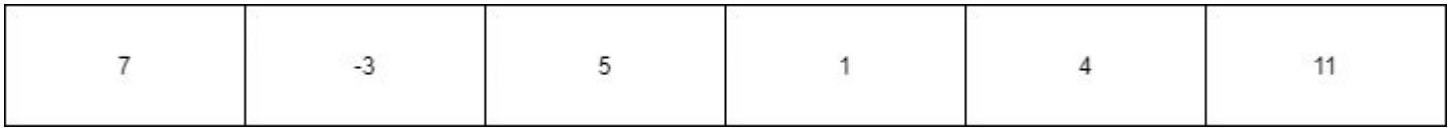
# Ayudantia 3

Sergio M Appel : matamalaappels@uc.cl
Sergio Gutiérrez : sergio.gutierrez@uc.cl

# Divide and Conquer

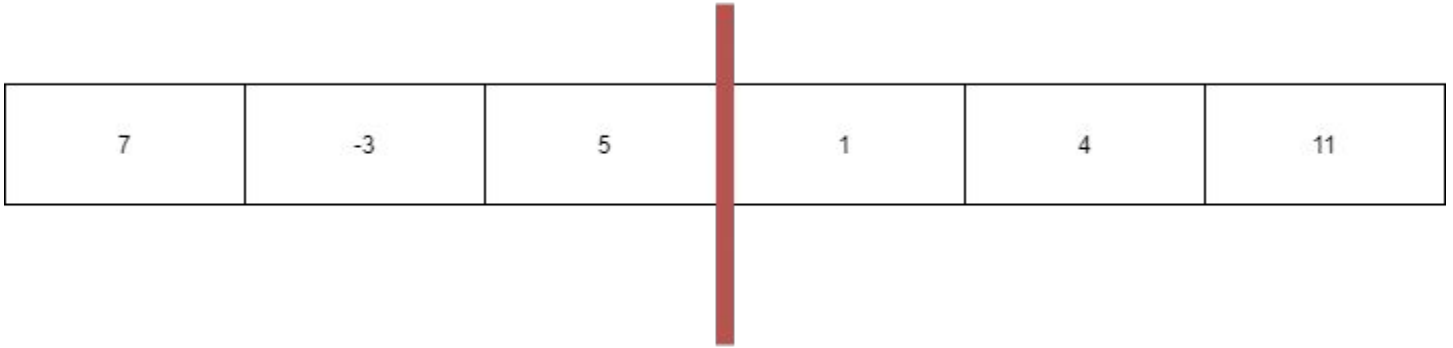
- Merge-sort
- Quick-sort

# Merge-Sort

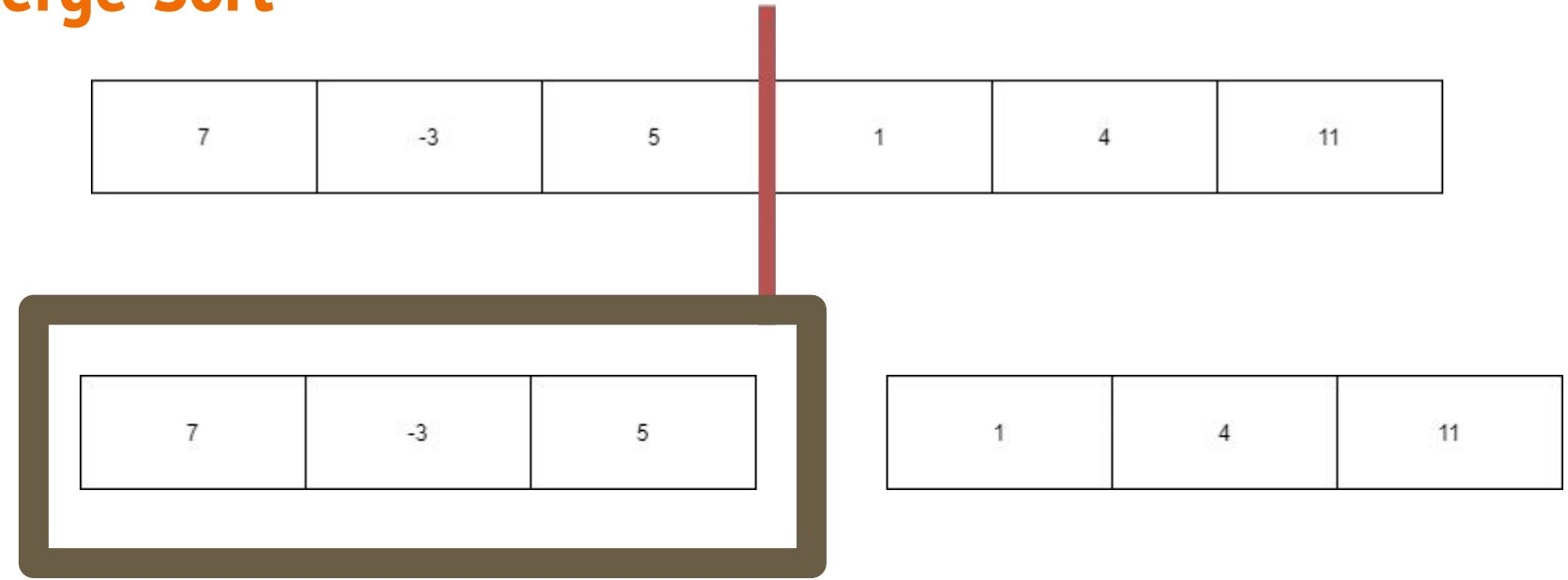



- Dividir nuestro array en 2
- Ordenar cada mitad de forma recursiva
- Unir las mitades

