DIVIDE Y VENCERÁS

- INTRODUCCIÓN
- ESQUEMA GENERAL
- EFICIENCIA
- EJEMPLOS: POTENCIA n-ésima DE a, BÚSQUEDA BINARIA, MERGESORT y QUICKSORT
- TAREAS

INTRODUCCIÓN

Es una técnica de diseño de algoritmos que consiste en resolver un problema a partir de la solución de subproblemas del mismo tipo y de menor tamaño.

Si los subproblemas son todavía relativamente grandes se aplicará de nuevo esta técnica hasta alcanzar subproblemas lo suficientemente pequeños para ser solucionados directamente.

Todo ello, naturalmente, sugiere el uso de recursión.

INTRODUCCIÓN

La resolución de un problema mediante esta técnica consta fundamentalmente de los siguientes pasos:

- **Dividir** el problema en ***a*** subproblemas del mismo tipo, pero de menor tamaño.
- **Resolver** independientemente todos los subproblemas.
- Por último, **combinar** las soluciones obtenidas en el paso anterior para construir la solución del problema original.

ESQUEMA GENERAL

Función DyV ($\overline{\boldsymbol{x}}$:*T1*) *retorna* ($\overline{\boldsymbol{y}}$:*T2*)
si $\overline{\boldsymbol{x}}$ *es suficientemente pequeño entonces devolver la solución para* $\overline{\boldsymbol{x}}$
sino
descomponer $\overline{\boldsymbol{x}}$ *en a casos más pequeños* $\overline{\boldsymbol{x_1}}, \overline{\boldsymbol{x_2}}, \ldots, \overline{\boldsymbol{x_a}}$
para $i \leftarrow 1$ *hasta a hacer*
$\overline{y_i} \leftarrow$ DyV($\overline{\boldsymbol{x_i}}$)
fpara
combinar los $\overline{\boldsymbol{y_i}}$ *para obtener la solución* $\overline{\boldsymbol{y}}$
devolver $\overline{\boldsymbol{y}}$
fsi
ffuncion

EFICIENCIA

En cuanto a la eficiencia hay que tener en consideración un factor importante durante el diseño del algoritmo: **el número de subproblemas y su tamaño**, pues esto influye de forma notable en la complejidad temporal del algoritmo resultante.

- El número **a** de subproblemas debe ser **pequeño**.
- Es importante conseguir que los **subproblemas** sean **independientes**, es decir, que no haya solapamiento entre ellos.

ECUACIÓN DE RECURRENCIA

El diseño Divide y Vencerás produce algoritmos recursivos cuyo tiempo de ejecución se puede expresar mediante una ecuación en recurrencia del tipo:

$$T(n) = \begin{cases} cn^k & si \quad 1 \leq n < b \\ a\,T(n/b) + cn^k & si \quad n \geq b \end{cases}$$

donde a>0, c>0, k≥0 y b>1.

El valor de ***a*** representa el número de subproblemas, ***n/b*** es el tamaño de cada uno de ellos y ***cn***^{k} representa el coste de descomponer y combinar o bien el de resolver un problema elemental.

ECUACIÓN DE RECURRENCIA

La solución a esta ecuación puede alcanzar distintas complejidades, siendo estas:

$$T(n) = \begin{cases} \theta(n^k) & si \;\; a < b^k \\ \theta(n^k \log n) & si \;\; a = b^k \\ \theta(n^{log_b a}) & si \;\; a > b^k \end{cases}$$

Las diferencias surgen de los distintos valores que pueden tomar ***a*** y ***b***, es decir, **número de subproblemas** y **su tamaño**.

Lo importante es observar que en todos los casos la complejidad temporal es de orden **polinómico o polilogarítmico** pero **nunca exponencial** frente a algoritmos recursivos que pueden alcanzar esta complejidad en muchos casos.

EJEMPLOS

Existen una serie de ejemplos considerados como representantes clásicos de este diseño, muy especialmente los algoritmos de ordenación: ordenación por fusión (Mergesort) y ordenación rápida (Quicksort).

Otros, como la búsqueda binaria, probablemente se trate de la aplicación más sencilla de Divide y Vencerás.

POTENCIA n-ÉSIMA DE a

Algoritmo visto en el tema “Diseño de Algoritmos Recursivos”.-

$\{ a \geq 0 \wedge n \geq 0 \}$

Funcion **POTENCIA** (a:entero, n:entero) retorna (p:entero)

si n = 0 entonces retorna 1

sino retorna **POTENCIA**(a, n-1) * a

fsi

ffunción

$\{ p = a^n \}$

POTENCIA n-ÉSIMA DE a según DyV

$\{ a \geq 0 \wedge n \geq 0 \}$

Funcion **POTENCIA_DyV** (a:entero, n:entero) retorna (p:entero)

$\{ p = a^n \}$

¿POTENCIA_DyV (a, n)?

POTENCIA n-ÉSIMA DE a según DyV

$$\overbrace{a * a * a * a \quad \ldots * a}^{\textbf{n veces}}$$

“**Dividimos**” la serie de a´s por la mitad

$$\underbrace{\overbrace{a * a * \ldots}^{\textbf{n/2 veces}} \overbrace{.. * a \;\; * a}^{\textbf{n/2 veces}}}_{\textbf{n veces}}$$

POTENCIA n-ÉSIMA DE a según DyV

Resolvemos cada subproblema de forma independiente

$$\underbrace{a * a}_{a^{n/2}} * \ldots\ldots \underbrace{* a * a}_{a^{n/2}}$$

Y, ya sólo nos queda **combinar** las soluciones parciales. En este punto hay que tener en cuenta si n es par o no, con objeto de construir de forma correcta la solución original, esto es, a^n.

POTENCIA n-ÉSIMA DE a según DyV

Tener en cuenta que si n es par entonces

$$a^n = a^{n/2} * a^{n/2}$$

mientras que si n es impar entonces

$$a^n = a^{n/2} * a^{n/2} * a$$

POTENCIA n-ÉSIMA DE a según DyV

$\{ a \geq 0 \land n \geq 0 \}$

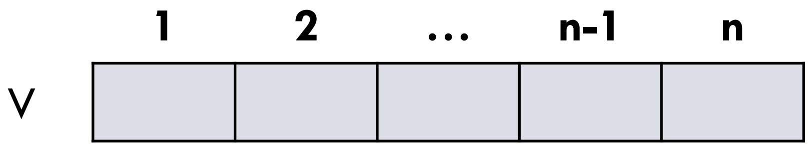
```
Funcion POTENCIA_DyV (a:entero, n:entero) retorna (p:entero)
var p:entero fvar
si n = 0 entonces retorna 1
    sino
            p = POTENCIA_DyV (a, n/2)
            si n es par entonces retorna p*p
                sino  retorna p*p*a
            fsi
fsi
ffunción
```

$\{ p = a^n \}$

$T(n) \in \theta(\log_2 n)$

BÚSQUEDA BINARIA

Dado un vector V[1..n] de n números enteros, siendo n≥0, se trata de diseñar una función recursiva que retorne la posición que ocupa un valor x, si está en el vector, o la que debería ocupar si no está. El vector se encuentra ordenado en sentido creciente.



BÚSQUEDA BINARIA

Estrategia:

1) "**Dividir**" el vector V por la mitad (**m = (1+n) div 2**)

	1	2	…	m-1	m	m+1	…	n-1	n
V									

BÚSQUEDA BINARIA

2) Comprobar si x está en la posición central, esto es, en V[m]. Pueden ocurrir tres cosas:

2.1) Que sí que esté, por tanto el problema está resuelto y la función retornará el valor m

	1	2	…	m-1	m	m+1	…	n-1	n
V									

BÚSQUEDA BINARIA

2.2) Que no esté y x < V[m], entonces continúa la búsqueda en la primera mitad de V, esto es, en V[1..m-1]

	1	**2**	**…**	**m-1**	**m**	**m+1**	**…**	**n-1**	**n**
V									

BÚSQUEDA BINARIA

2.3) Que no esté y x > V[m], entonces continúa la búsqueda en la segunda mitad de V, esto es, en V[m+1..n]

	1	2	…	m-1	m	m+1	…	n-1	n
V									

BÚSQUEDA BINARIA


```
función Busqueda_binaria (V[1..n]:vector de enteros; x, inicio, fin:entero) retorna (e:entero, b:booleano)
    var m : entero fvar
    si inicio > fin entonces retorna ( inicio, falso )
      sino
           m = (inicio + fin) div 2
           si x = V[m] entonces retorna (m, verdadero)
             sino      si x > V[m] entonces retorna Busqueda_binaria (V, x, m+1, fin)
                           sino retorna Busqueda_binaria (V, x, inicio, m-1)
                       fsi
           fsi
    fsi
ffunción
```

donde el vector V está ordenado en sentido ascendente y la llamada inicial a la función es: Busqueda_binaria(V, x, 1, n)

MERGESORT

Dado un vector V[1..n] de n números enteros, siendo n≥0, el problema consiste en ordenar el vector en sentido creciente.

