Esquemas Algorítmicos

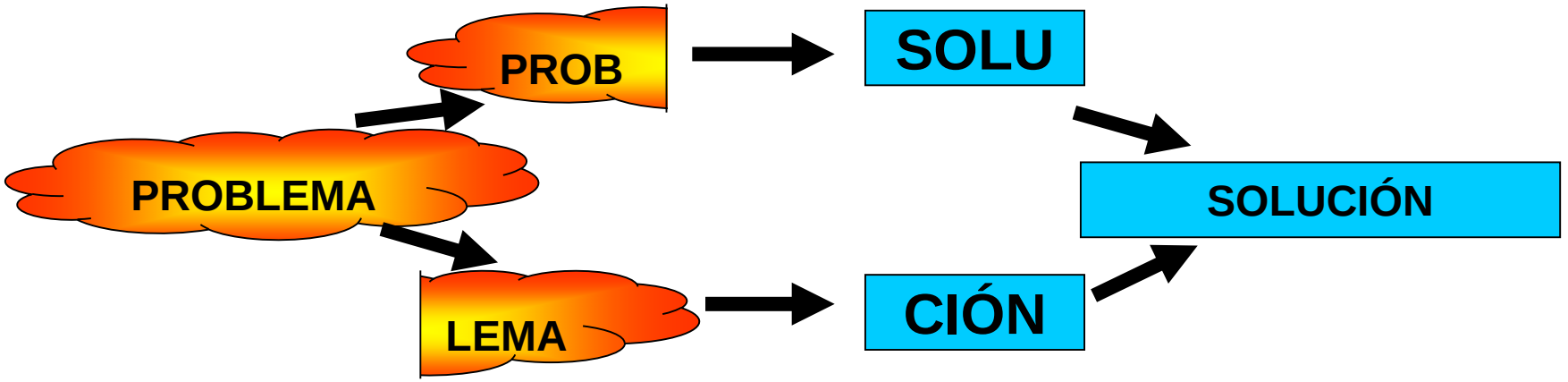
- Conocer un conjunto de técnicas de resolución de familias de problemas.

- Dado un problema concreto: caracterizar convenientemente el problema, valorar y elegir la técnica más adecuada.

Introducción

- descomponer el ejemplar a resolver en un cierto número de subejemplares más pequeños del mismo problema;
- resolver independientemente cada subejemplar;
- combinar los resultados obtenidos para construir la solución del ejemplar original.