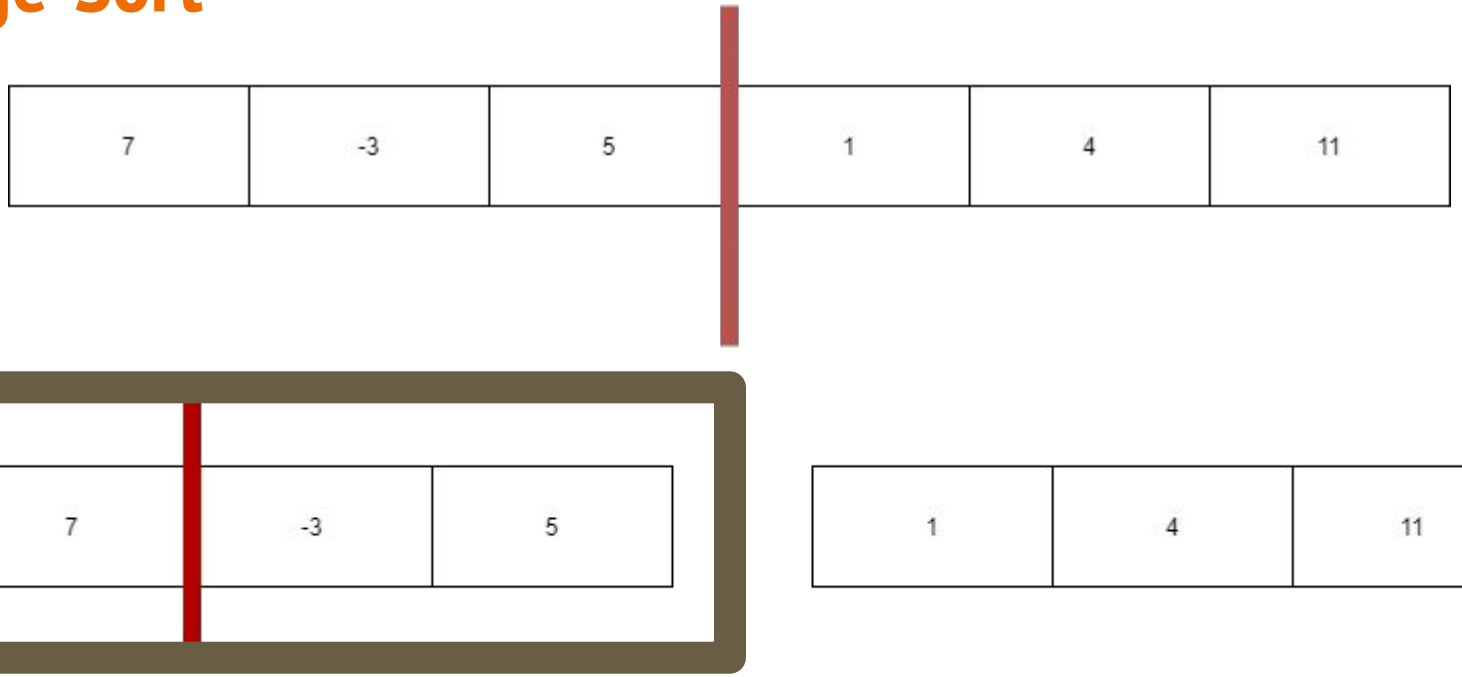
# Merge-Sort



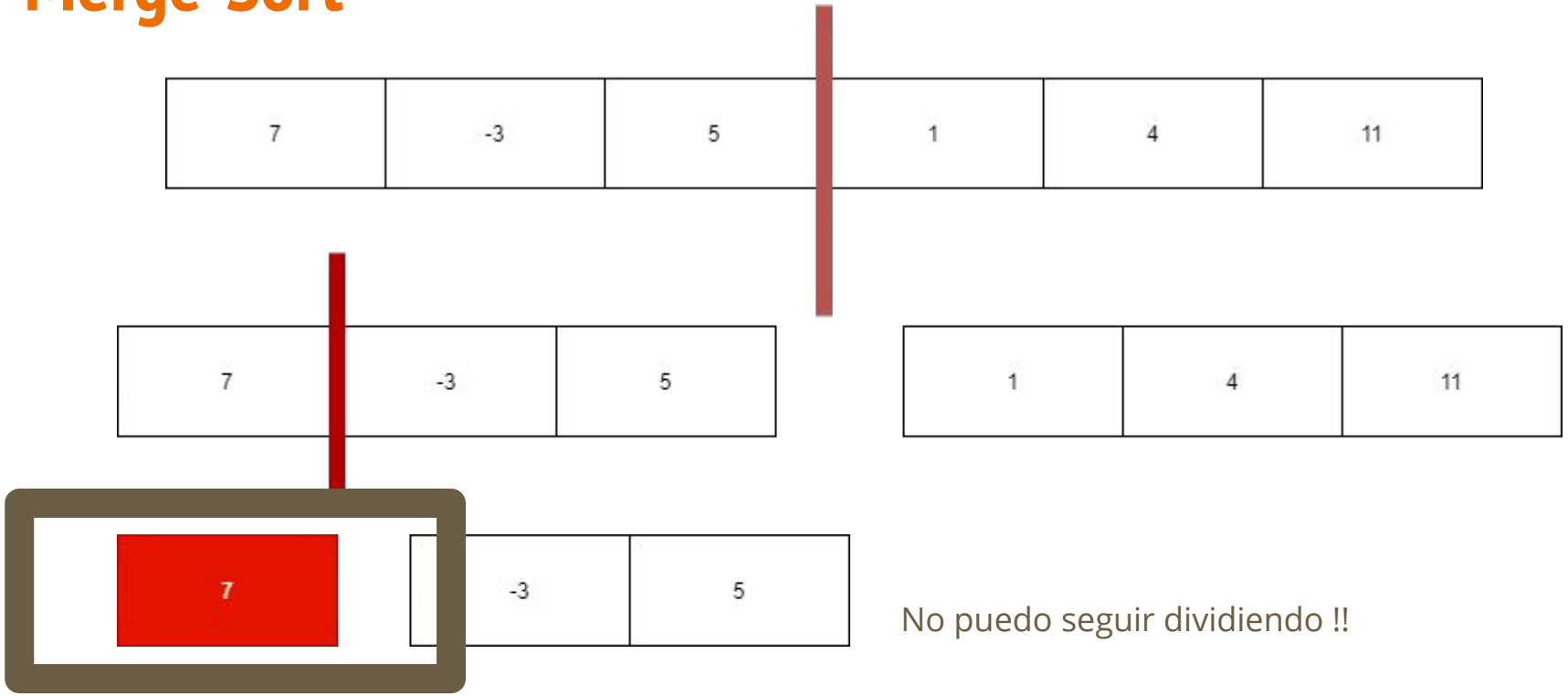
# Merge-Sort



# Merge-Sort

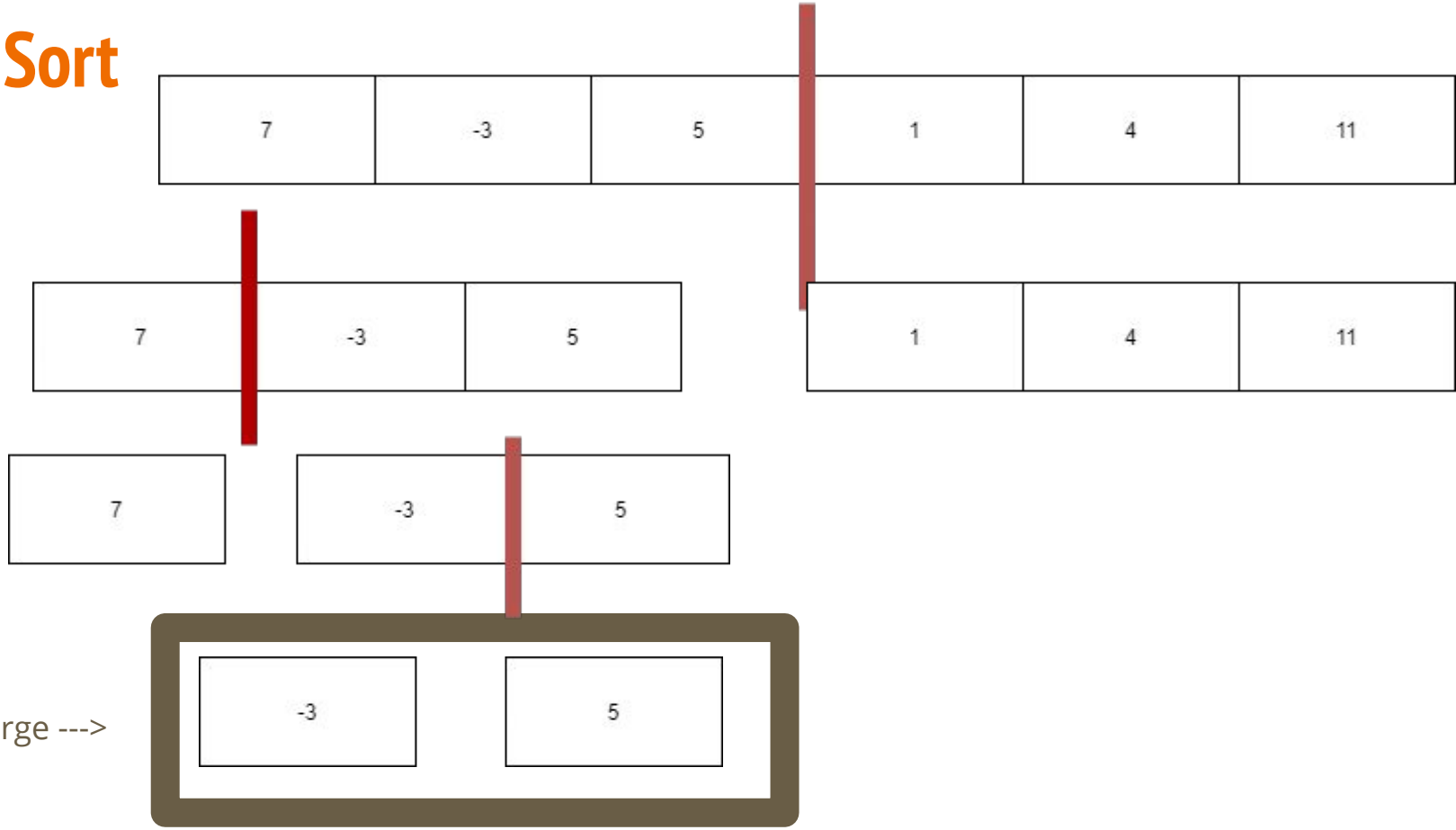


# Merge-Sort

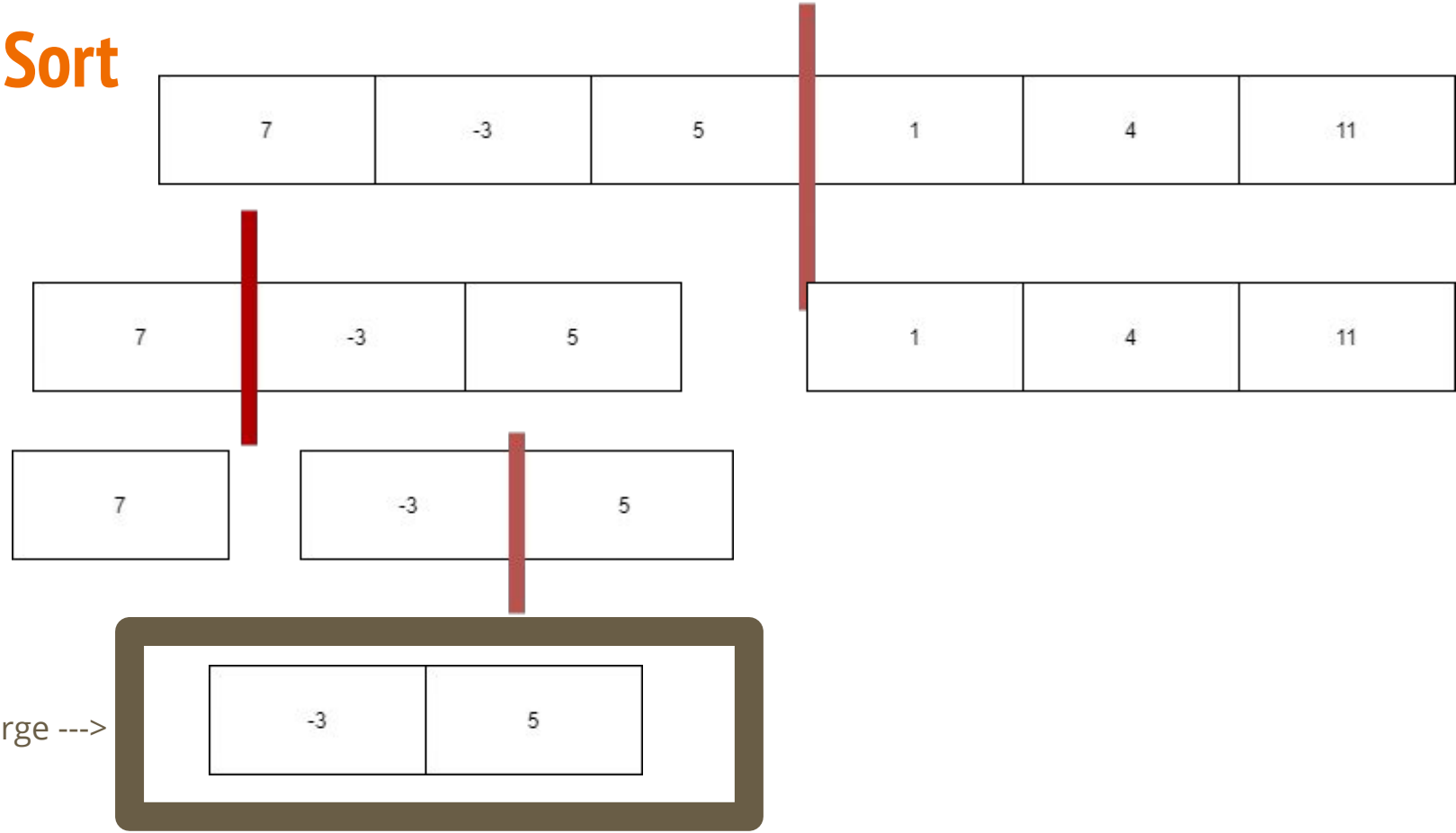



No puedo seguir dividiendo !!