	1	**2**	**…**	**n-1**	**n**
V					

MERGESORT

Estrategia:

1) "**Dividir**" el vector V por la mitad (**m = (1+n) div 2**)

	1	2	…	m-1	m	m+1	…	n-1	n
V									

MERGESORT

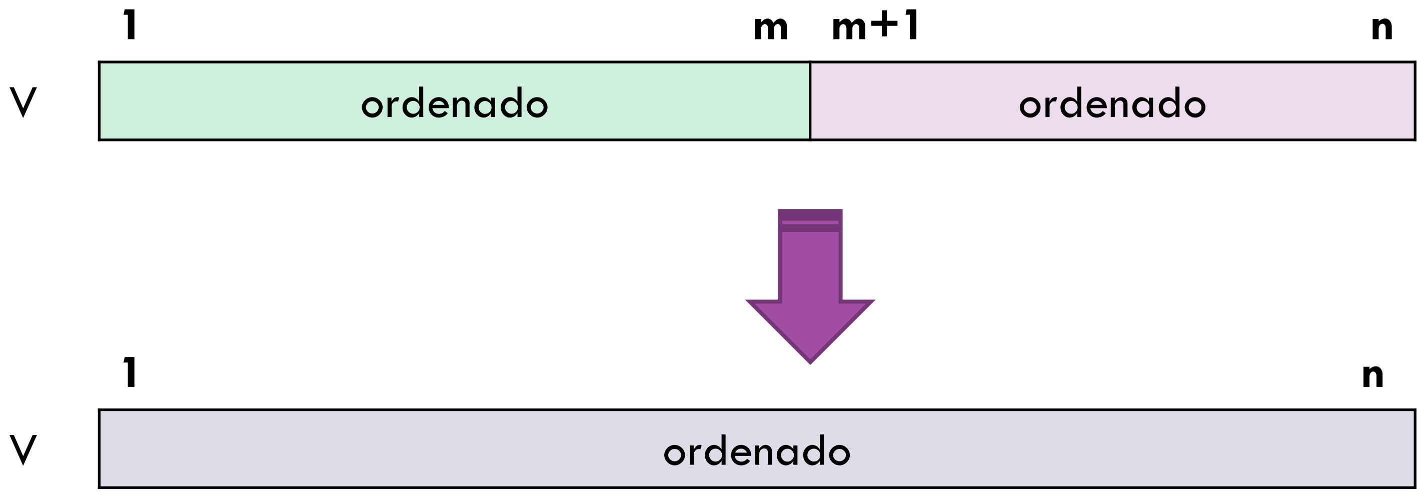
2) **Resolver** los dos subproblemas, esto es, V[1..m] y V[m+1..n]

Los casos bases se alcanza cuando hay que ordenar una sección de vector de tamaño 0 o 1. En este último caso, la única componente del vector ya está ordenada, por lo que no hay nada que hacer, al igual que en el caso de tamaño 0.

	1	2	…	m-1	m	m+1	…	n-1	n
V									

MERGESORT

3) Combinar (mezclar) en tiempo lineal las dos secciones del vector ya ordenadas para generar la ordenación del vector V[1..n]



MERGESORT


```
Procedimiento MergeSort ( V[1..n] : vector de enteros; inicio, fin : entero)
    si inicio < fin  entonces
        MergeSort (V, inicio, (inicio + fin)div2)            /* resolver */
        MergeSort (V, (inicio + fin)div2 + 1, fin)           /* resolver */
        Mezcla (V, inicio, (inicio + fin)div2, fin)          /* combinar */
    fsi
fprocedimiento
```

donde la llamada inicial a la función es: MergeSort(V, 1, n)

$$T(n) \in \theta(n\log_2 n)$$

MEZCLA

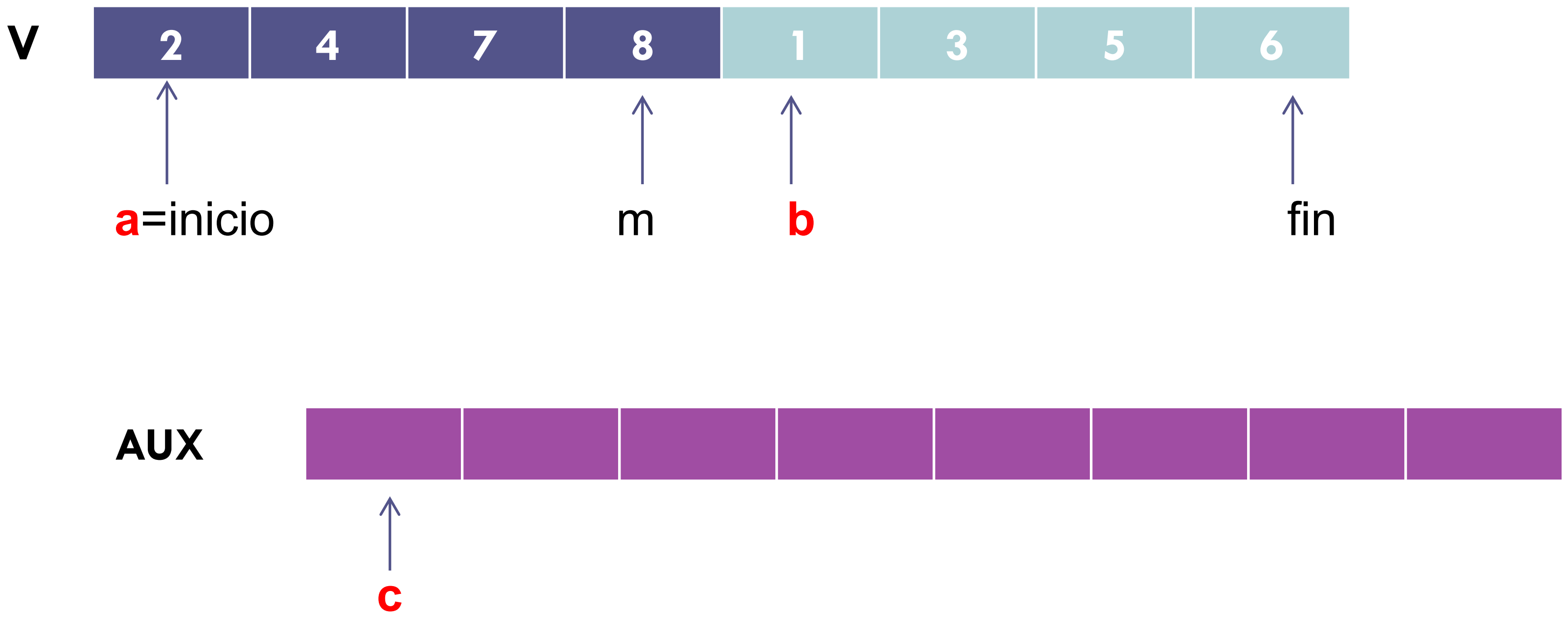
La función mezcla permite obtener un vector ordenado a partir de dos secciones del vector ya ordenadas, con un coste lineal con la suma de los tamaños de las dos secciones del vector.

Para ello se precisa la creación de un vector auxiliar en el que almacenar el resultado de la mezcla.