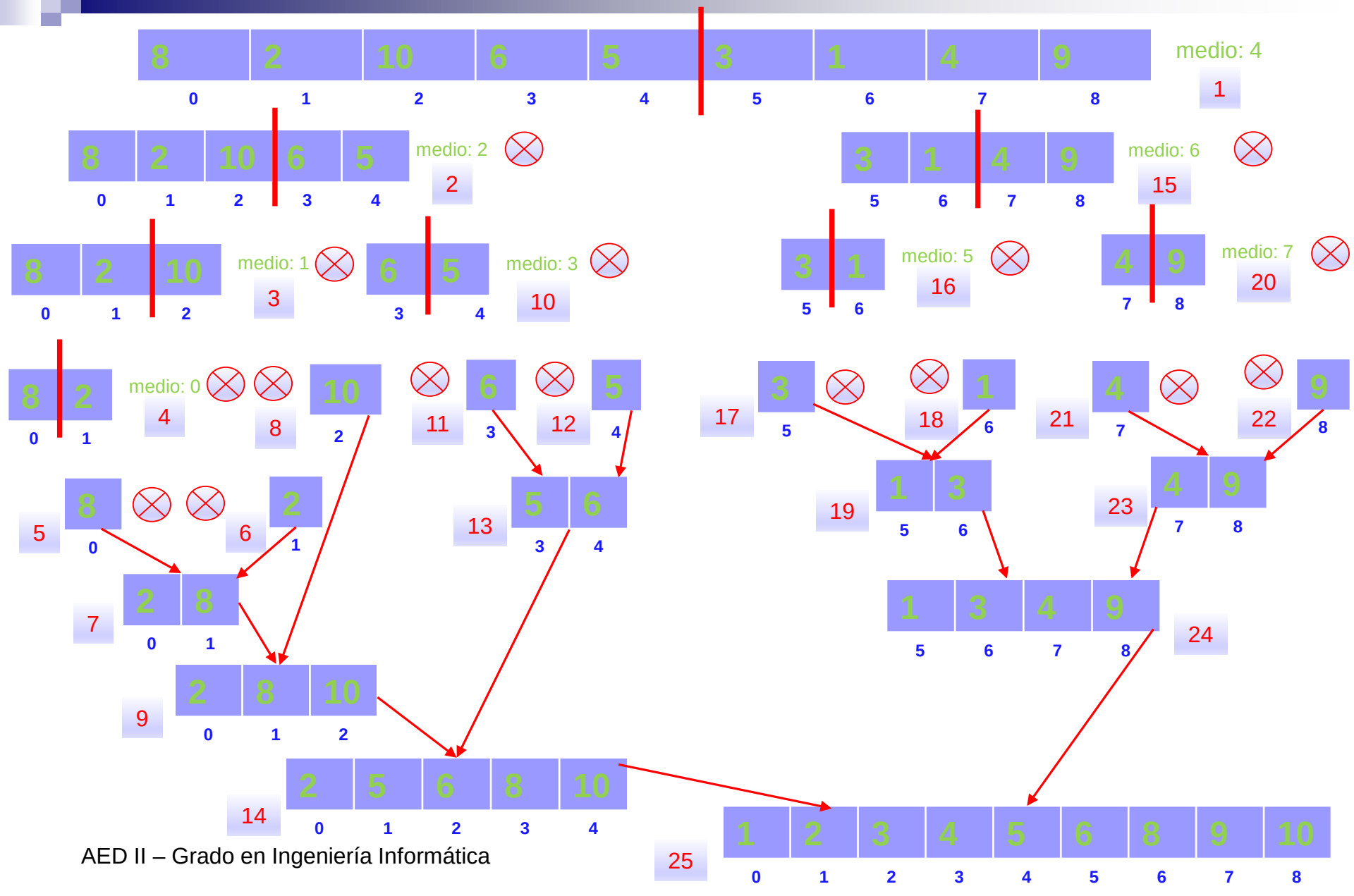
  - aplicar esta técnica recursivamente



Ordenación por mezcla (mergesort)

- El **MERGESORT** consiste en :
  - Dividir los elementos en dos secuencias de la misma longitud aproximadamente.
  - ***Ordenar*** de forma independiente cada subsecuencia.
  - ***Mezclar*** las dos secuencias ordenadas para producir la secuencia final ordenada.

Ejemplo: Mergesort



Código MergeSort

```
public void mergeSort (int [] aux, int inicio, int fin){
    if (inicio <  fin)
    {
        int medio = (inicio+ fin)/2;
        mergeSort (aux, inicio, medio);
        mergeSort (aux, medio+1, fin);

        merge (aux, inicio, medio, medio+1, fin);
    }
}
```

Código Mezcla de MergeSort.

- El proceso de ***mezcla*** es el siguiente:
  - Repetir mientras haya elementos en una de las dos secuencias:
    - Seleccionar el menor de los elementos de las subsecuencias y añadirlo a la secuencia final ordenada.
    - Pasar al siguiente elemento en la subsecuencia seleccionada en el paso anterior.
  - Copiar en la secuencia final los elementos de la subsecuencia en la que aún quedan elementos.

