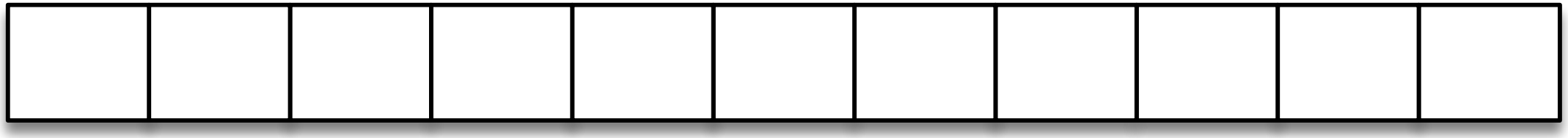
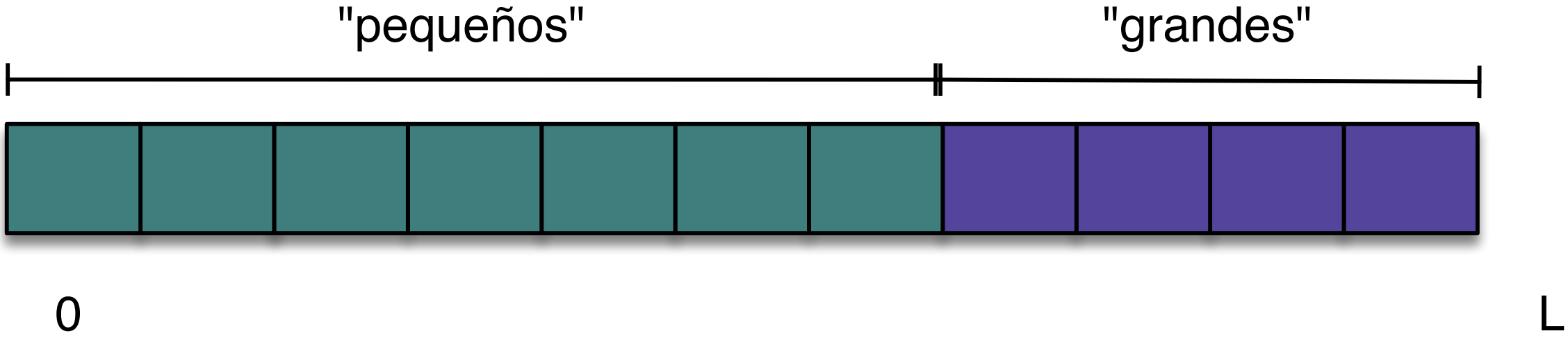
QuickSort

- Després de corregir els exàmens encara hi ha lo pitjor: ordenar-los
- Com ordenar a mà molt exàmens és complex, una posible idea és primer separar-los en vàries franges:
  - A-E, F-L, M-P, Q-Z
  - Ordenar cada franja per separat
  - Després només cal posar cada franja, una darrera de l'altre, en el seu ordre
    - I segur que al fer-ho així, ja queda tot ordenar, per què?
- Fixeu-vos que això ens dóna una estrategia "recursiva" per ordenar
  - I aquesta, més o menys, és la idea darrera de l'algoritme del QuickSort

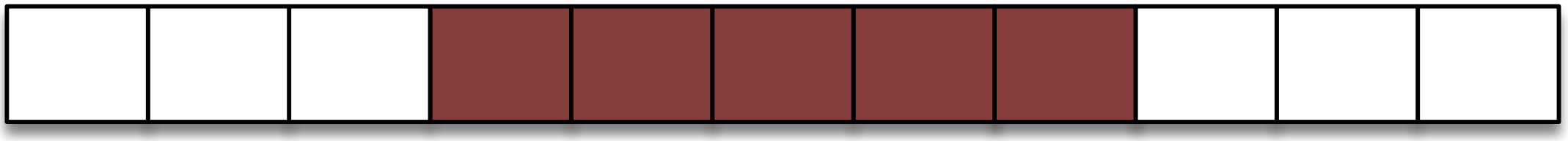


0 L

- El punt de partida será l'array que volem ordenar
- Durant l'exposició suposarem que és un array d'enters
- Òbviament l'algoritme d'ordenació la única cosa que pot modificar dels elements de l'array són les seves posicions
  - No pot esborrar elements
  - No pot afegir elements
  - Per garantir això només farem intercanvis (swaps) entre els elements de dues posicios