Vamos a verlo a través de un ejemplo:

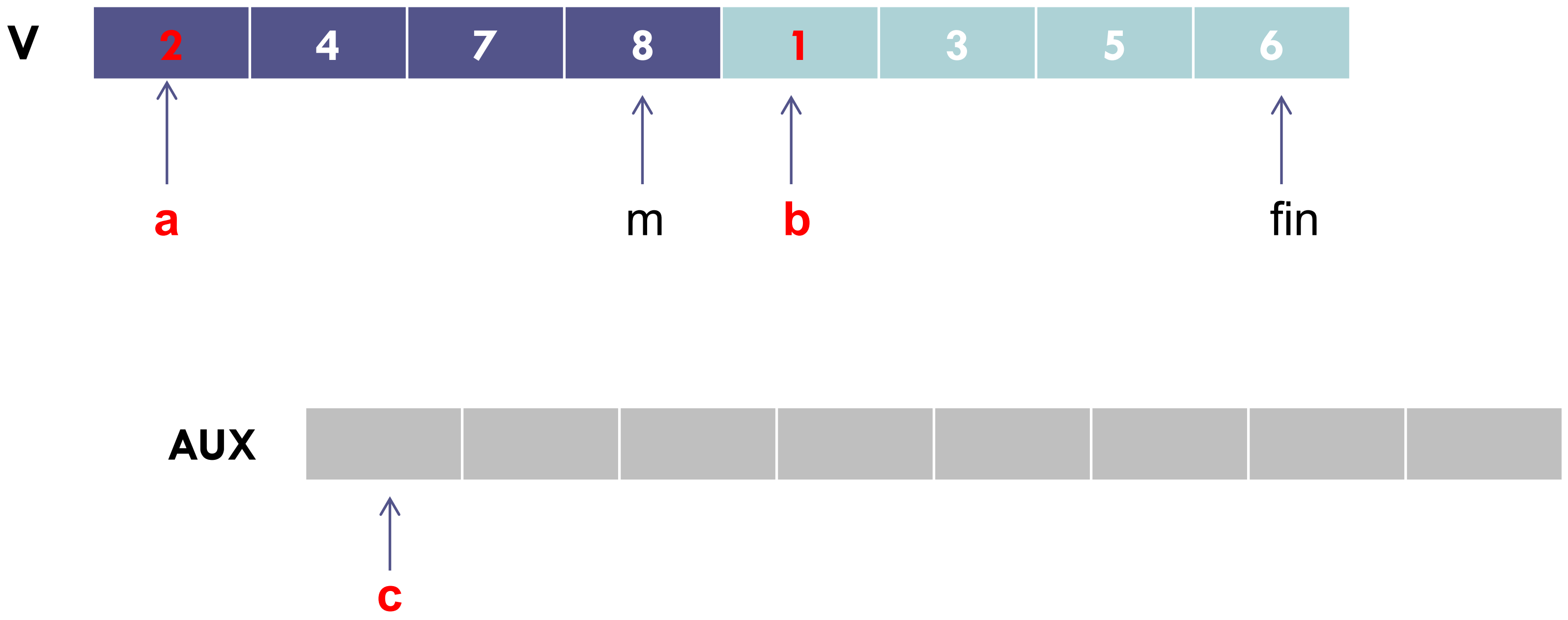
EJEMPLO DE FUNCIONAMIENTO DE MEZCLA

Situación inicial.-

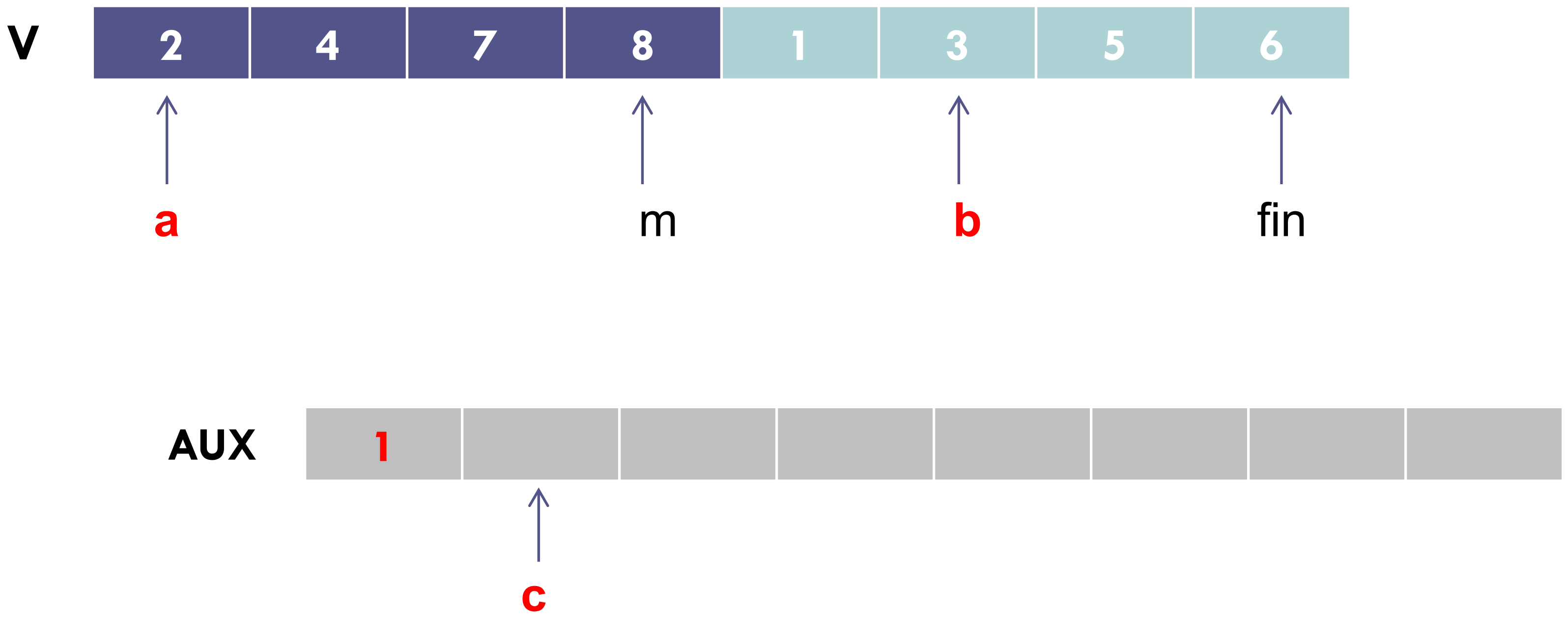


EJEMPLO DE FUNCIONAMIENTO DE MEZCLA

Paso 1.- Dado que V[a] > V[b] ⇒ en AUX introduciremos V[b], incrementaremos b y c