Código Mezcla de MergeSort.

```
public static void merge (int [] aux, int inicio1, int fin 1, int inicio2, int fin2) {
        int i = inicio1; //Variable de primer elemento de la primera subsecuencia
        int j = inicio2; //Variable del primer elemento de la segunda subsecuencia

         ArrayList<Integer> temp = new ArrayList<>();

        while (i <=  fin1 && j<= fin2) {
                  if (aux[i] <= aux[j]) {
                      temp.add(aux[i++]);
                  }else {
                        temp.add(aux[j++]);
                  }
        }

        while (i <= fin1) {
           temp.add(aux[i++]);
        }
        while (j <= fin2) {
            temp.add(aux[j++]);
       }

        for (i = inicio1; i <= fin2; i++) { //Paso todos los elementos del Temporal al array
            aux[i] = temp.remove(0);
       }
}
```

Ordenación Rápida (QuickSort)

- En la ordenación rápida, **la secuencia inicial de elementos se divide en dos subsecuencias de diferente tamaño**. La obtención de las dos subsecuencias es el proceso que acarrea más tiempo mientras que la combinación de las subsecuencias ordenadas para obtener la secuencia final consume muy poco tiempo.
- Para dividir en dos la secuencia de elementos, se selecciona un elemento sobre el cual efectuar la división, el ***PIVOTE***. Se dividen los elementos en dos grupos, los elementos menores que el pivote y aquellos mayores o igual al pivote.

Ordenación QuickSort.

- La elección del elemento Pivote se puede seleccionar de diferentes formas:
  - El mayor de los dos primeros elementos distintos encontrados.
  - El primer elemento.
  - El último elemento.
  - El elemento medio.
  - Un elemento aleatorio.
  - Mediana de tres elementos (El primer elemento, el elemento del medio y el último elemento).

Pasos a seguir QuickSort.

- El método de ordenación rápida se basa en **ordenar los elementos comprendidos entre aux_i y aux_j conforme a las siguientes cinco etapas**:

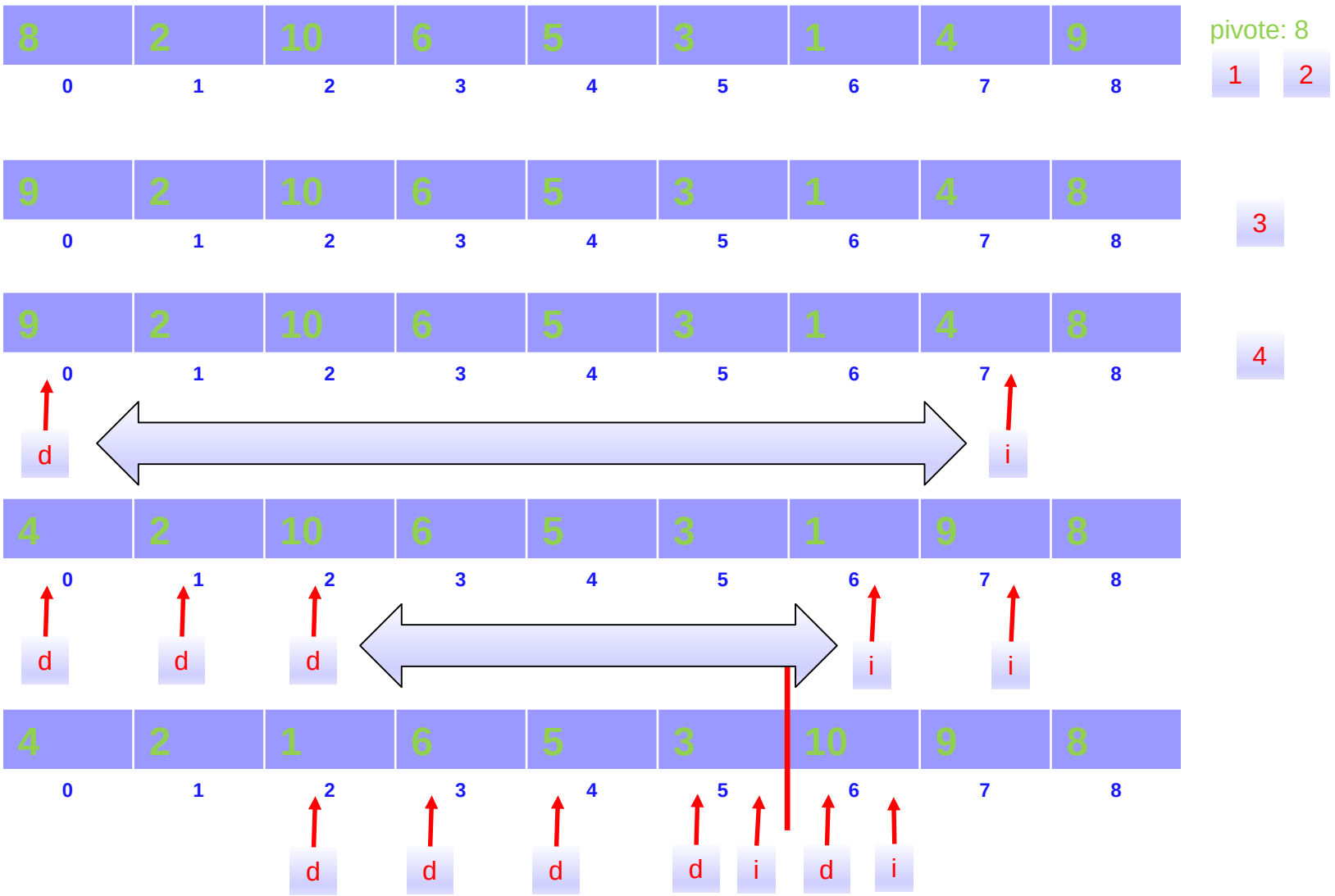
1. Si desde aux_i a aux_j hay al menos dos elementos distintos entonces comenzar la aplicación del algoritmo.
2. **Seleccionar el PIVOTE** como el elemento mayor de los dos primeros elementos distintos encontrados.
3. **Intercambiar el elemento PIVOTE con el último**
4. **Permutar los elementos** desde aux_i hasta aux_j de modo que, para algún i <= **k** <= j :