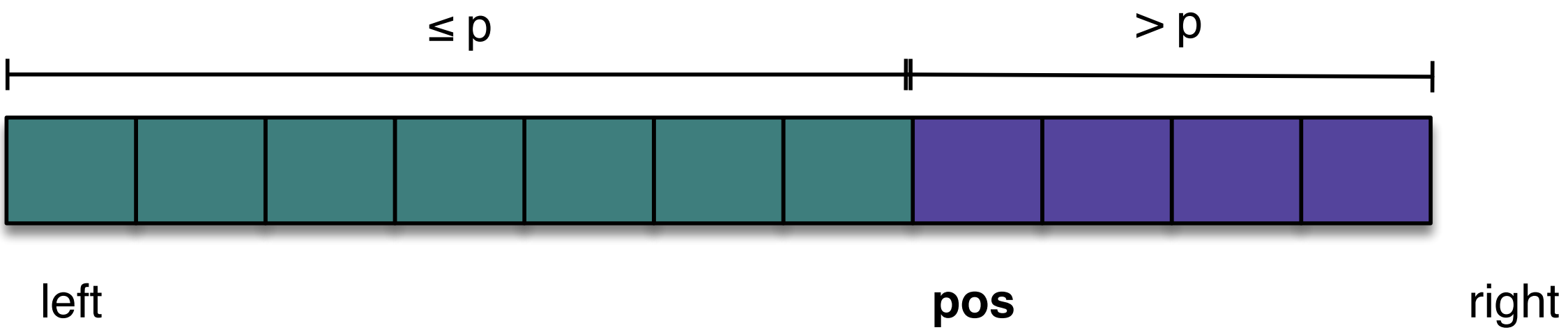


- Com hem dit abans la idea de l'algoritme és separar en troços de manera que els elements d'un tros no "interfereixin" amb els de l'altre
  - Cap dels elements petits és més gran que un dels grans; cap dels elements grans és més petit que un dels petits
- Amb les lletres dels cognoms, amb certa experiencia, podem més o menys preveure les franges a usar; en el cas general, no
  - El que és petit en un array pot ser gran en un altre

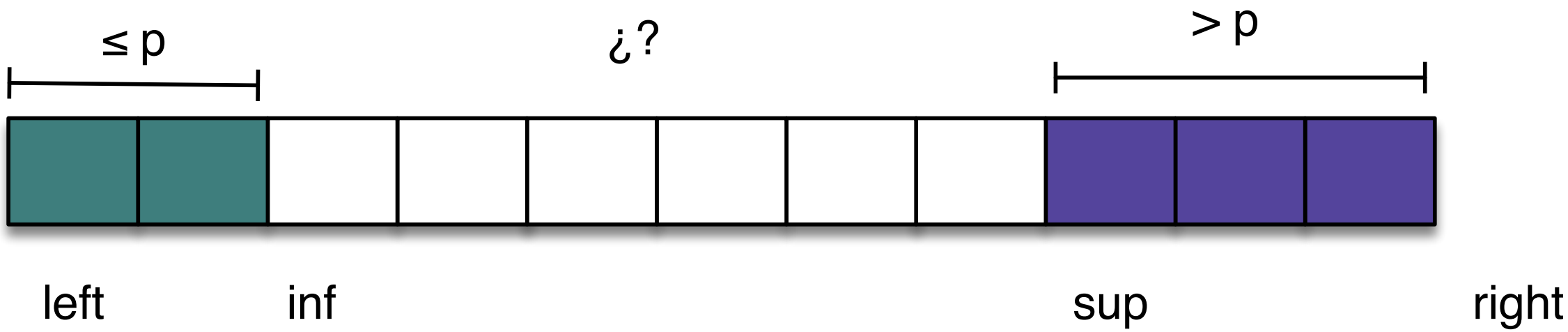


- Per poder fer recursivitat usarem els límits **left** i **right** per demarcar la zona de l'array sobre la que treballem
- A més, com es pot veure a la imatge, per tal de fer la crida inicial (per ordernar tot l'array)
  - left = 0
  - right = L
- Això ens permet fer ja un primer esquema de l'algoritme
  - Els casos simples serán quan podem garantir que el subarray està ordenat, és a dir, quan la mida del subarray sigui 0 o 1.