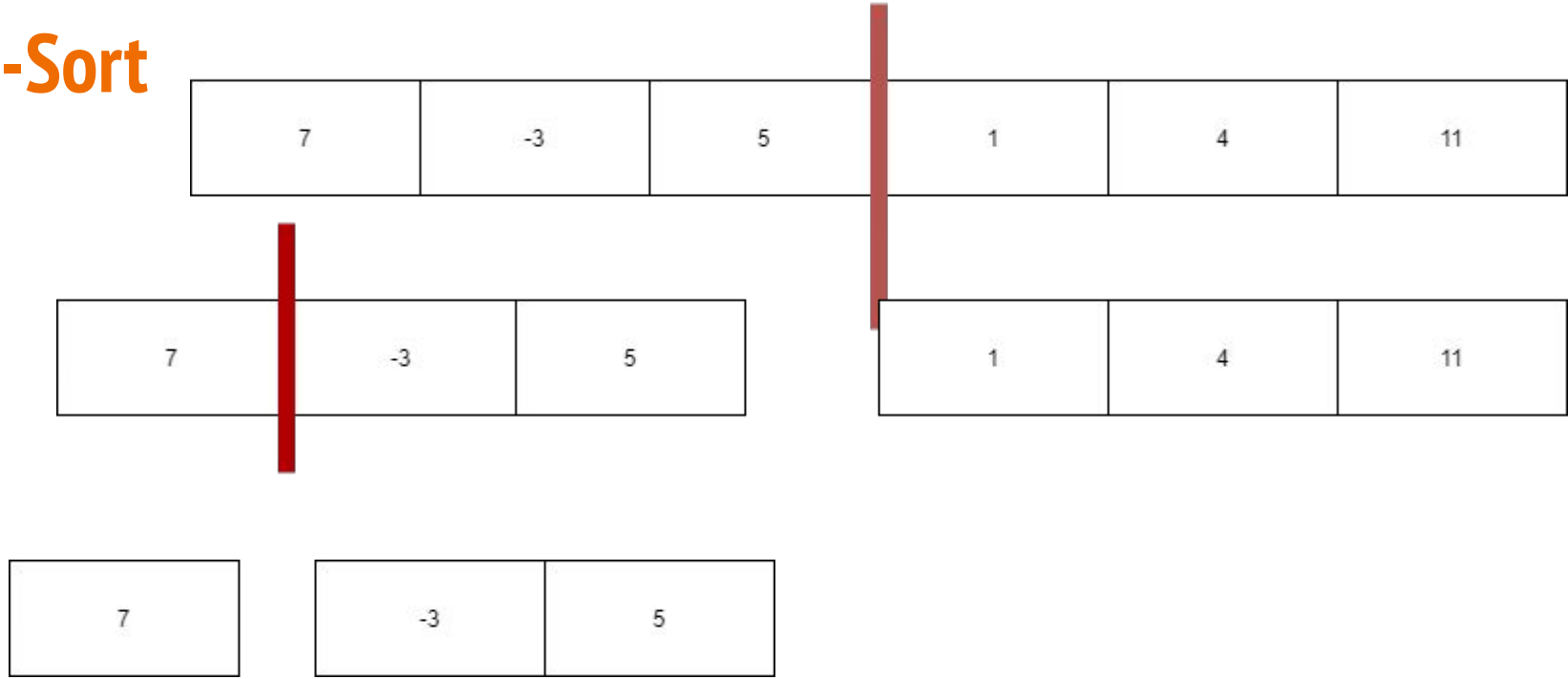
# Merge-Sort



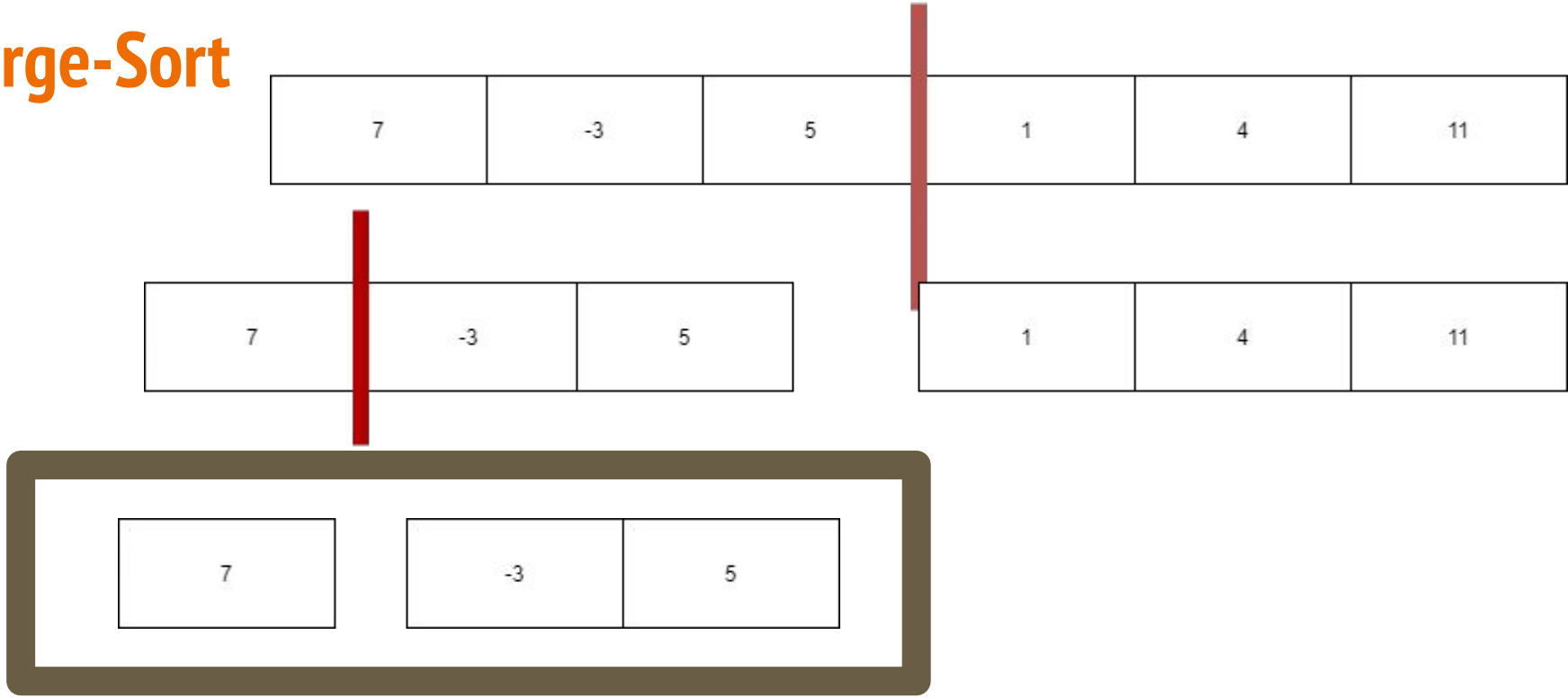
# Merge-Sort



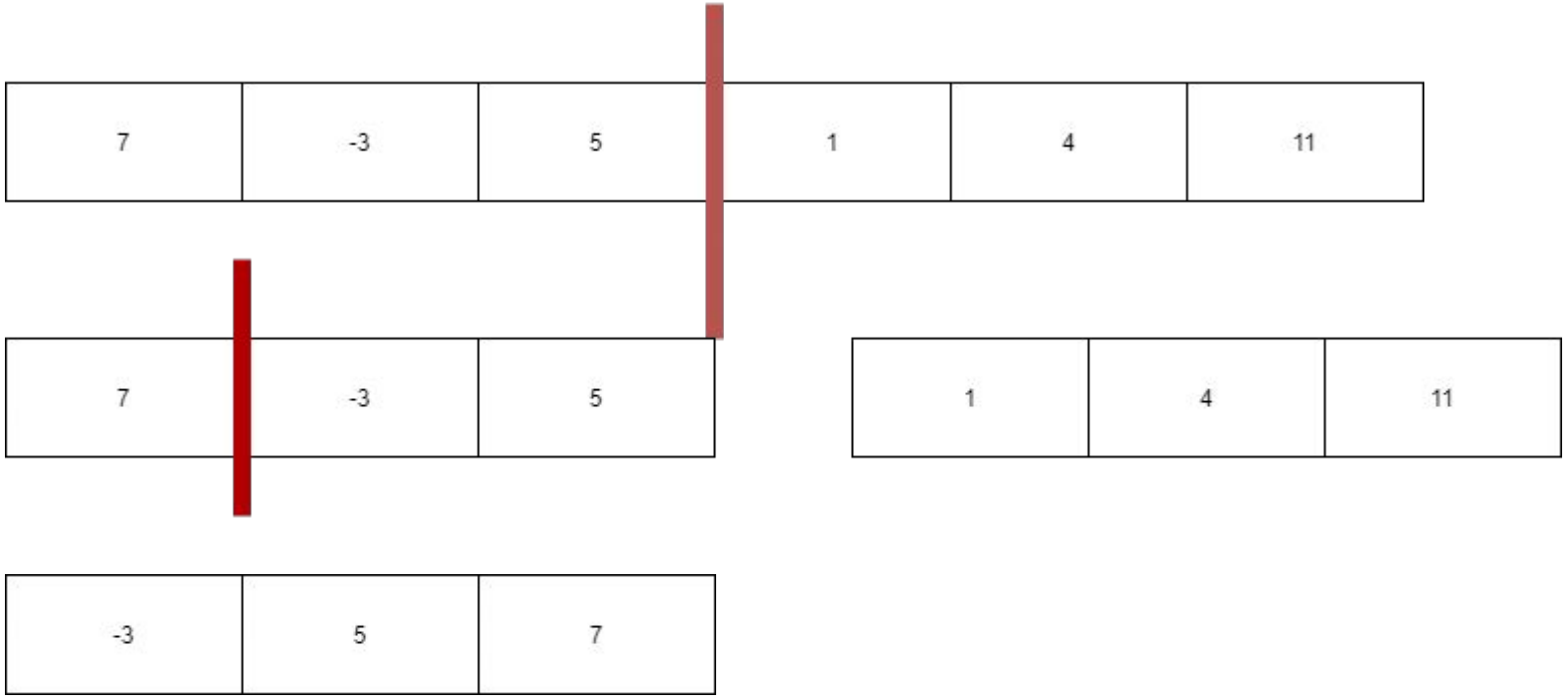
# Merge-Sort



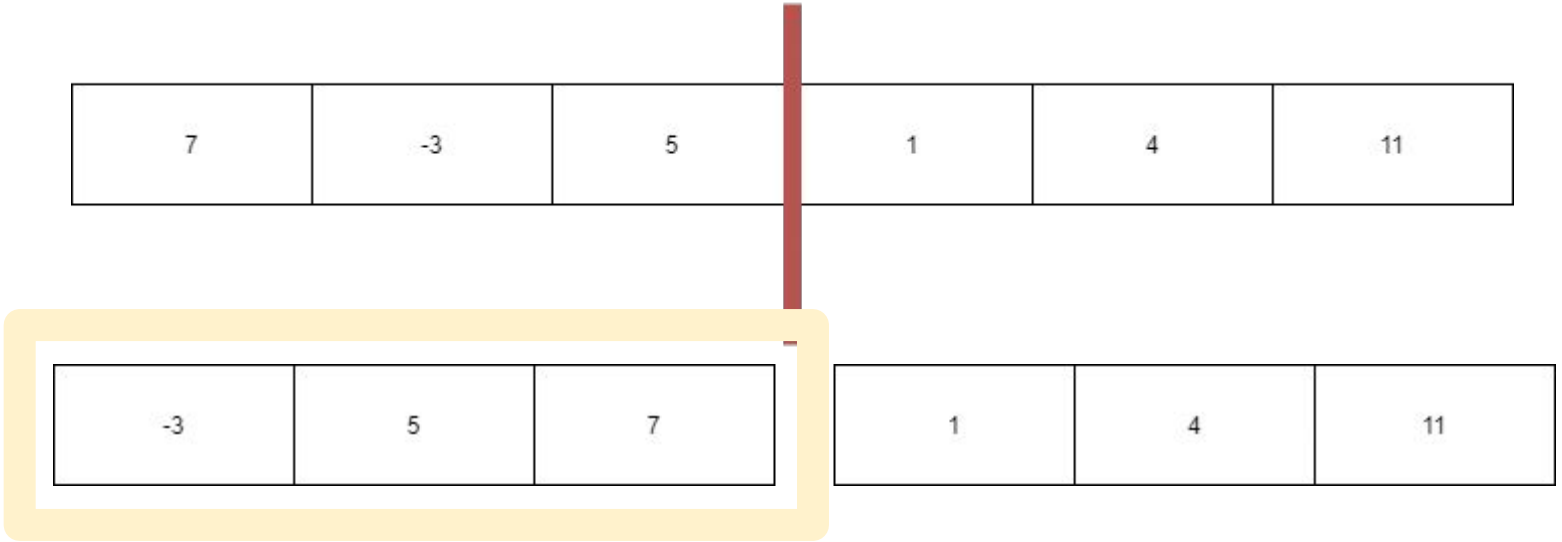
# Merge-Sort



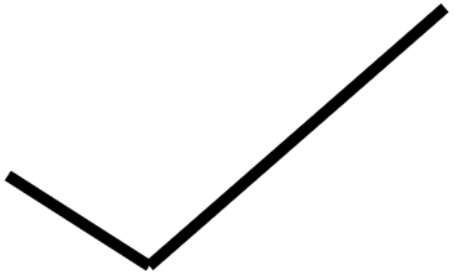
# Merge-Sort



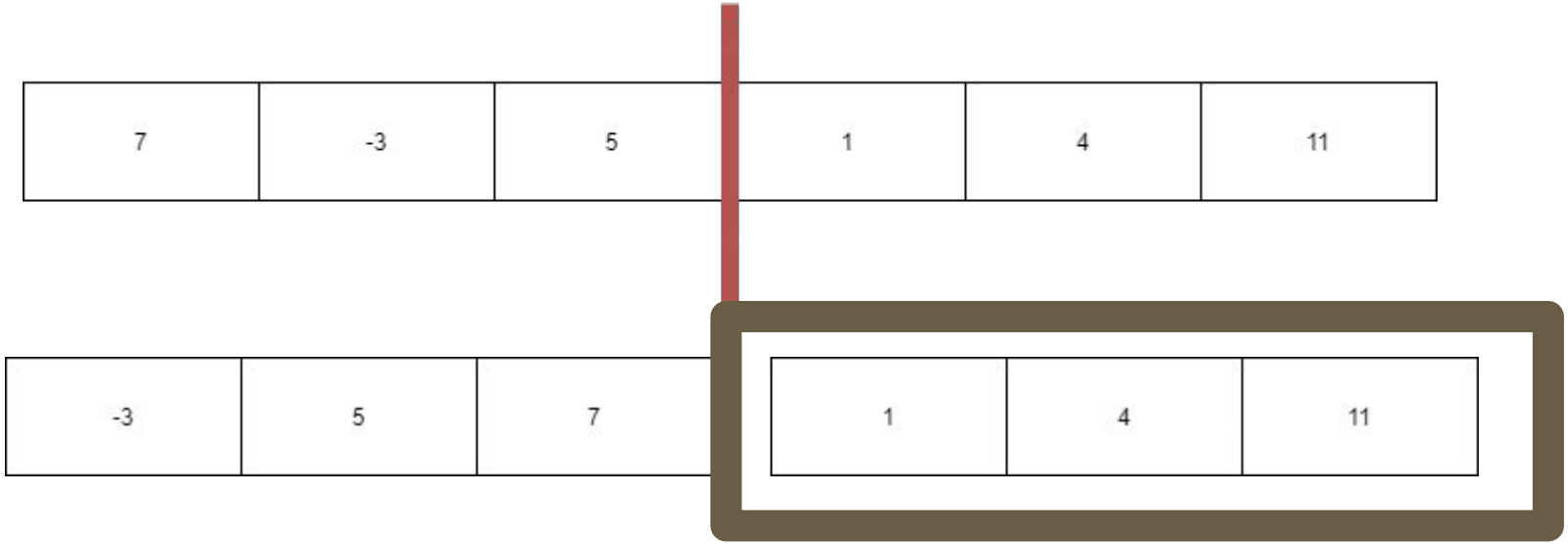
# Merge-Sort




- Quedamos listos con el lado Izquierdo

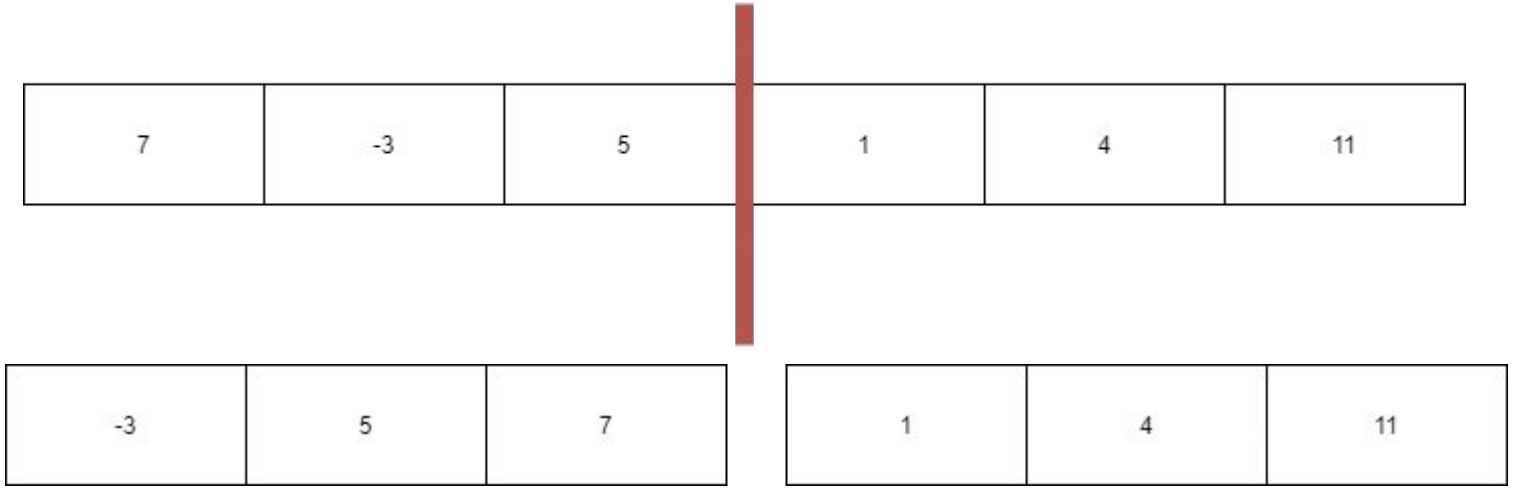


# Merge-Sort



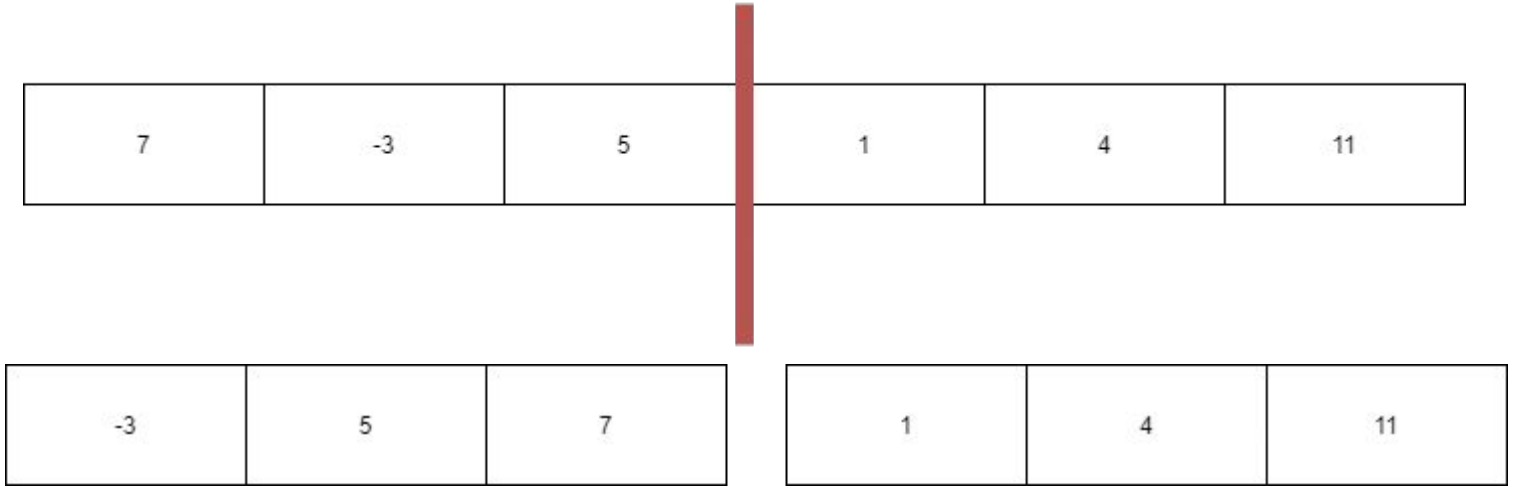

- Aplicamos Merge al lado derecho

# Merge-Sort



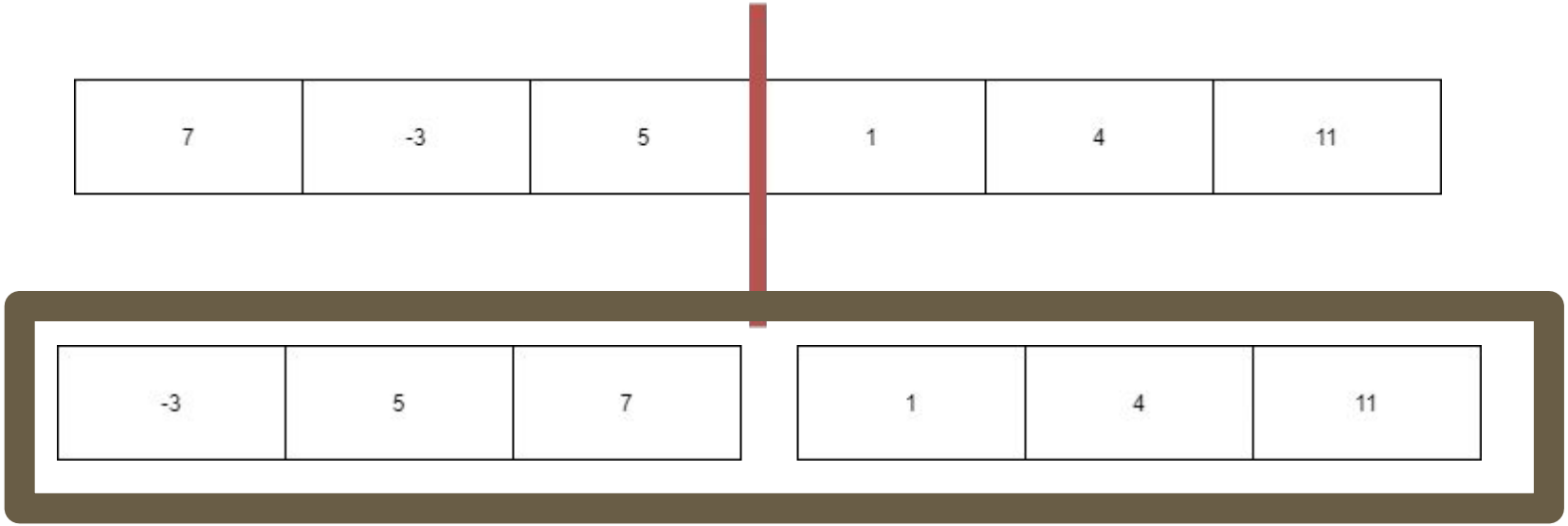

- Hacemos FF

# Merge-Sort



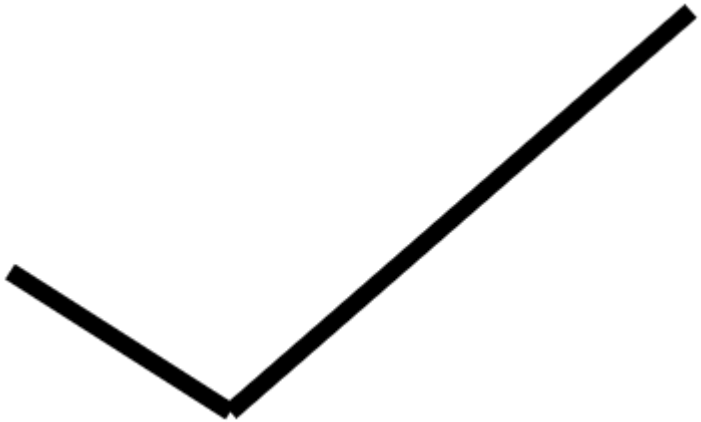