   $aux_i, ..., aux_{k-1}$ < PIVOTE

   $aux_k, ..., aux_j$ >= PIVOTE

   Es decir, en las (k-1) primeras posiciones queden los elementos menores que pivote, mientras que en la posición **k** hacia delante queden los elementos mayores o iguales que el pivote.
5. Invocar a: QUICKSORT desde **i** hasta **(k – 1)**
   QUICKSORT desde **k** hasta **j**

Paso1, 2, 3 y 4


8 2 10 6 5 3 1 4 9
0 1 2 3 4 5 6 7 8
pivote: 8
1
2
9 2 10 6 5 3 1 4 8
0 1 2 3 4 5 6 7 8
3
9 2 10 6 5 3 1 4 8
0 1 2 3 4 5 6 7 8
4
d
i
4 2 10 6 5 3 1 9 8
0 1 2 3 4 5 6 7 8
d
d
d
i
i
4 2 1 6 5 3 10 9 8
0 1 2 3 4 5 6 7 8
d
d
d
d
i
d
i

Código: selección del Pivote

- Para la elección del pivote se puede utilizar la siguiente función que localiza el elemento mayor de los dos primeros elementos distintos existentes entre *inicio* y *fin*.

```
private static int  buscaPivote (int [] aux, int inicio, int fin)
{
        int primer = aux[inicio];
        int k = inicio + 1;

        while (k <= fin)
        {
           if (aux[k] > primer) {
               return k;
           }
           else if (aux[k] < primer) {
                       return inicio;
                  }
                else {
                           k++;
                 }
        }
        //Si llega al final del array y todos los elementos son iguales, o si sólo hay un elemento
        return -1;
}
```