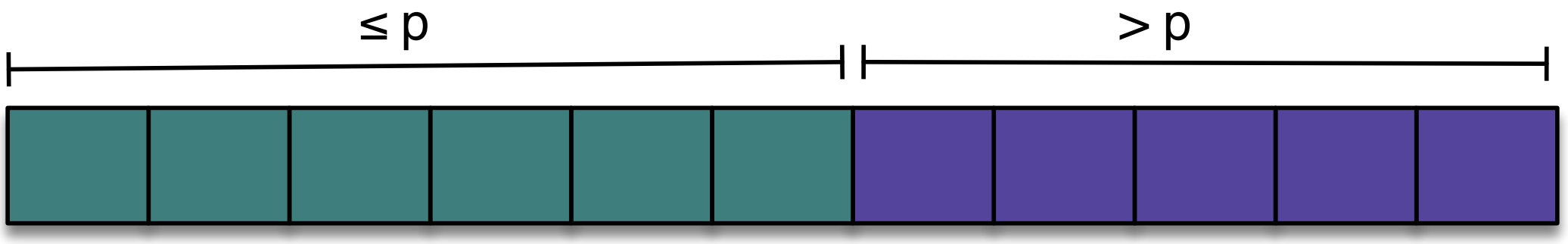
```
public void quickSort(int[] v) {
    quickSort(v, 0, v.length);
}

public void quickSort(int[] v, int left, int right) {
    // 0 <= left <= right <= v.length

    if (right - left > 1) {

        // Particionar el vector y devolver la posición
        // de corte.

        ¿?

        quickSort(v, left, ¿?);  // “pequeños”
        quickSort(v, ¿?, right); // “grandes”
  }
}
```



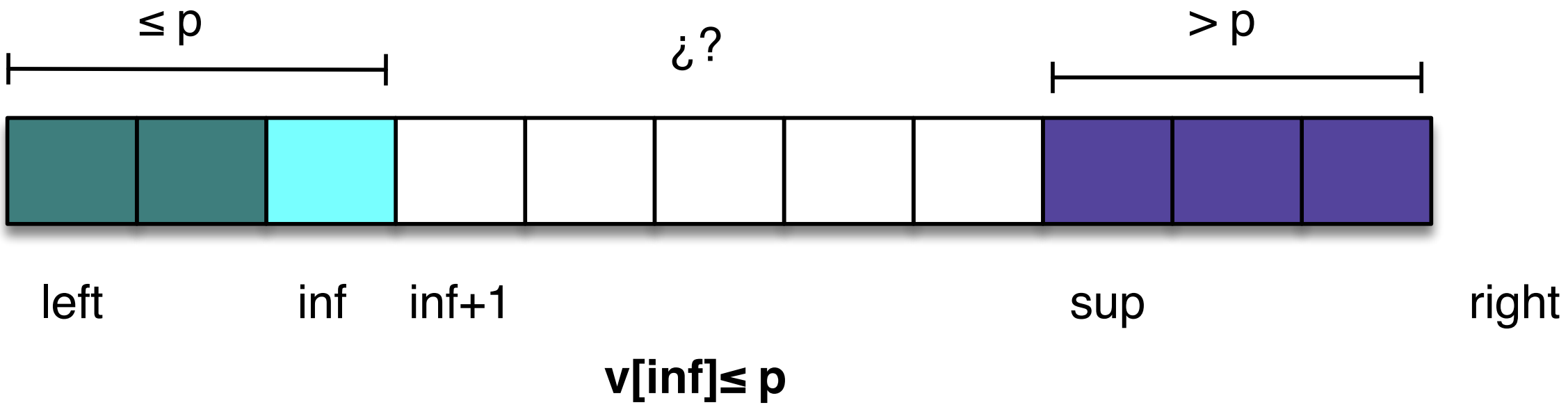
- La idea per definit “petits” i “grans” de manera que tingui alguna cosa a veure amb el contingut real de l’array és agafar un valor d’entre els elements de l’array, que anomenarem pívot ($\boldsymbol{p}$), i buscar la posició (**pos**) que deixa els elements $\leq \boldsymbol{p}$ al prefix (esquerra) i els elements $> \boldsymbol{p}$ al sufix (dreta)
  - **Això ens dóna una definició objectiva de ”petits” i “grans”**