- Está Ordenado !

# Merge-Sort



# Merge-Sort

| -3 | 1 | 4 | 5 | 7 | 11 |
|---|---|---|---|---|---|



# Implementación C

```c
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    /* Copy the remaining elements of L[], if there
    are any */
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    /* Copy the remaining elements of R[], if there
    are any */
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}
```

# Llamado recursivo

```
void mergeSort(int arr[], int l, int r)
{
    if (l < r) {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l + (r - l) / 2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}

/* UTILITY FUNCTIONS */
```

# QuickSort (breve repaso)

- Usar el algoritmo **partition** para dividir
- Ordenar cada partición de forma recursiva

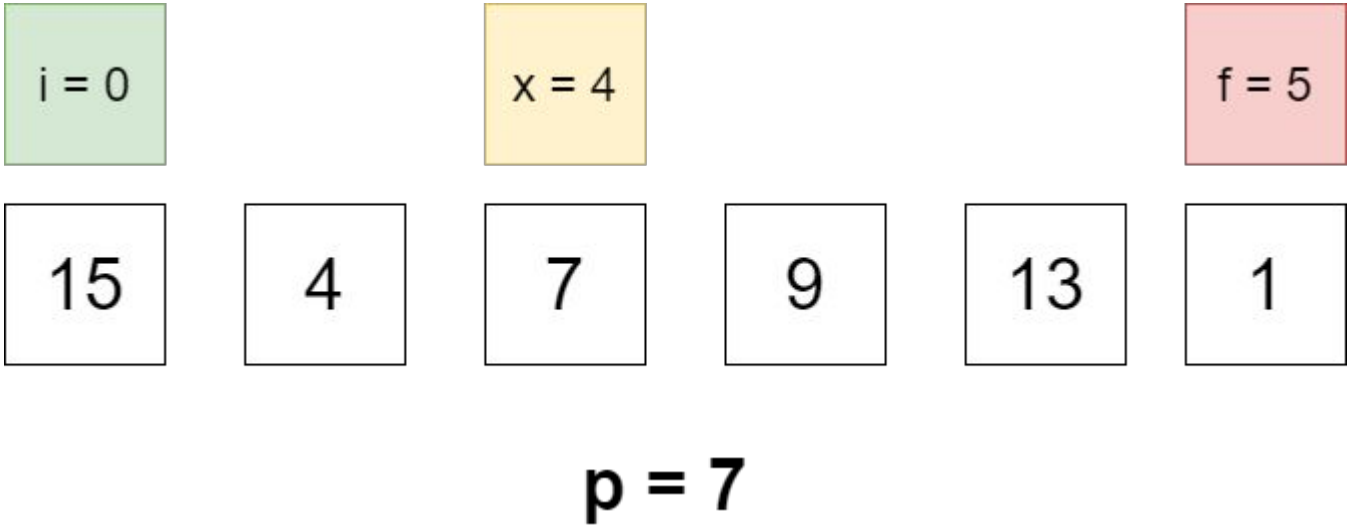
# Partition

Para entender QuickSort, primero hay que entender partition

$$
\begin{aligned}
&partition(A,\ i,\ f):\\
&\quad x \leftarrow un\ indice\ aleatorio\ en\ [i,\ f], \quad p \leftarrow A[x]\\
&\quad A[x] \rightleftarrows A[f]\\
&\quad j \leftarrow i\\
&\quad for\ k \in [i,\ f-1]:\\
&\quad\quad if\ A[k] < p:\\
&\quad\quad\quad A[j] \rightleftarrows A[k]\\
&\quad\quad\quad j \leftarrow j+1\\
&\quad A[j] \rightleftarrows A[f]\\
&\quad return\ j
\end{aligned}
$$