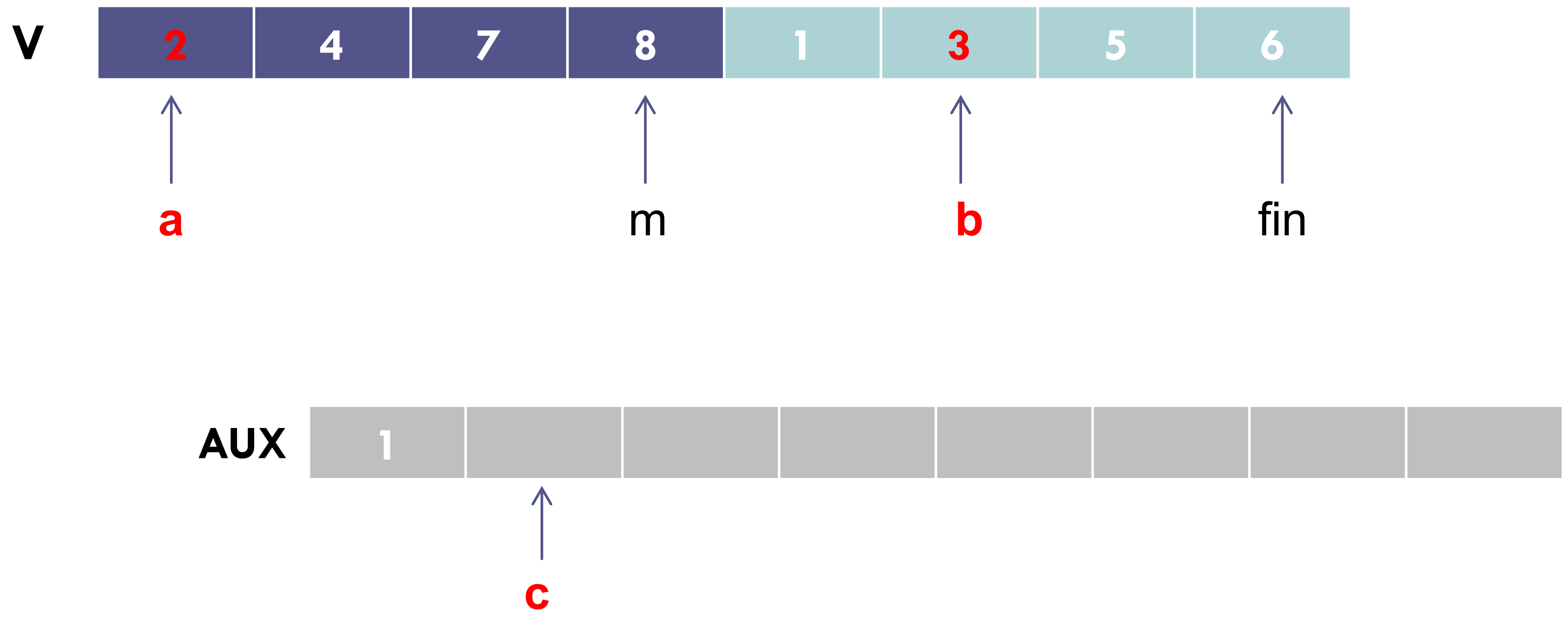


EJEMPLO DE FUNCIONAMIENTO DE MEZCLA

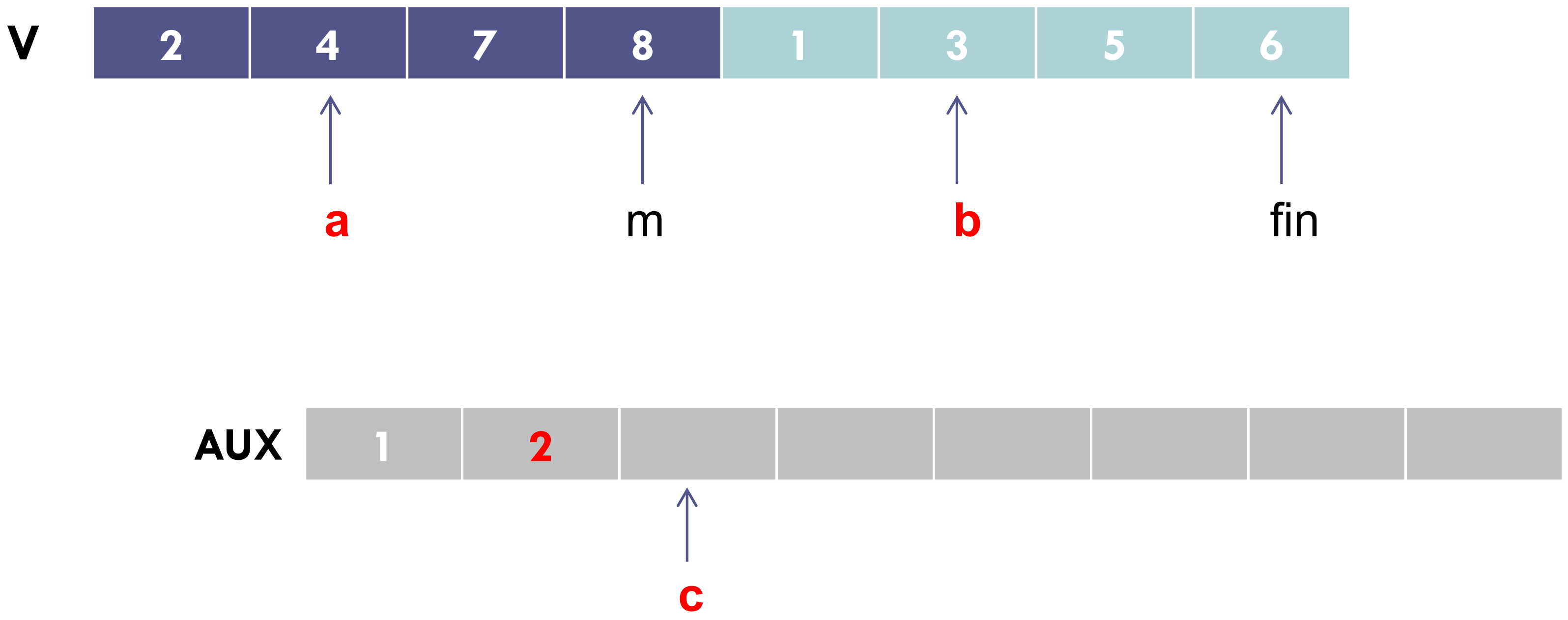


EJEMPLO DE FUNCIONAMIENTO DE MEZCLA

Paso 2.- V[a] < V[b] ⇒ en AUX introduciremos V[a], incrementaremos a y c

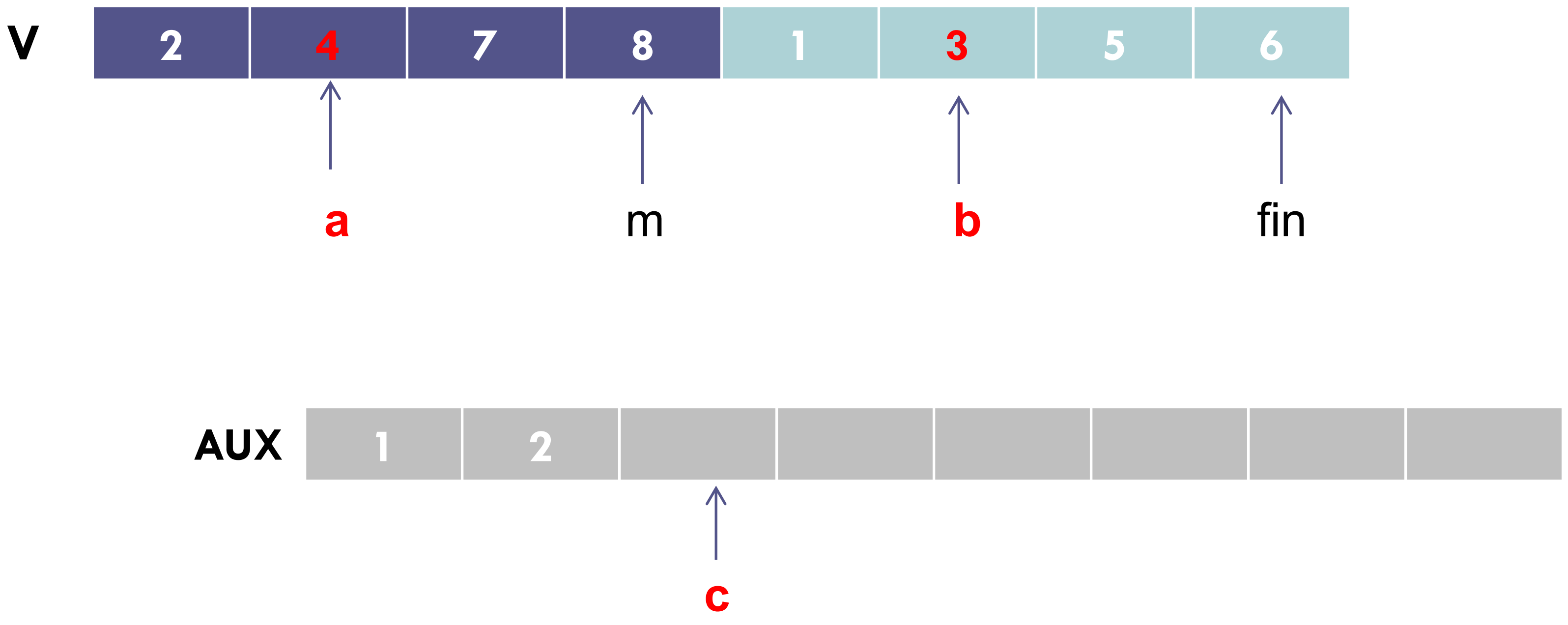


EJEMPLO DE FUNCIONAMIENTO DE MEZCLA

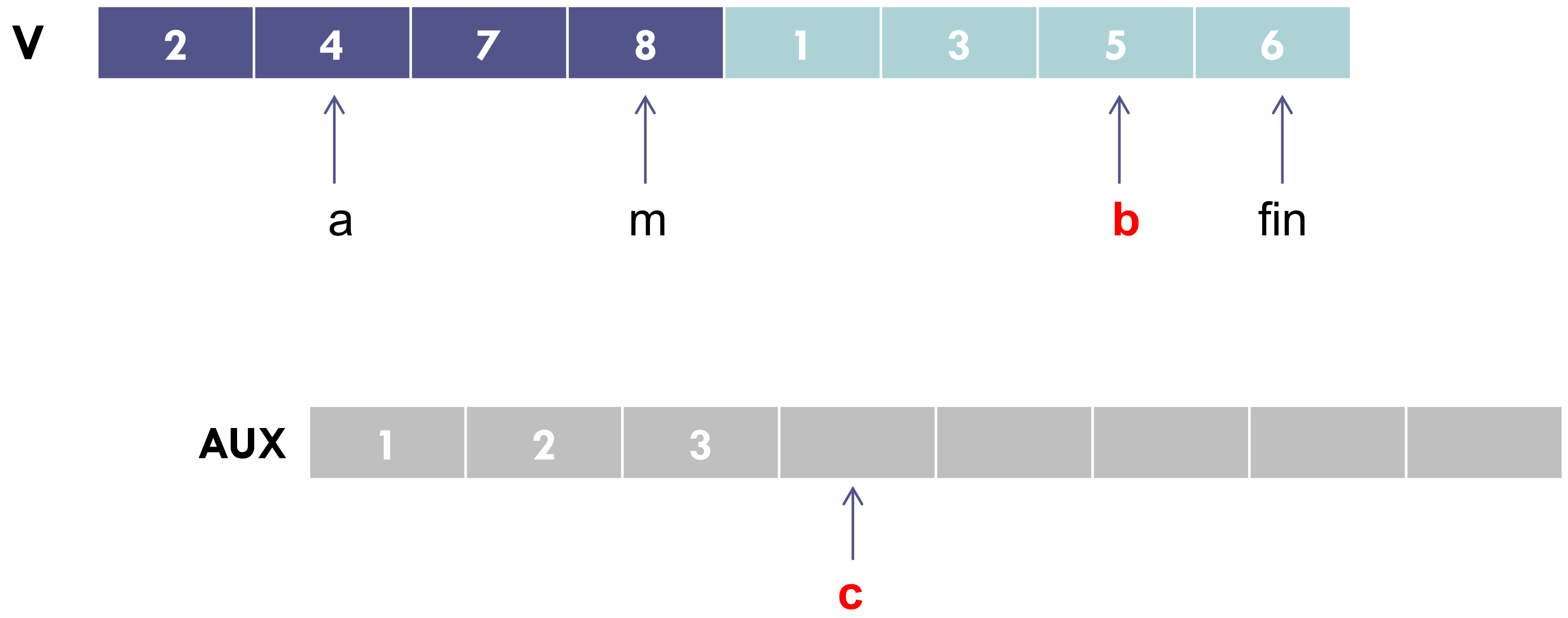