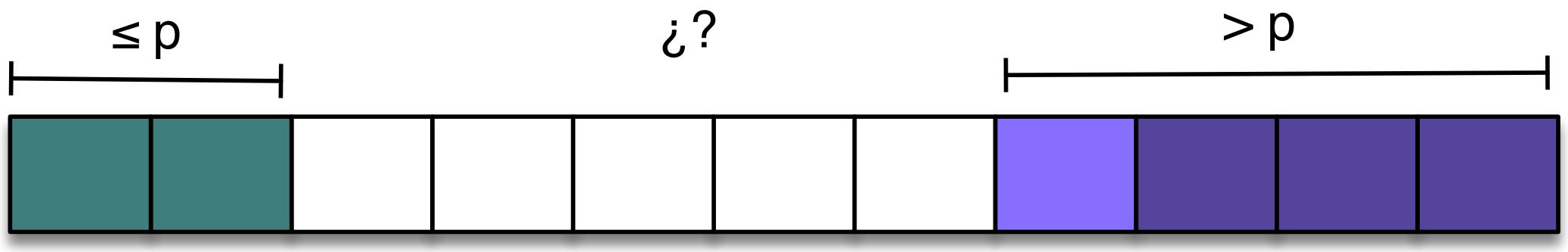


- (Normalment el mètode de partició es fa iterativament, però com estem al tema de recursivitat, també ho farem de forma recursiva)
- El que tindrem és que dividirem l'array en tres zones:
  - **[left, inf)** -> elements que sabem que són $\leq \boldsymbol{p}$
  - **[inf, sup)** -> elements que no saben què són **¿?**
  - **[sup, right)** -> elements que sabem que són $> \boldsymbol{p}$

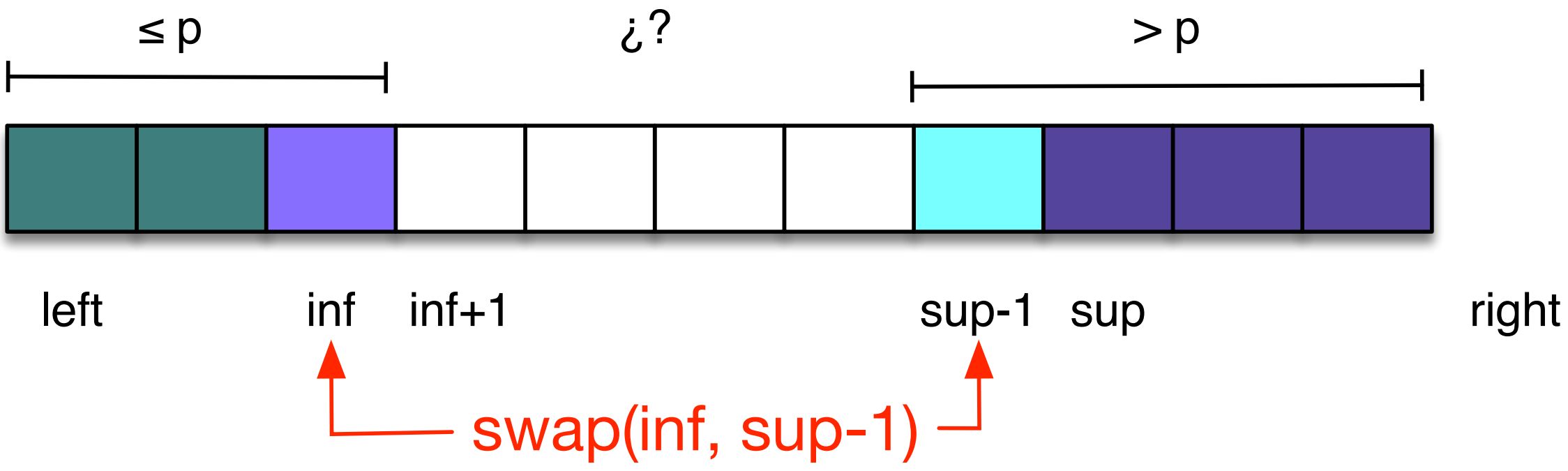


- Si la zona intermèdia (¿?) és buida ja sé quina posició separa els petits ($\leq \boldsymbol{p}$) dels grans ($> \boldsymbol{p}$)
  - **Això defineix el cas simple !!!**
- Ara queda veure què fer quan la zona intermèdia no és buida