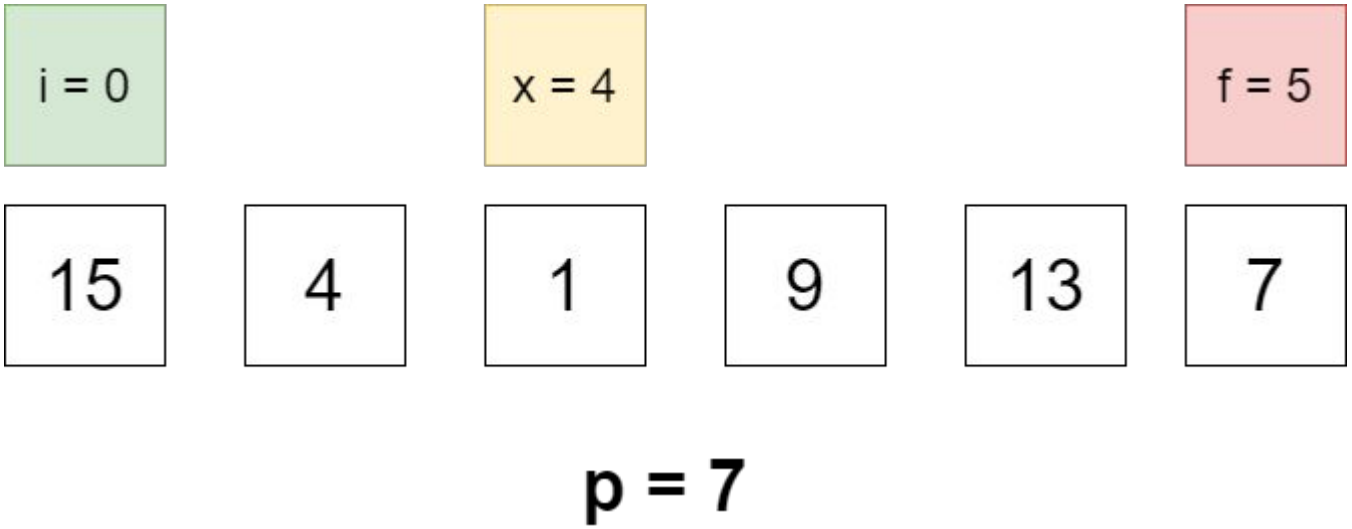
# Partition (ejemplo)