EJEMPLO DE FUNCIONAMIENTO DE MEZCLA

Paso 3.- V[a] > V[b] ⇒ en AUX introduciremos V[b], incrementaremos b y c

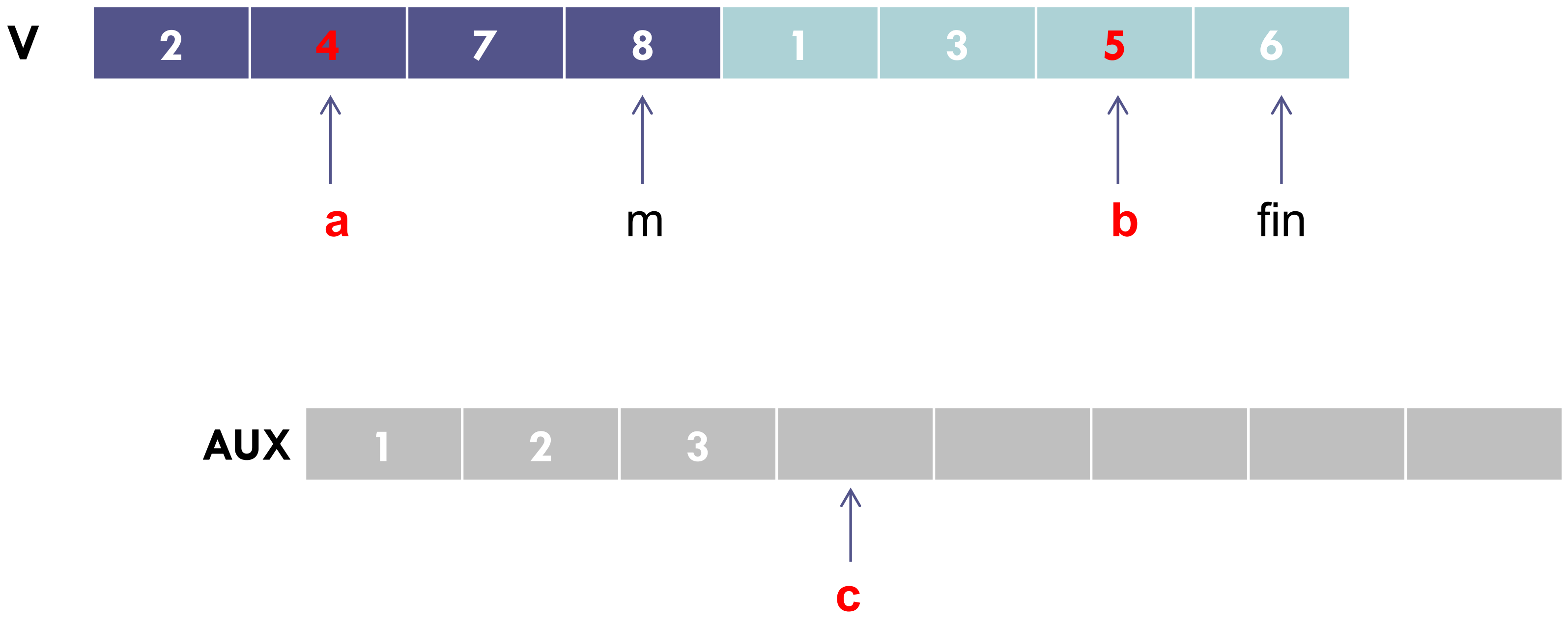


EJEMPLO DE FUNCIONAMIENTO DE MEZCLA

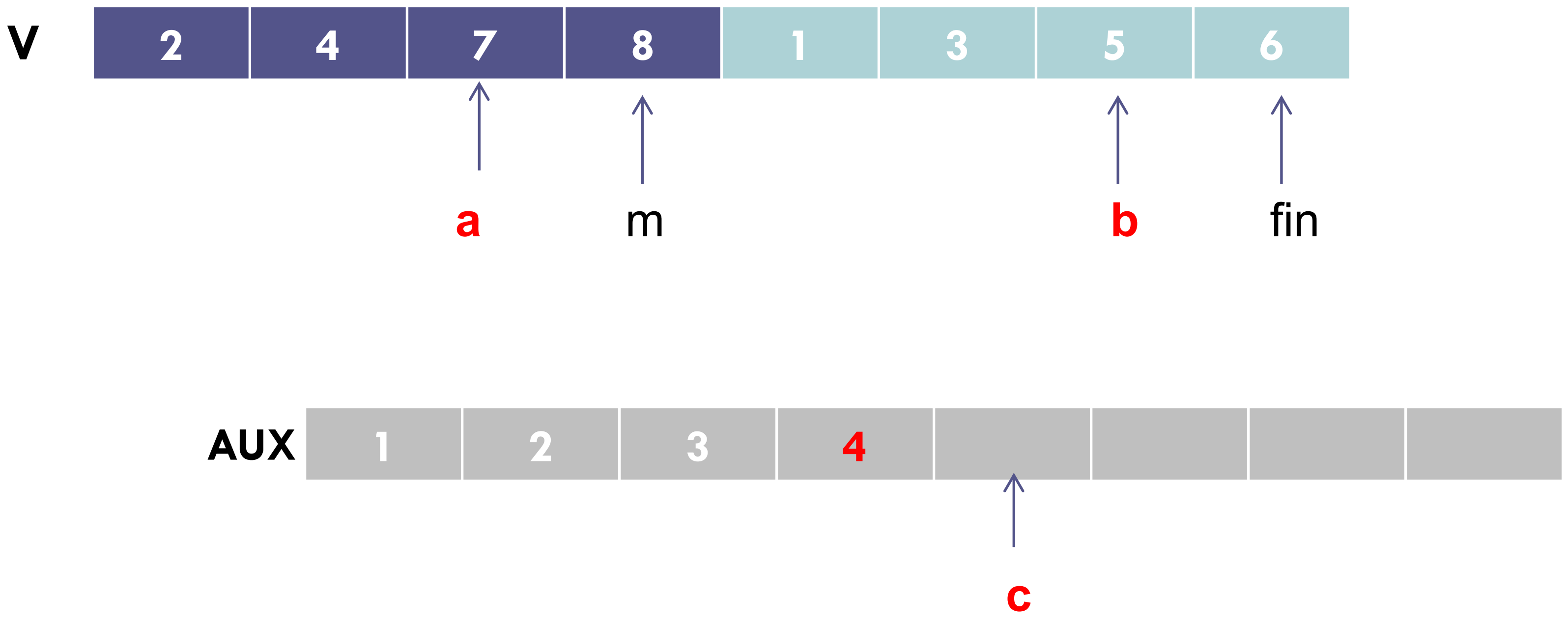


EJEMPLO DE FUNCIONAMIENTO DE MEZCLA

Paso 4.- V[a] < V[b] ⇒ en AUX introduciremos V[a], incrementaremos a y c