≤ p
¿?
> p
left
inf
inf+1
sup
right
v[inf]≤ p



v[sup-1]> p



v[inf] >p && v[sup-1]≤ p

```
public int partition(int[] v, int pivot, int inf, int sup) {
    // 0 <= left <= inf <= sup <= right <= v.length
    if ( inf == sup ) {
        return inf;
    } else if (v[inf] <= pivot) {
        return partition(v, pivot, inf + 1, sup);
    } else if (v[sup - 1] > pivot) {
        return partition(v, pivot, inf, sup - 1);
    } else {
        swap(v, inf, sup - 1);
        return partition(v, pivot, inf + 1, sup - 1);
    }
}
```

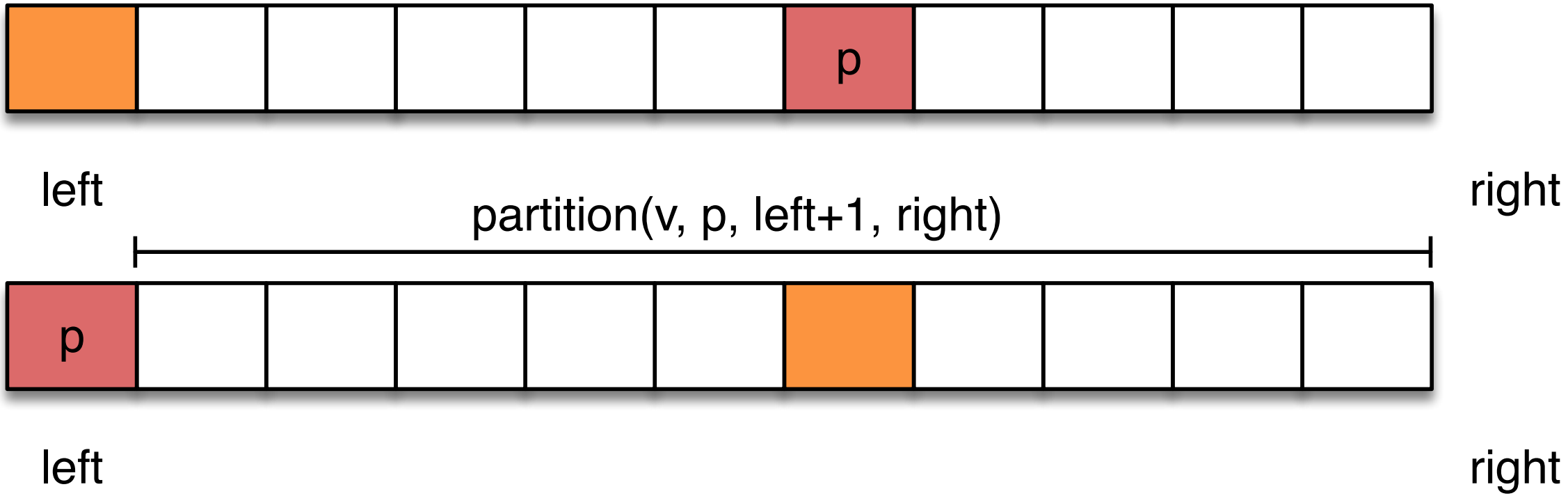
```
public void quickSort(int[] v, int left, int right) {
    // 0 <= left <= right <= v.length
    if (right - left > 1) {
        int pivotValue = choosePivot(v, left, right);
        int pos = partition(v, pivotValue, left, right);
        quickSort(v, left, pos);
        quickSort(v, pos, right);
    }
}

public int choosePivot(int[] v, int left, int right) {
    // Returns any element of v whose position is
    // >= left and < right
    ¿?
}
```

Però tenim un problema ...

