Ordenamiento

Ordenar una estructura de datos consiste en reacomodar sus elementos de manera tal que queden ordenados de acuerdo a un atributo **clave**.

Los algoritmos de ordenamiento resultan un tema de interés por varios motivos:

- Son importantes en diversas aplicaciones, en particular en el área de Bases de Datos, en donde los requerimientos de eficiencia hacen del ordenamiento un tema crítico.
- Existen muchísimos métodos para resolver el mismo problema y por lo tanto es un tema interesante para introducir nociones de tiempo de ejecución y eficiencia.
- Permiten ilustrar temas importantes de Resolución de Problemas.

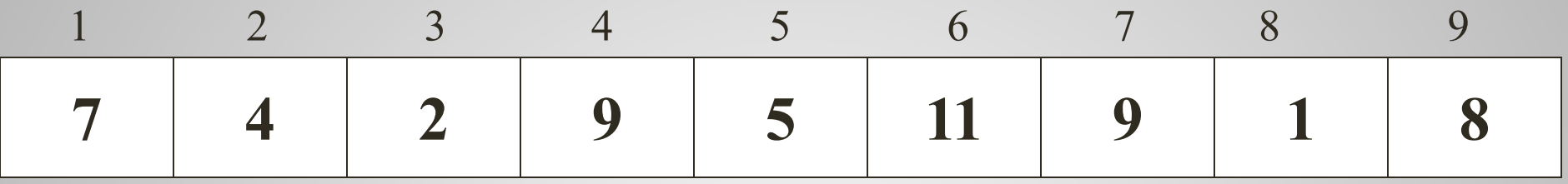
Merge Sort

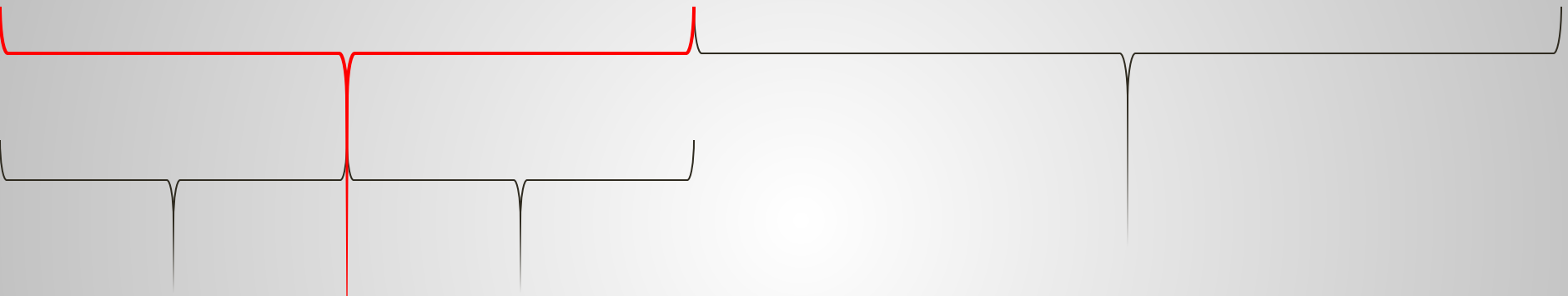
El método Merge Sort consiste en partir una estructura en mitades, ordenar cada mitad y luego intercalar ordenadamente ambas mitades.

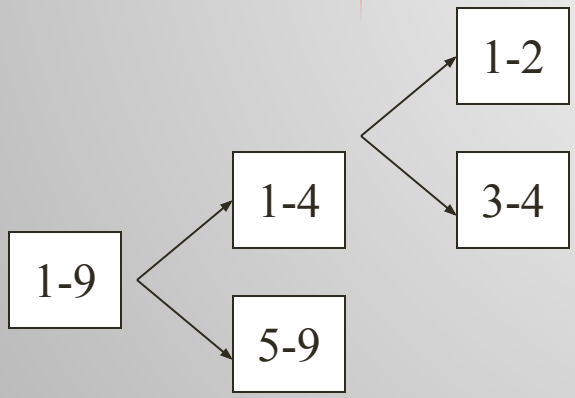
Cada mitad se ordena aplicando el mismo método.

