,  
ya que este será modificado in-place. Lo más  
óptimo para este paso sería crear la copia  
en un espacio de memoria que ya haya  
sido previamente reservado, ya que esto evita-  
rsa la reserva de memoria recurrente.

Las líneas 7 y 8 representan el caso en el  
cual todos los elementos de la primera mitad  
ya han sido previamente asignadas al arreglo  
original y solamente la segunda mitad debe  
ser procesada. De manera similar, las líneas  
9 y 10 representan el caso en el cual todos  
los elementos de la segunda mitad ya han  
sido previamente asignados al arreglo original  
y solamente la primer mitad debe ser  
procesada.

Finalmente, las líneas 11-15 representan el  
caso en el cual ambas mitades tienen elementos  
restantes, donde debemos asignar el elemento  
menor en su posición actual del arreglo  
original para de esta manera preservar el  
orden ascendente.