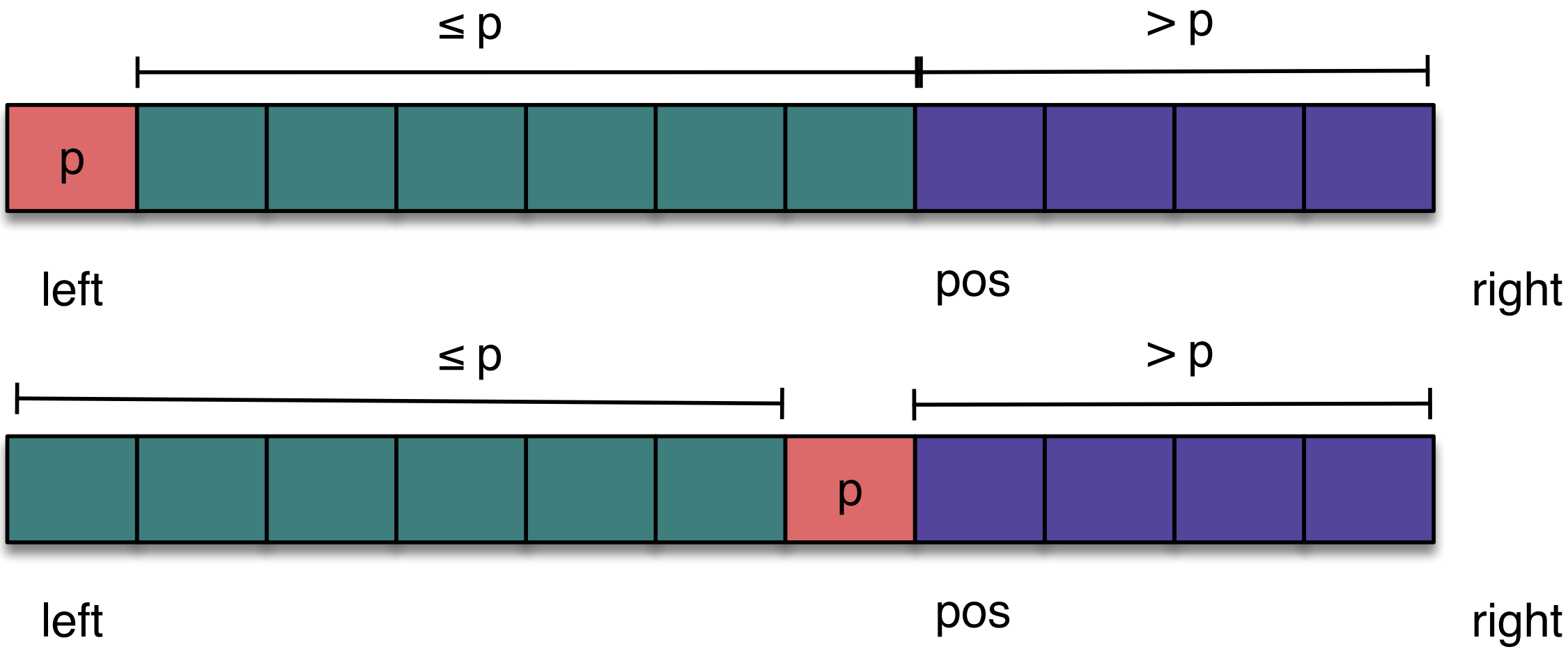
- Una de les coses que hem de garantir en tot algoritme recursiu és que les crides les fem sobre paràmetres més petits
- En el cas que ens ocupa, hem de garantir que a les crides recursives el vector a ordenar és més petit
  1. pos – left < right – left ⇔ pos < right
  2. right – pos < right – left ⇔pos > left
- La primera, vol dir que hem de garantir que hi ha algún element més gran que el pívot
- La segona, vol dir que hi ha algún element més petit o igual que el pívot
  - Cosa que hem garantit agafant com a pívot un dels elements [left, right)

… l una solució

- El “truco” consisteix en treure el pívot de la zona a particionar per, una vegada feta la partició, reintegrar-lo al lloc que li toca



… I una solució



```
public void quickSort(int[] v, int left, int right) {
    // 0 <= left <= right <= v.length
    if (right - left > 1) {
        int pivotPos = choosePivotPosition(v, left, right);
        int pivotValue = v[pivotPos];
        swap(v, left, pivotPos);
        int pos = partition(v, pivotValue, left + 1, right);
        swap(v, left, pos - 1);
        quickSort(v, left, pos - 1);
        quickSort(v, pos, right);
    }
}

public int choosePivotPosition(int[] v, int left, int right) {
    return left + (right - left) / 2;
}
```