- Dividir en “mitades”
  - Ordenar la primera mitad
  - Ordenar la segunda mitad
  - Intercalar las mitades ordenadas

Merge Sort







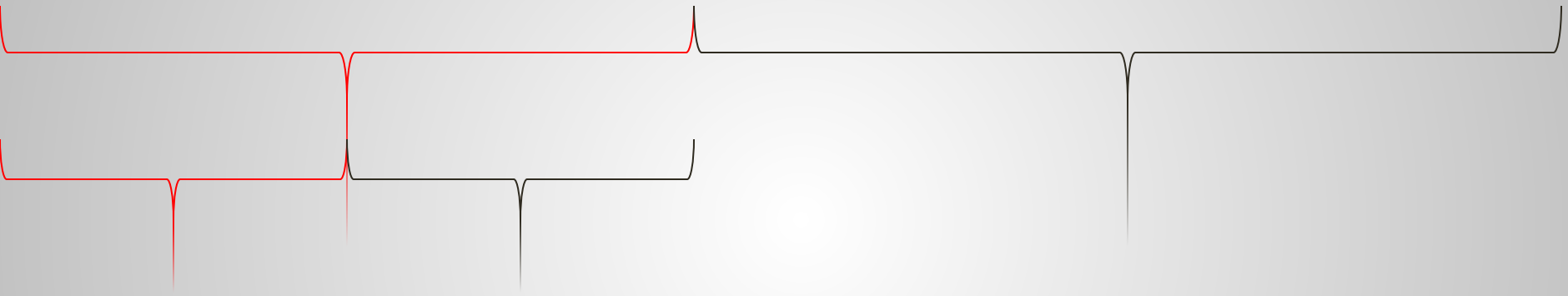
Merge Sort

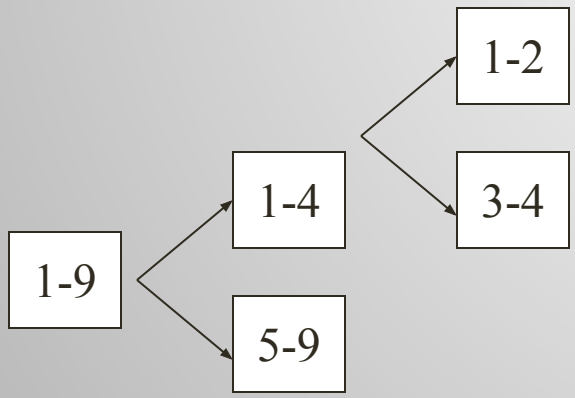
si la estructura tiene más de 2 elementos
    Dividir en “mitades”
    Ordenar la primera mitad
    Ordenar la segunda mitad
    Intercalar las mitades ordenadas
sino
    si la cantidad de componentes es 2
        Comparar e intercambiar

Observemos que este algoritmo NO depende:
- Del lenguaje de programación
- Del tipo de componentes