Paso 4: Permutación de elementos

- Para el paso 4 de permutación de los elementos se utilizan dos cursores:
  - derecha : para mover a la derecha mientras el elemento sea menor que el pivote.
  - izquierda: para mover a la izquierda mientras el elemento sea mayor o igual que el pivote
- Repetir mientras derecha < izquierda:
  - EXPLORACIÓN: Mover

    derecha hacia la derecha sobre cualquier elemento MENOR que el pivote y

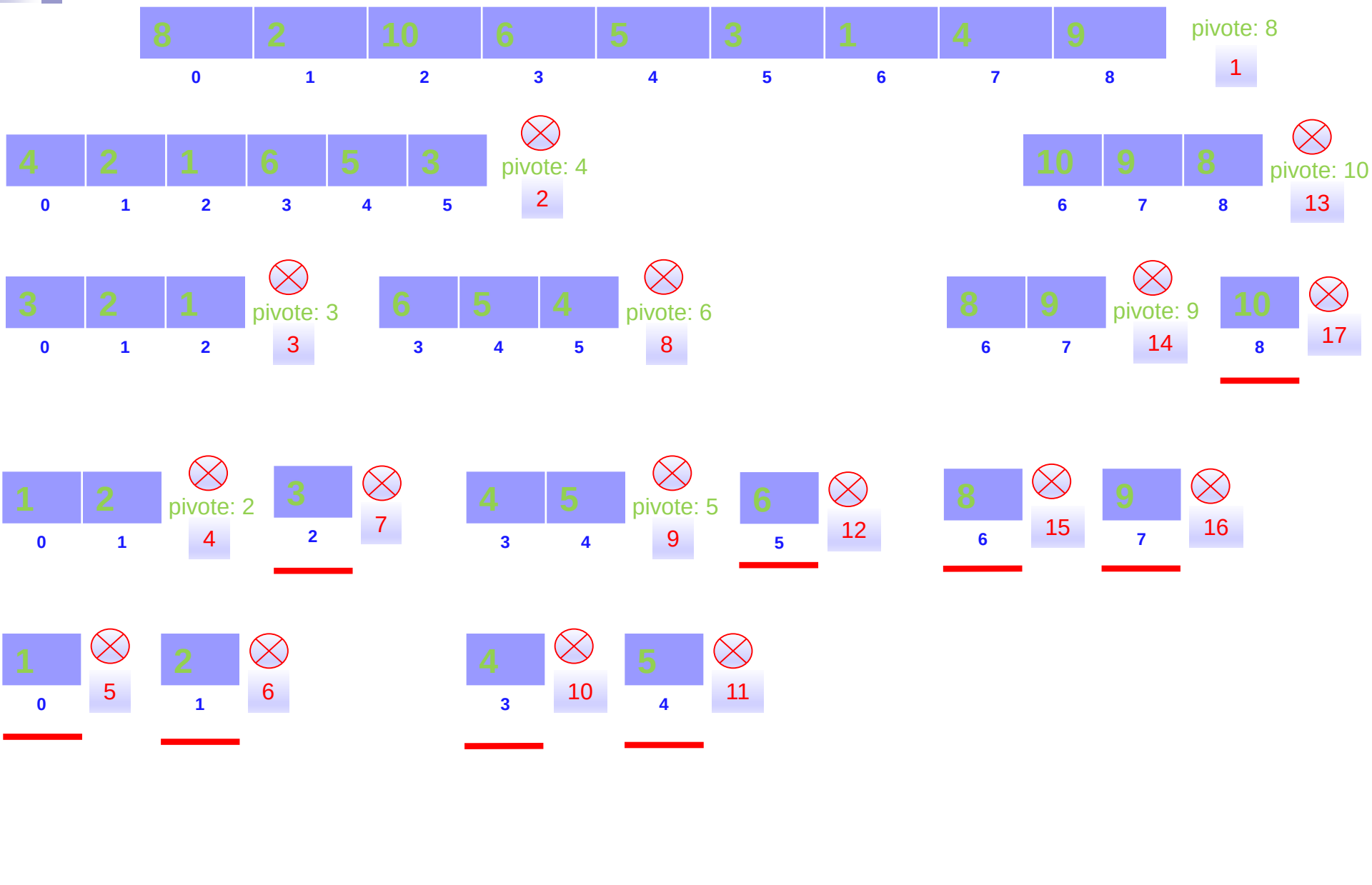
    izquierda hacia la izquierda sobre cualquier elemento MAYOR o IGUAL que el pivote.
  - CAMBIO : Si derecha < izquierda se intercambian $aux_{derecha}$ y $aux_{izquierda}$
- COMPROBACIÓN: Si derecha > izquierda hemos acabado con éxito la reordenación.