**Partition(A, 0, 5)**

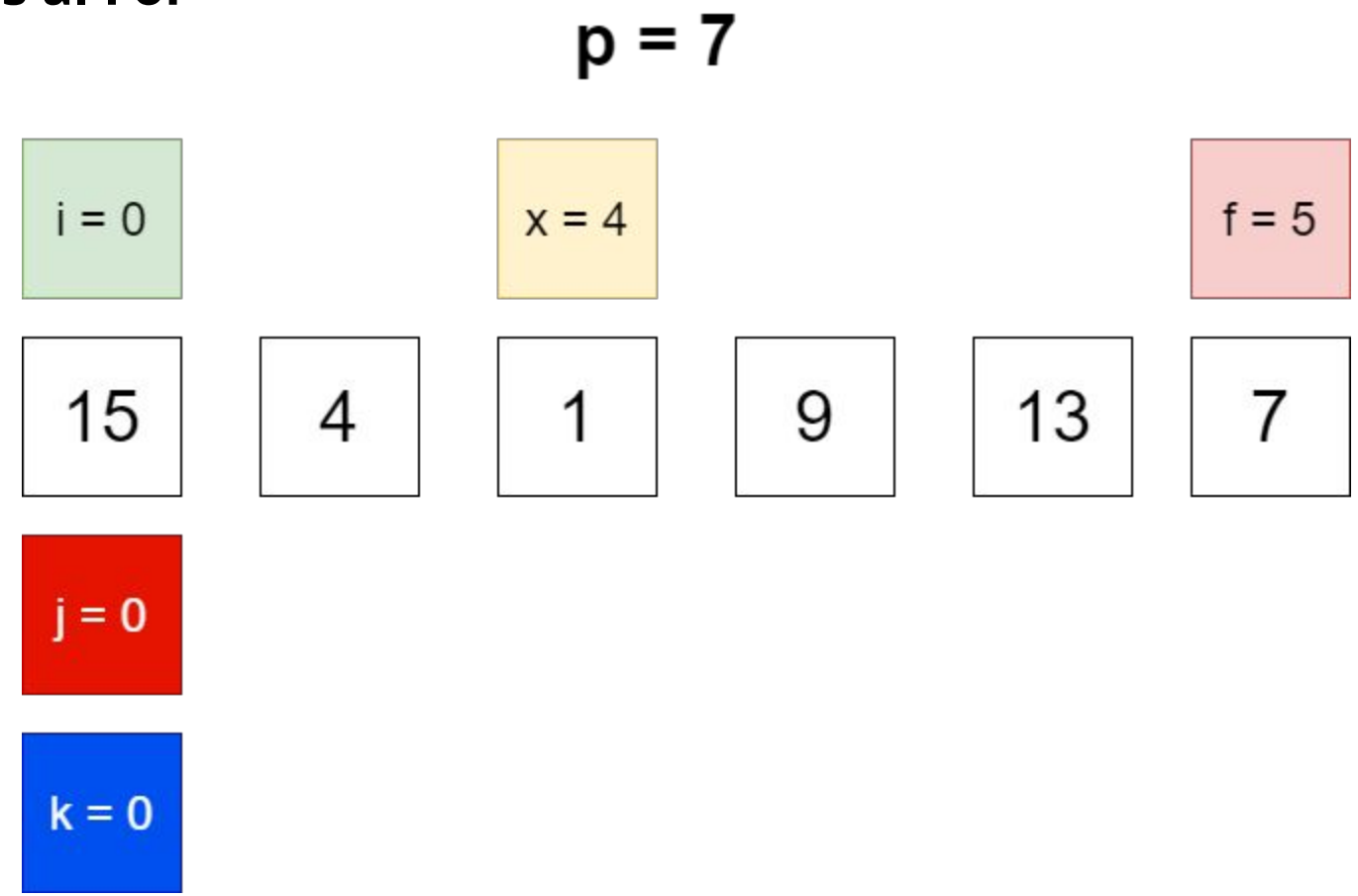


# Partition (ejemplo)

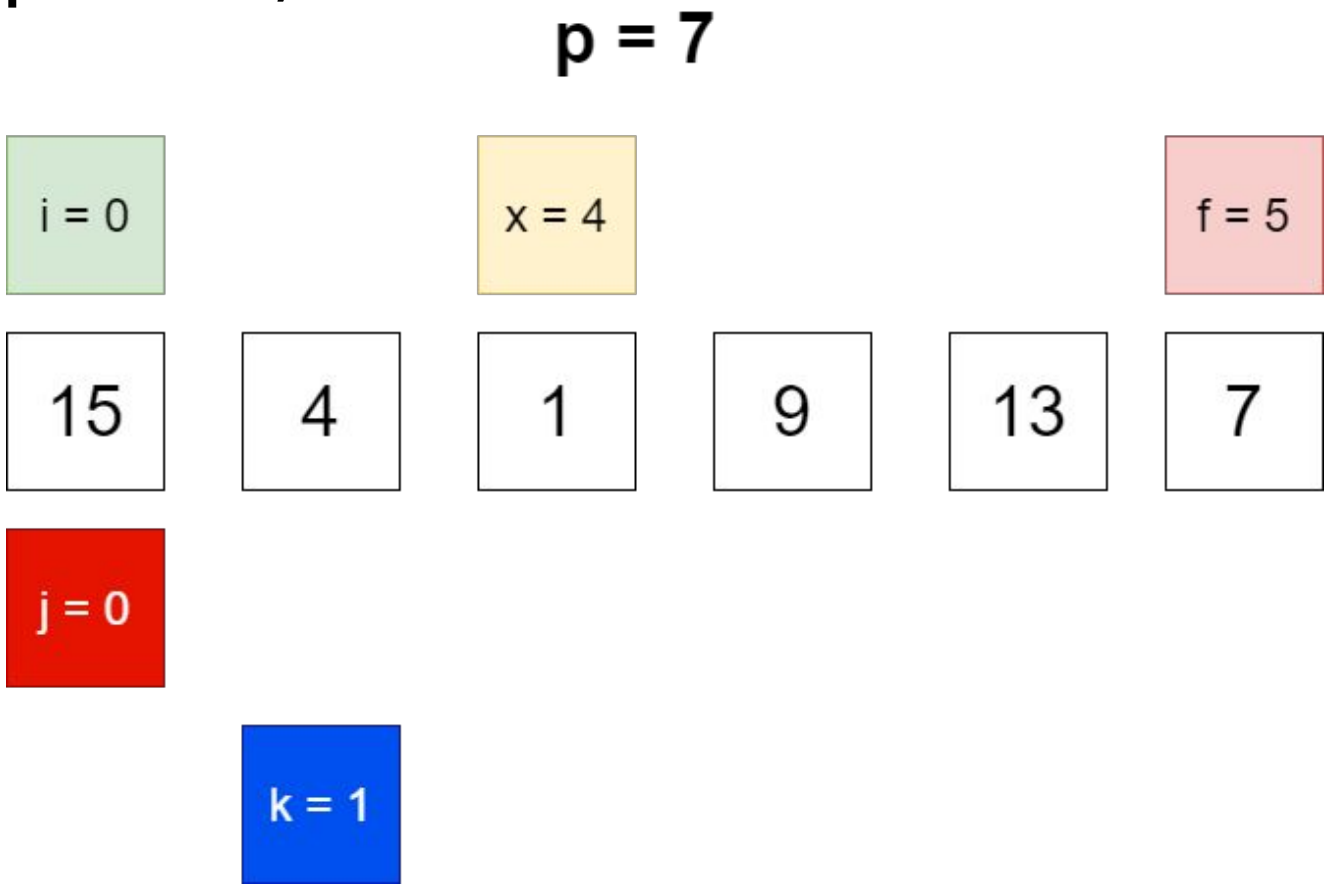
**Cambiamos A[ x ] por A[ f ]**



# Entramos al For

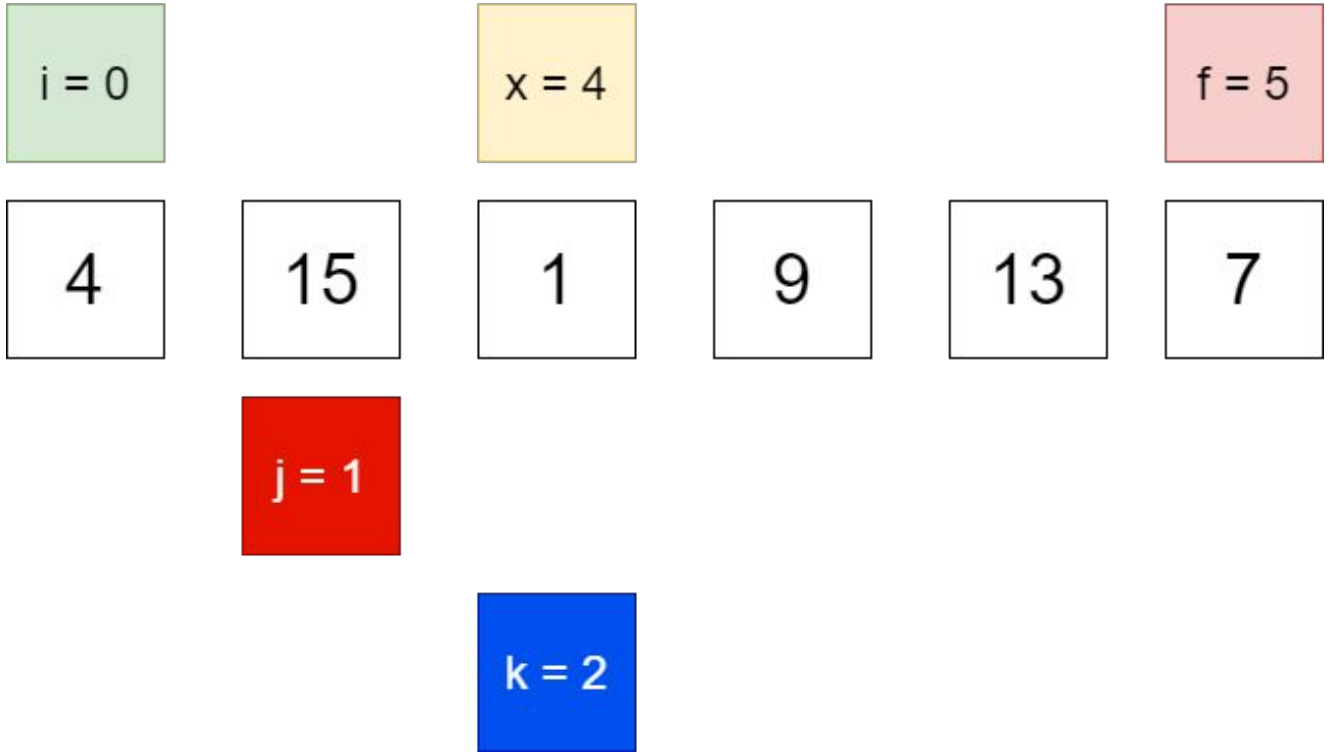


**15 > 7, no pasa nada, avanzamos**

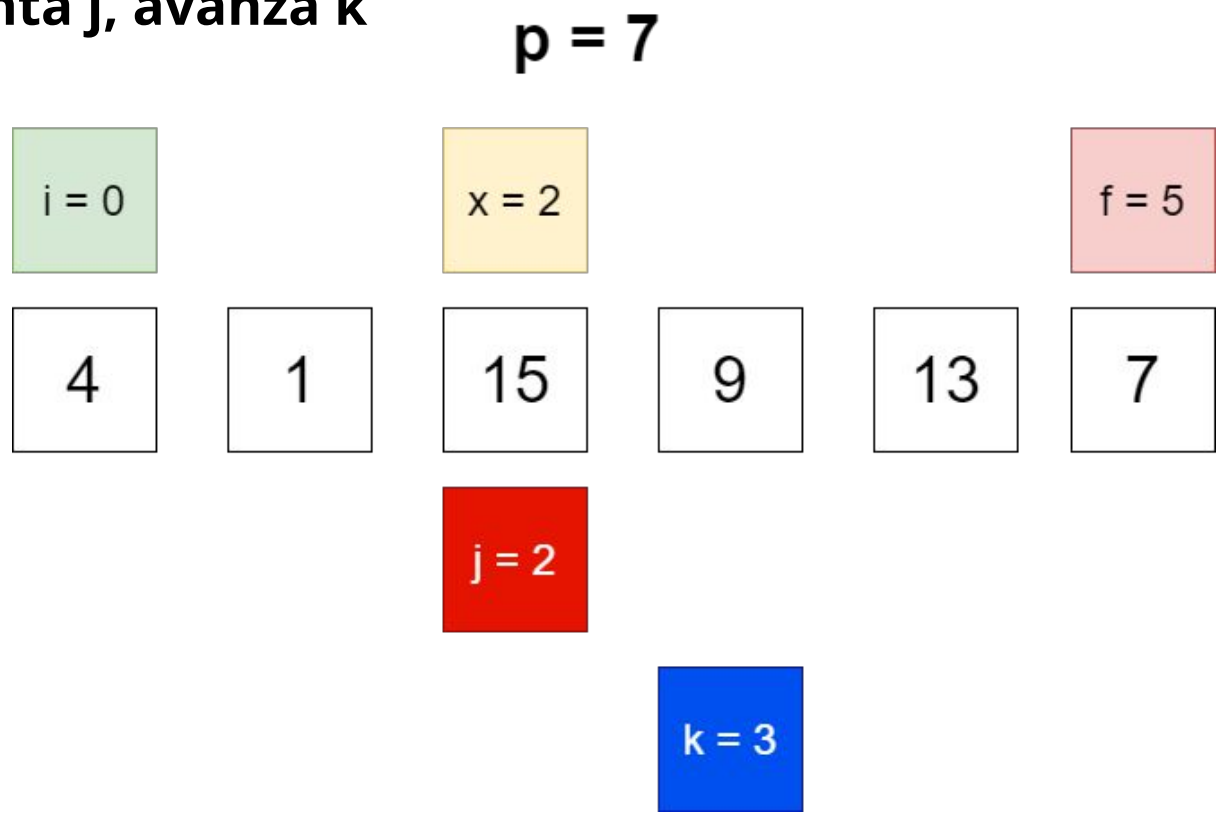


**4 < 7, se cambian 4 con 15 y se incrementa j, avanza k**

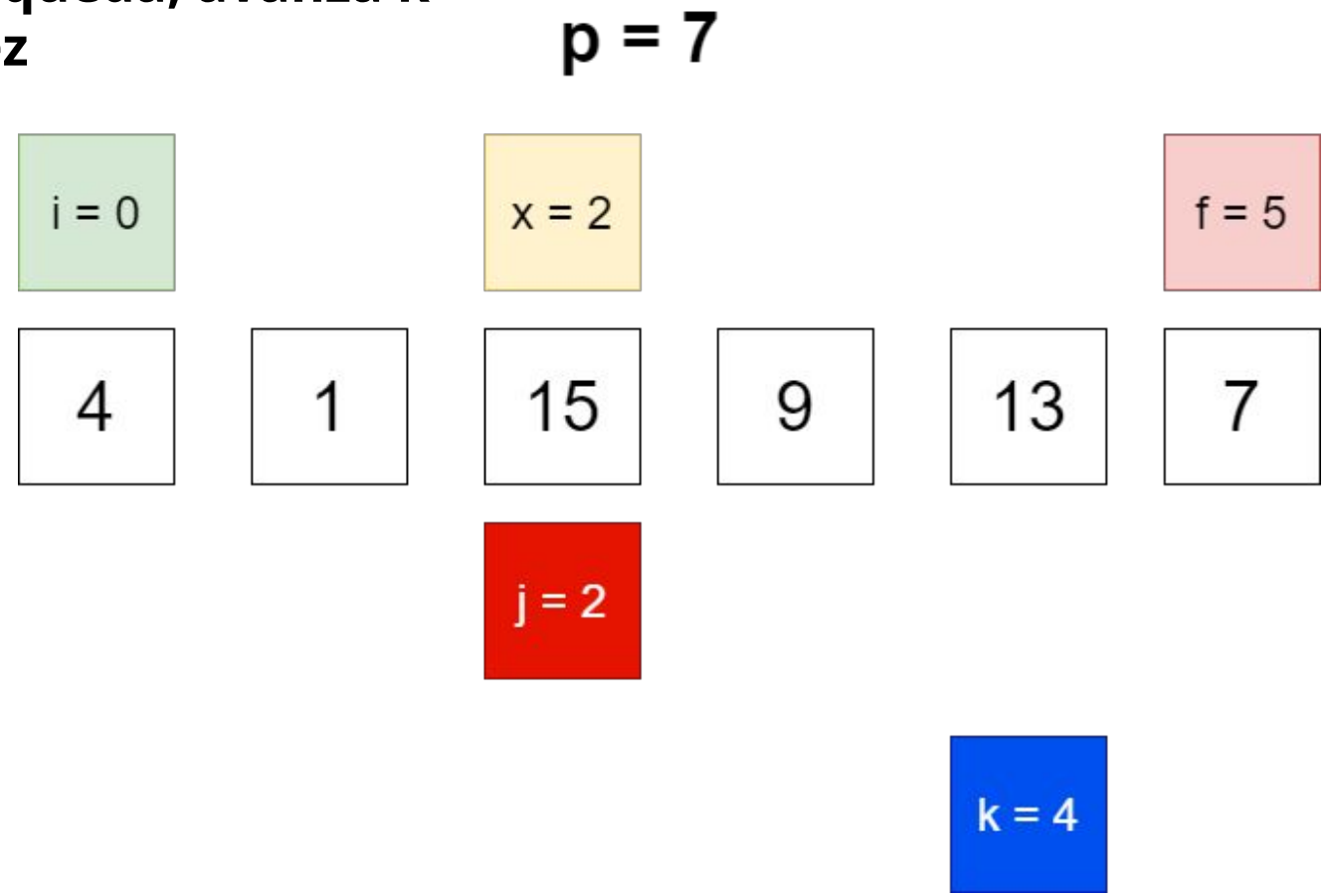
**p = 7**



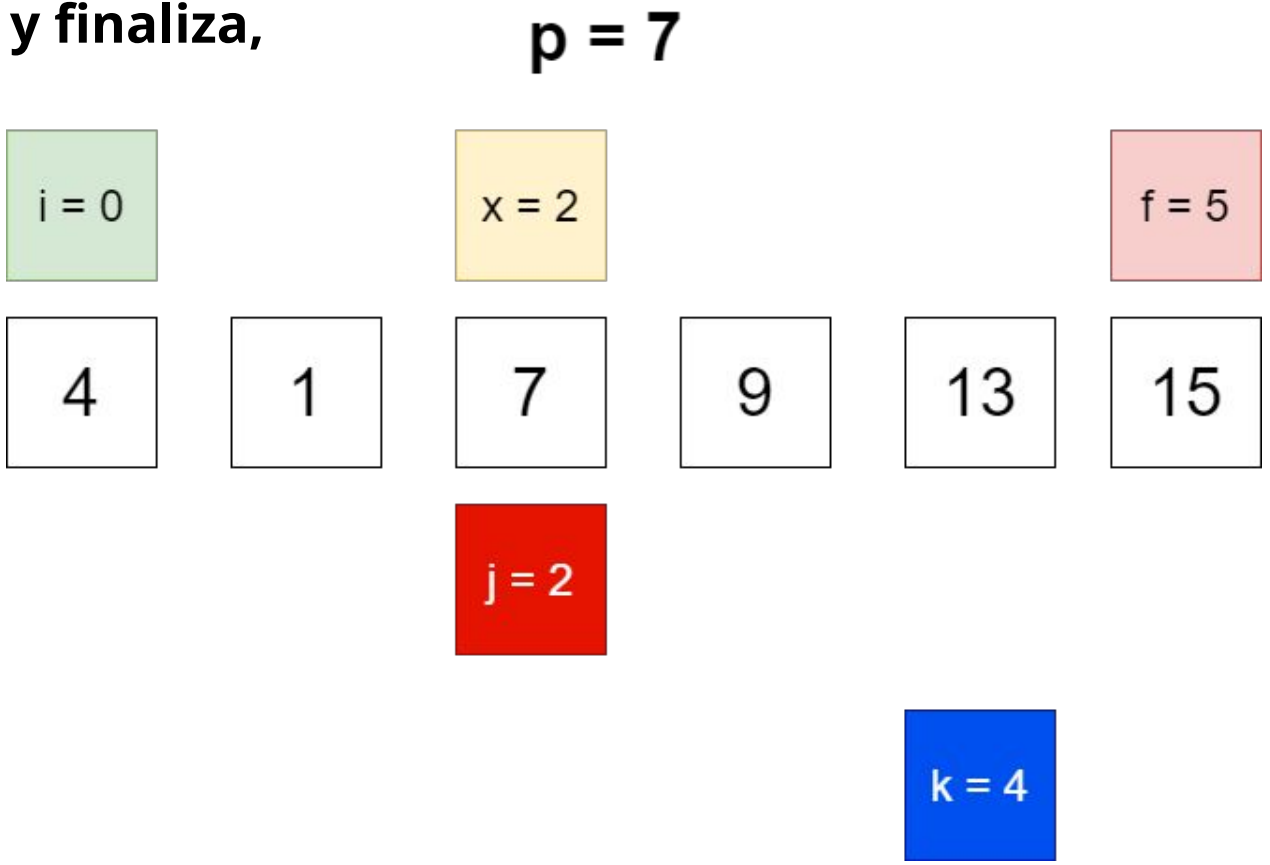
**1 < 7, se cambian 1 con 15 y se incrementa j, avanza k**



**9 > 7, j se queda, avanza k ultima vez**



**Terminó el loop, se cambian A[j] con A[f] y finaliza, retorna j**



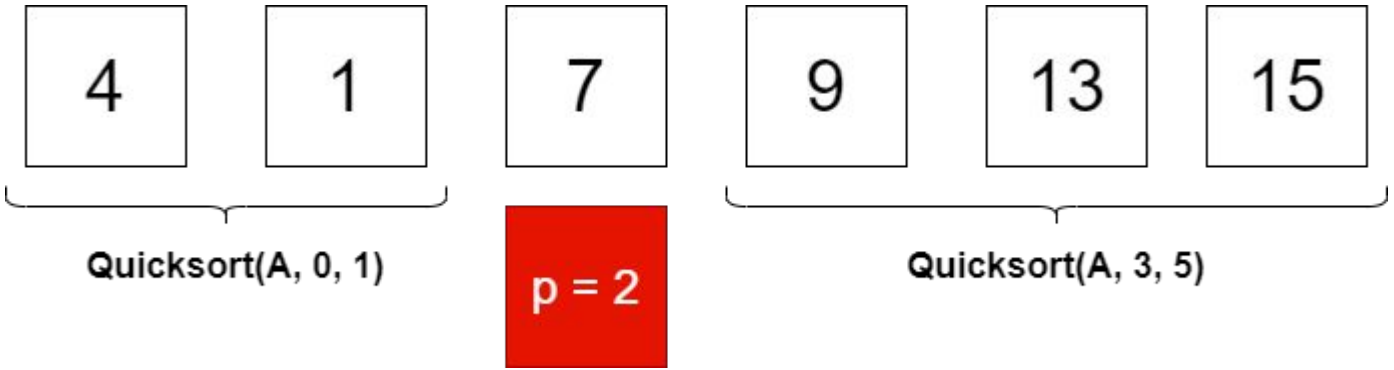