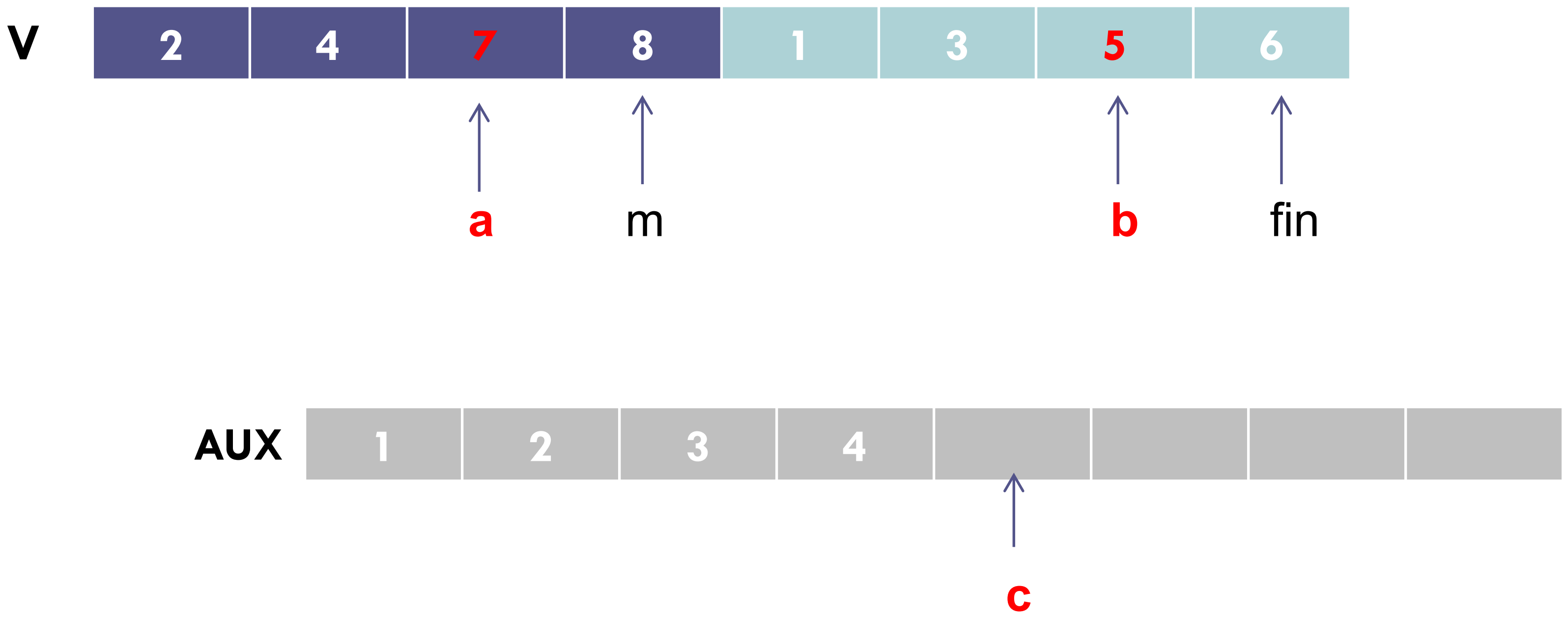


EJEMPLO DE FUNCIONAMIENTO DE MEZCLA

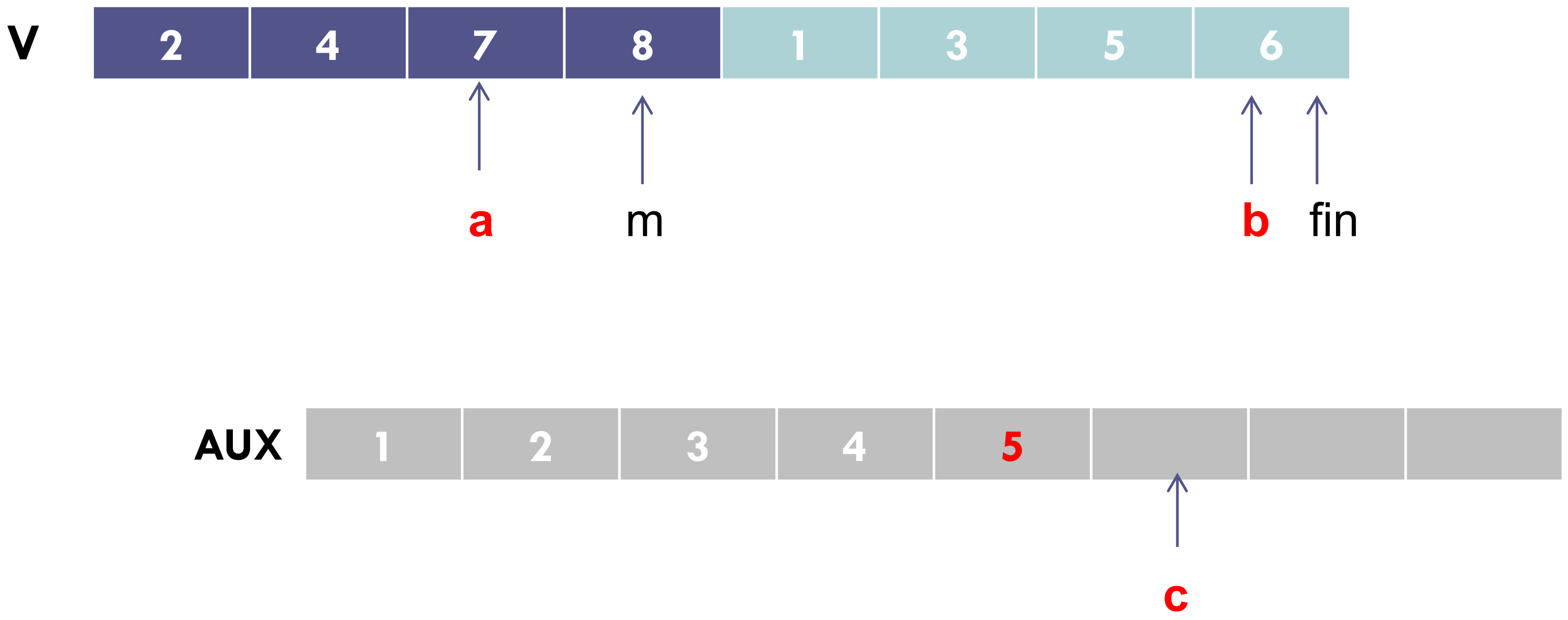


EJEMPLO DE FUNCIONAMIENTO DE MEZCLA

Paso 5.- V[a] > V[b] ⇒ en AUX introduciremos V[b], incrementaremos b y c

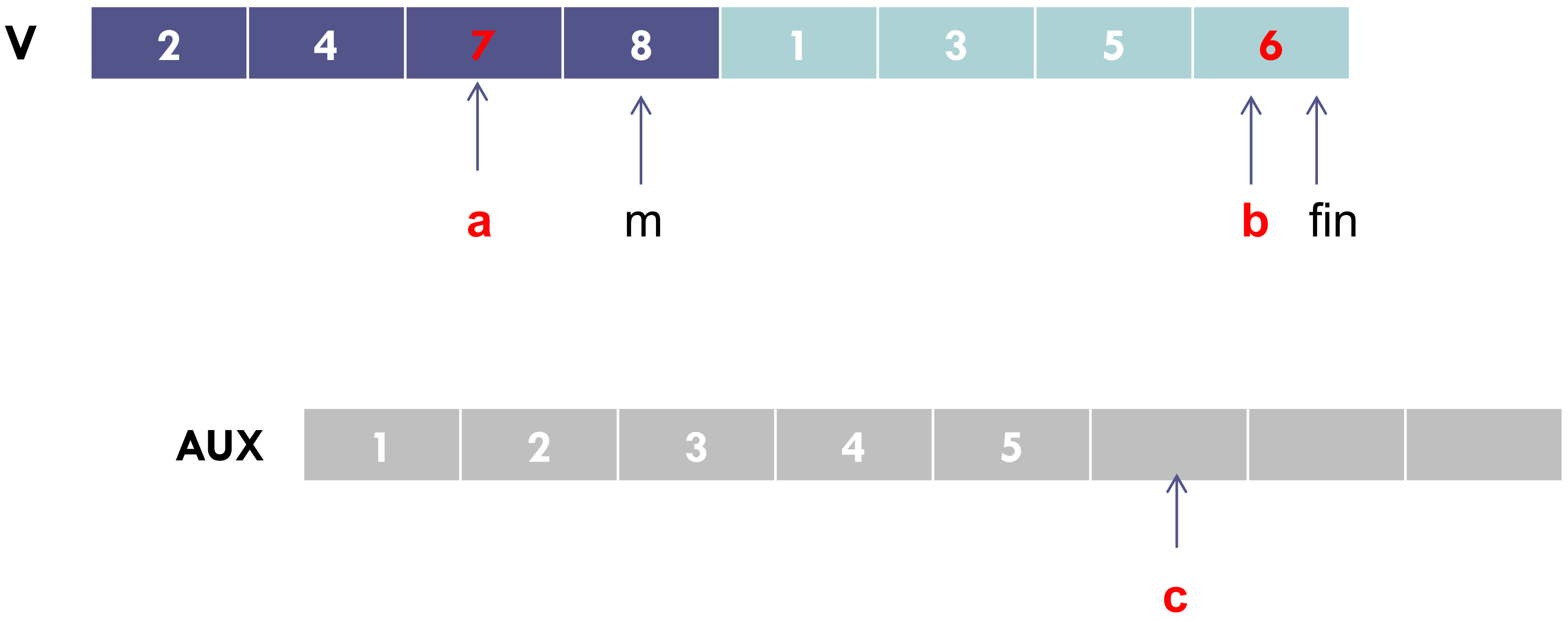


EJEMPLO DE FUNCIONAMIENTO DE MEZCLA