Código: partición en subproblemas

```
private static int particion (int [] aux, int inicio, int fin, int pivote)
{
	int derecha = inicio;
	int izquierda = fin-1; // pivote está en la última posición
	do {
		while (aux[derecha] < pivote) {
			derecha++;
		}
		while (aux[izquierda] >= pivote) {
			izquierda--;
		}
		// intercambia los valores de las posiciones derecha e izquierda
		if (derecha< izquierda)
			intercambiar(aux,derecha,izquierda);
	} while (derecha <= izquierda) ;

return derecha; //primera posición de la segunda mitad
}
```

Ejemplo: Quicksort



Código QuickSort

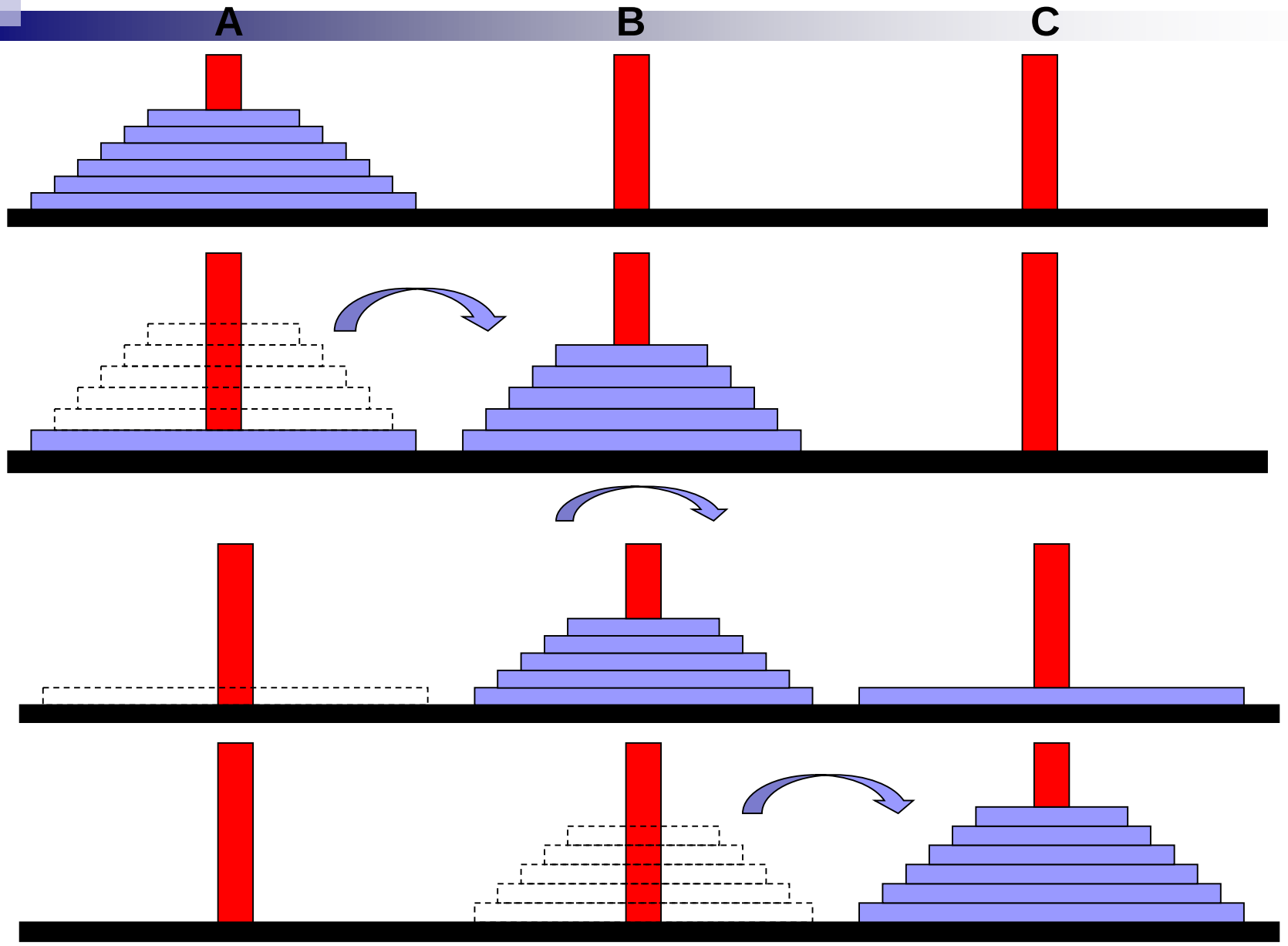
```
public static void quickSort (int [] aux, int inicio, int fin)
{
    int indicePivote = buscaPivote(aux, inicio, fin);
    if (indicePivote != -1)
    {
       int pivote = aux[indicePivote];
       intercambiar(aux,indicePivote,fin); //pivote a la última posición
       int k = particion(aux,inicio,fin,pivote);
       quickSort(aux, inicio, k-1);
       quickSort(aux, k,fin);
    }
}
```