# QuickSort

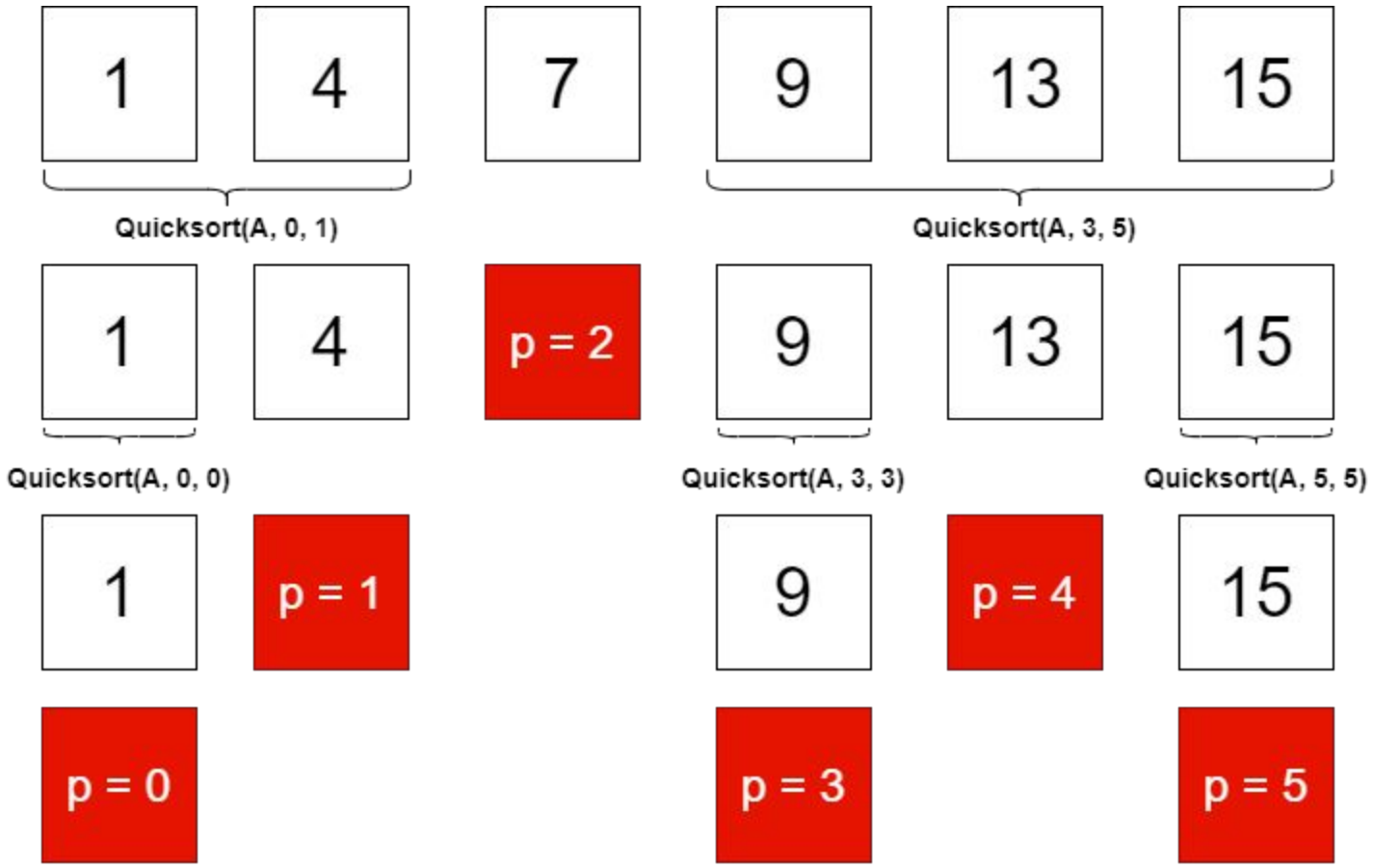
Ahora que tenemos claro partition, vemos quicksort

$$
\begin{aligned}
&quicksort(A, i, f):\\
&\quad if\ i \leq f:\\
&\qquad p \leftarrow partition(A,\ i,\ f)\\
&\qquad quicksort(A,\ i,\ p-1)\\
&\qquad quicksort(A,\ p+1,\ f)
\end{aligned}
$$

# QuickSort (ejemplo)

Seguimos con el ejemplo de partition




1
4
7
9
13
15
Quicksort(A, 0, 1)
Quicksort(A, 3, 5)
1
4
p = 2
9
13
15
Quicksort(A, 0, 0)
Quicksort(A, 3, 3)
Quicksort(A, 5, 5)
1
p = 1
9
p = 4
15
p = 0
p = 3
p = 5

# QuickSort (ejemplo)

Seguimos con el ejemplo de partition