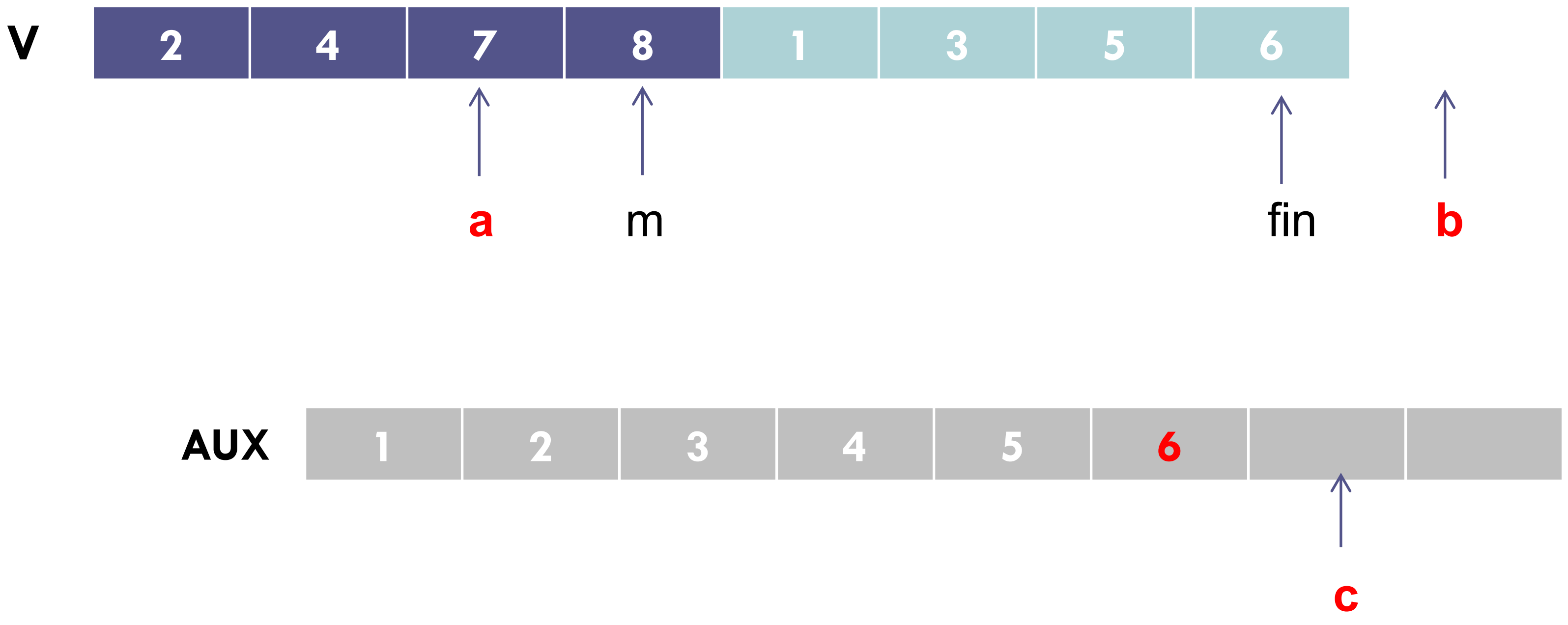


EJEMPLO DE FUNCIONAMIENTO DE MEZCLA

Paso 6.- V[a] > V[b] ⇒ en AUX introduciremos V[b], incrementaremos b y c



EJEMPLO DE FUNCIONAMIENTO DE MEZCLA



EJEMPLO DE FUNCIONAMIENTO DE MEZCLA

Paso 7.- Dado que la 2ª mitad del vector ha finalizado, se escribirá en AUX lo que queda de la 1ª mitad del vector.