El esquema divide y vencerás: Torres de Hanoi

- El de las **Torres de Hanoi** es un juego matemático consistente en mover unos discos de una torre a otra. La leyenda cuenta que existe un templo (llamado *Benares*), bajo la bóveda que marca el centro del mundo, donde hay tres varillas de diamante creadas por Dios al crear el mundo, colocando 64 discos de oro en la primera de ellas. Unos monjes mueven los discos a razón de uno por día, y, el día en que tengan todos los discos en la tercera varilla, el mundo terminará. Como se comprobará a continuación, en realidad 64 discos son suficientes para muchos años.

- En este juego, se trata de pasar un número de discos (típicamente, con tres existe una dificultad suficiente como para plantearlo como un pasatiempo), de un poste de origen (el primero, más a la izquierda) a un poste de destino (el tercero, a la derecha), utilizando como poste auxiliar el del medio. Sólo se puede mover un disco de cada vez, y nunca poner un disco sobre un segundo que sea de menor diámetro que el primero. Así, al comienzo del juego todos los discos están apilados en el primero (el de la izquierda), cada disco se asienta sobre otro de mayor diámetro, de manera que tomados desde la base hacia arriba, su tamaño es decreciente. El objetivo, como ya se ha dicho, es mover uno a uno los discos desde el poste A (origen) al poste C (destino) utilizando el poste B como auxiliar, para lo cuál se puede emplear una técnica *divide y vencerás*, como se explica a continuación.

El esquema divide y vencerás: Torres de Hanoi



El esquema divide y vencerás: Torres de Hanoi

Vamos a plantear la solución de tal forma que el problema vaya dividiendo en problemas más pequeños, y a cada uno de ellos aplicarles la misma solución. Se puede expresar así:

El problema de mover n discos de A a C consiste en:

mover los n-1 discos superiores de A a B
mover el disco n de A a C
mover los n-1 discos de B a C

Un problema de tamaño *n* ha sido transformado en un problema de tamaño *n-1*. A su vez cada problema de tamaño *n-1* se transforma en otro de tamaño *n-2* (empleando el poste libre como auxiliar).

El problema de mover los n-1 discos de A a B consiste en:

mover los n-2 discos superiores de A a C
mover el disco n-1 de A a B
mover los n-2 discos de C a B

De este modo se va progresando, reduciendo cada vez un nivel de dificultad del problema hasta que sólo haya que mover un único disco. La técnica consiste en ir intercambiando la finalidad de los postes, origen, destino y auxiliar. La condición de terminación es que el número de discos sea 1. Cada acción de mover un disco realiza los mismos pasos, por lo que puede ser expresada de manera recursiva.