Merge Sort

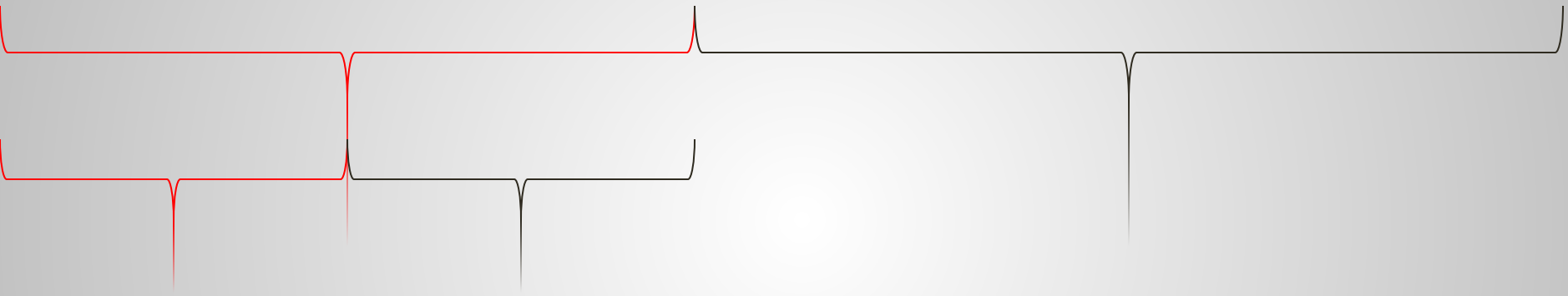
7 > 4

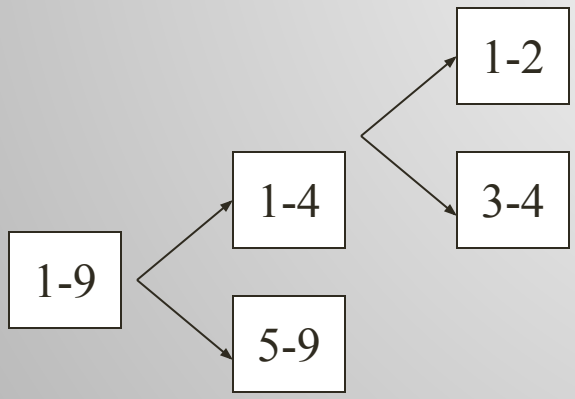




Merge Sort

1	2	3	4	5	6	7	8	9
4	**7**	**2**	**9**	**5**	**11**	**9**	**1**	**8**

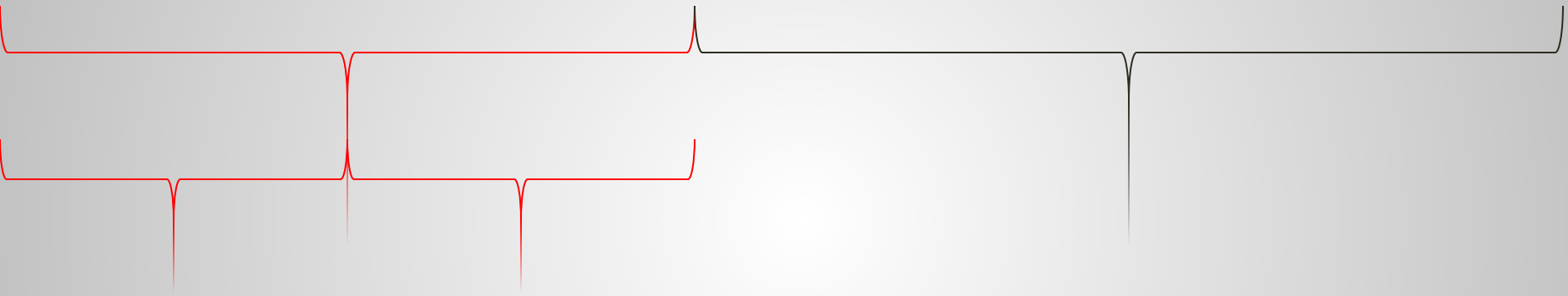


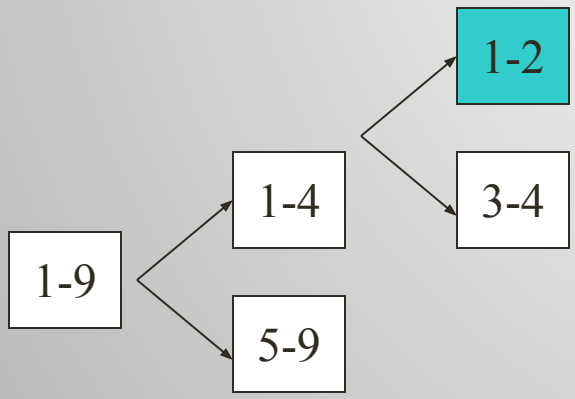