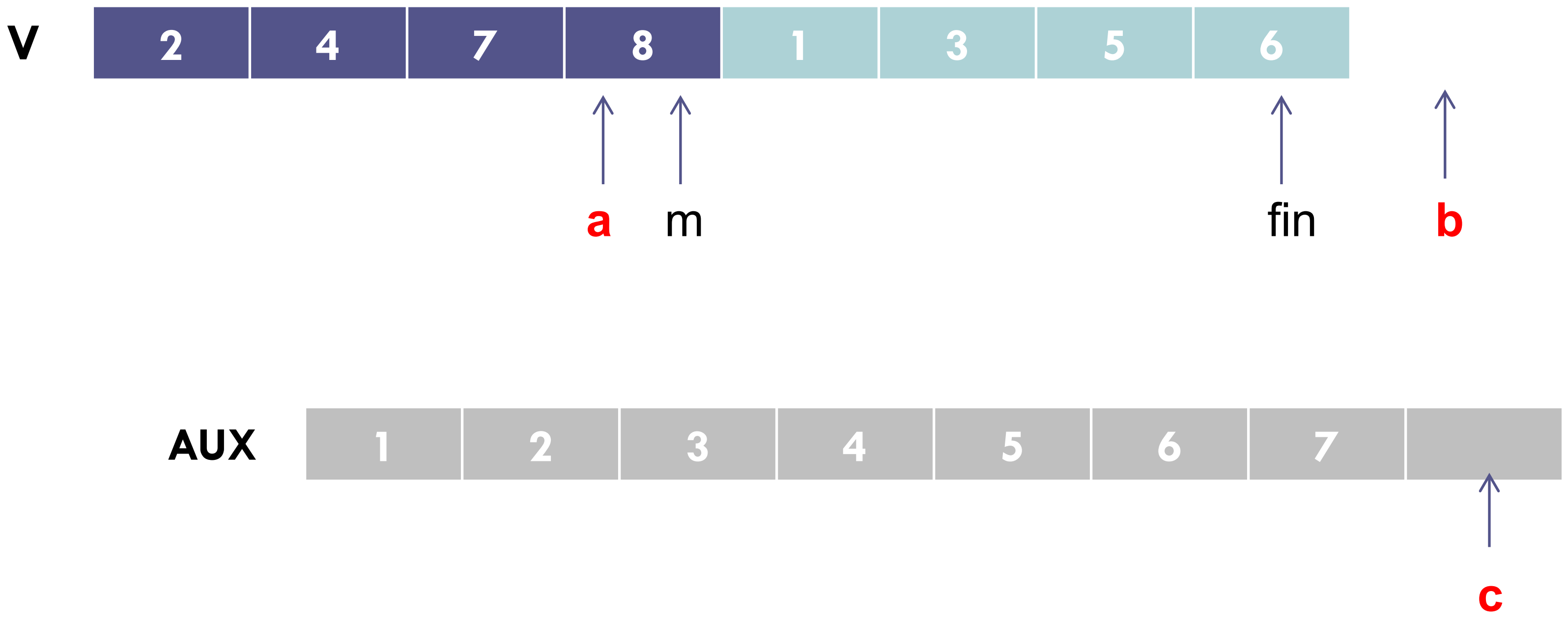
V	2	4	7	8	1	3	5	6

a m fin b

AUX	1	2	3	4	5	6		

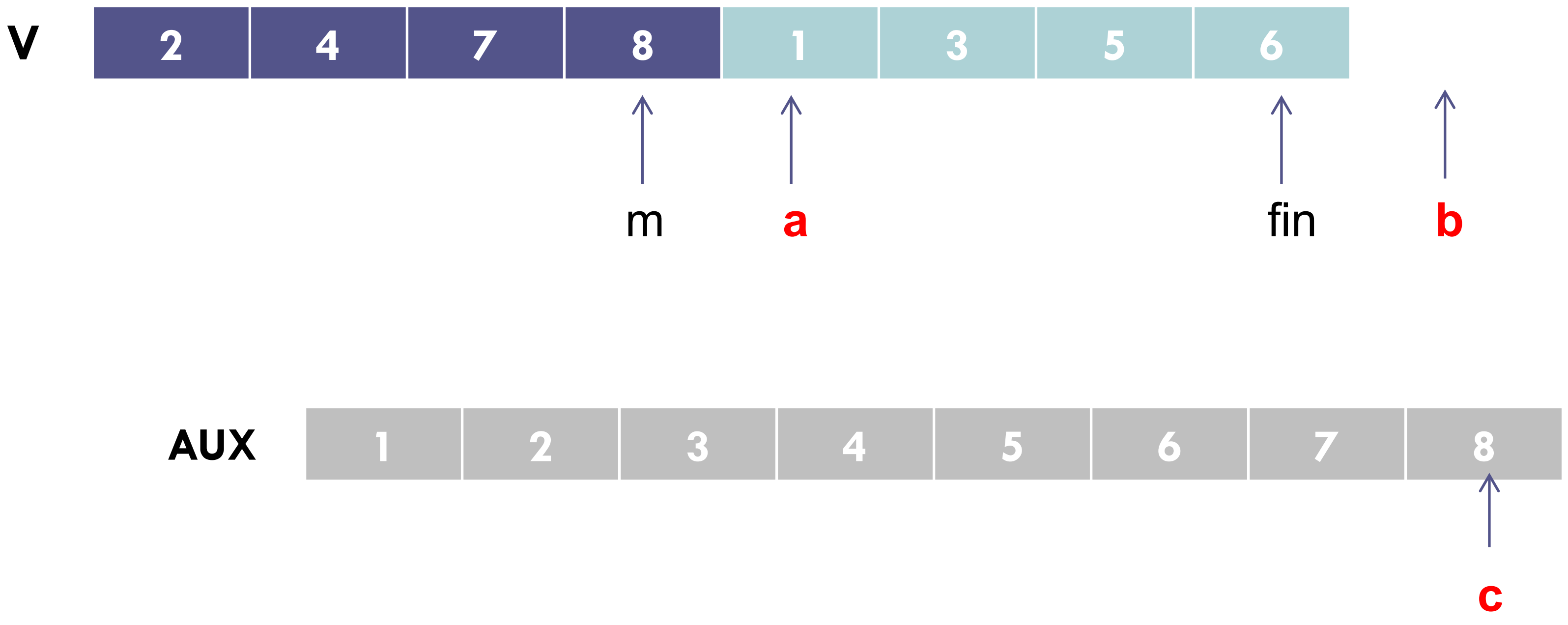
c

EJEMPLO DE FUNCIONAMIENTO DE MEZCLA



EJEMPLO DE FUNCIONAMIENTO DE MEZCLA

Paso 8.-



QUICKSORT

Dado un vector V[1..n] de n números enteros, siendo n≥0, el problema consiste en ordenar el vector en sentido creciente.

- Estrategia DyV.-

  - **Seleccionar** un elemento pivote y **dividir** el vector de tal modo que los elementos menores que el pivote queden a un lado y los mayores al otro. El pivote se situará en el medio
  - **Resolver** los dos subproblemas que corresponden a las dos secciones del vector: donde están los menores que el pivote y donde están los mayores que el pivote
  - **Combinar** : No habría que hacer nada

QUICKSORT

V							
50	31	75	92	43	81	13	57

QUICKSORT

V

50	31	75	92	43	81	13	57

Seleccionar pivote (por ej. V[1]=50), partir (situar los elementos menores que el pivote a la izquierda y los mayores que el pivote a la derecha, situar el pivote en su posición definitiva y devolver dicha posición, **índice_pivote**)

QUICKSORT

V	50	31	75	92	43	81	13	57

Seleccionar pivote (por ej. V[1]=50), partir (situar los elementos menores que el pivote a la izquierda y los mayores que el pivote a la derecha, situar el pivote en su posición definitiva y devolver dicha posición, **índice_pivote**)

V	43	31	13	50	92	81	75	57

QUICKSORT

V	50	31	75	92	43	81	13	57

Seleccionar pivote (por ej. V[1]=50), partir (situar los elementos menores que el pivote a la izquierda y los mayores que el pivote a la derecha, situar el pivote en su posición definitiva y devolver dicha posición, **indice_pivote**)

V	43	31	13	50	92	81	75	57

llamada a Quicksort con elementos menores que el pivote

llamada a Quicksort con elementos mayores que el pivote

V	43	31	13		50		92	81	75	57

indice_pivote representa la posición que el pivote elegido ocupa entre los menores y los mayores que él. Será su posición definitiva en el vector ordenado. En este caso, 4.

QUICKSORT


```
Procedimiento QuickSort ( V[1..n] : vector de enteros; inicio, fin : entero)
        si  inicio < fin entonces
                indice_pivote=Partir( V, inicio, fin )         /* dividir */
                QuickSort ( V, inicio, indice_pivote-1 )       /* resolver */
                QuickSort ( V, indice_pivote+1, fin )          /* resolver */
        fsi
fprocedimiento
```

donde la llamada inicial a la función es: **QuickSort(V, 1, n)**

$$T(n) \in \Omega(n\log_2 n) \text{ y } T(n) \in O(n^2)$$

ESTRATEGIAS DE SELECCIÓN PIVOTE EN QUICKSORT

- La selección del pivote debe minimizar la posibilidad de obtener una partición desequilibrada.

- Diferentes estrategias de elección del pivote:
  - Seleccionar el primer elemento del vector:
    - Mala elección si el vector está ordenado (pivote = minimo(vector))
  - Elegir el elemento central:
    - Perfecto si el vector ya está ordenado. Más que elegir un buen pivote, evita elegir uno malo.
  - Partición con la mediana del vector.
    - Mediana de un grupo N de elementos: El N/2-ésimo menor elemento.
    - Esta es la elección perfecta para cualquier vector de entrada

QUICKSORT vs MERGESORT

- Tanto QuickSort como MergeSort resuelven de manera recursiva dos subproblemas.
- QuickSort NO garantiza que el tamaño de los subproblemas sea el mismo (MergeSort sí).
- QuickSort es más rápido porque el paso de partición puede hacerse más rápido que el paso de fusión en MergeSort.
- QuickSort no requiere un vector auxiliar.
- QuickSort es el mejor algoritmo para la ordenación de vectores de gran dimensión.

TAREAS

- En la sesión de prácticas, el alumno completará el código que se le facilita, incorporando:
  - Función que proporcione el máximo elemento de un vector de enteros, dicha función debe seguir la metodología de Divide y Vencerás
  - Función que proporcione el producto de los elementos de un vector de enteros, dicha función debe seguir la metodología de Divide y Vencerás
  - [Opcional] Función que resuelve el ejercicio 1 del examen del tema 2 realizado el 10/11/2021 siguiendo la metodología de Divide y Vencerás
  - Entregar el fichero .c a través del Campus Virtual al finalizar la sesión de prácticas.
- Al alumno se le proporcionan las implementaciones de los algoritmos vistos a lo largo del presente documento para su trabajo autónomo: Potencia n-ésima de a, Búsqueda binaria, Mergesort y Quicksort