Merge Sort

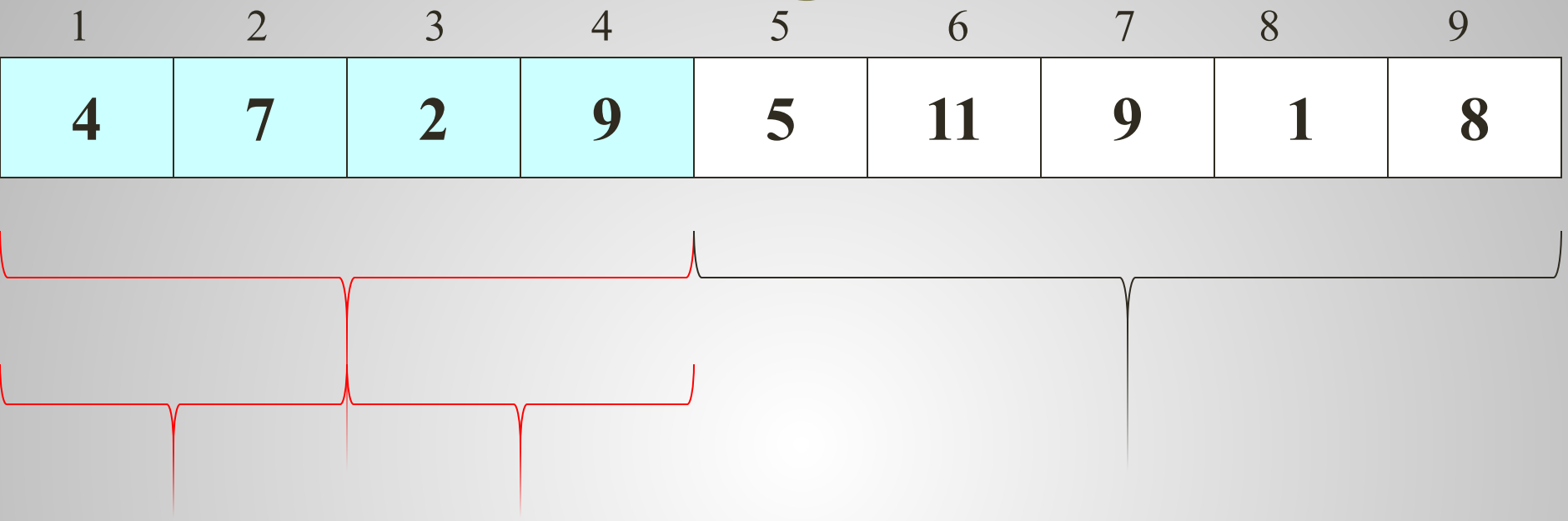
2 < 9

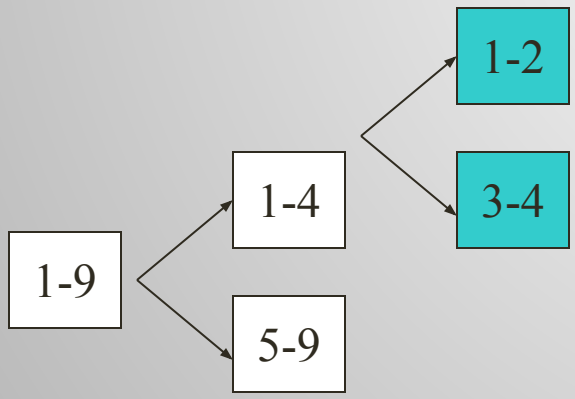
4	7	2	9	5	11	9	1	8





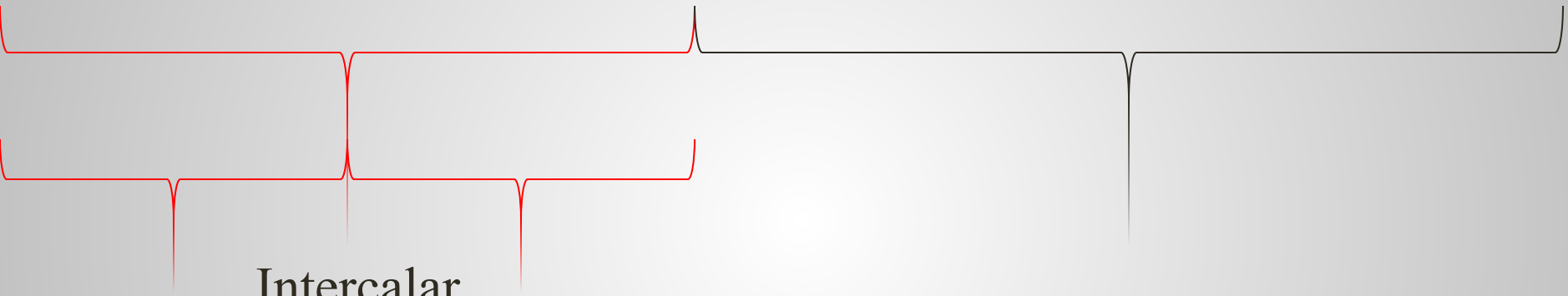
Merge Sort



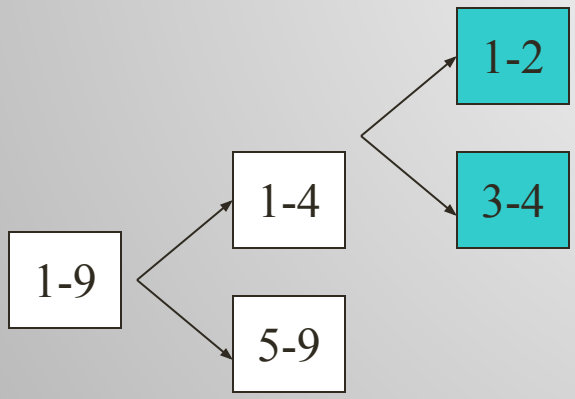


Merge Sort

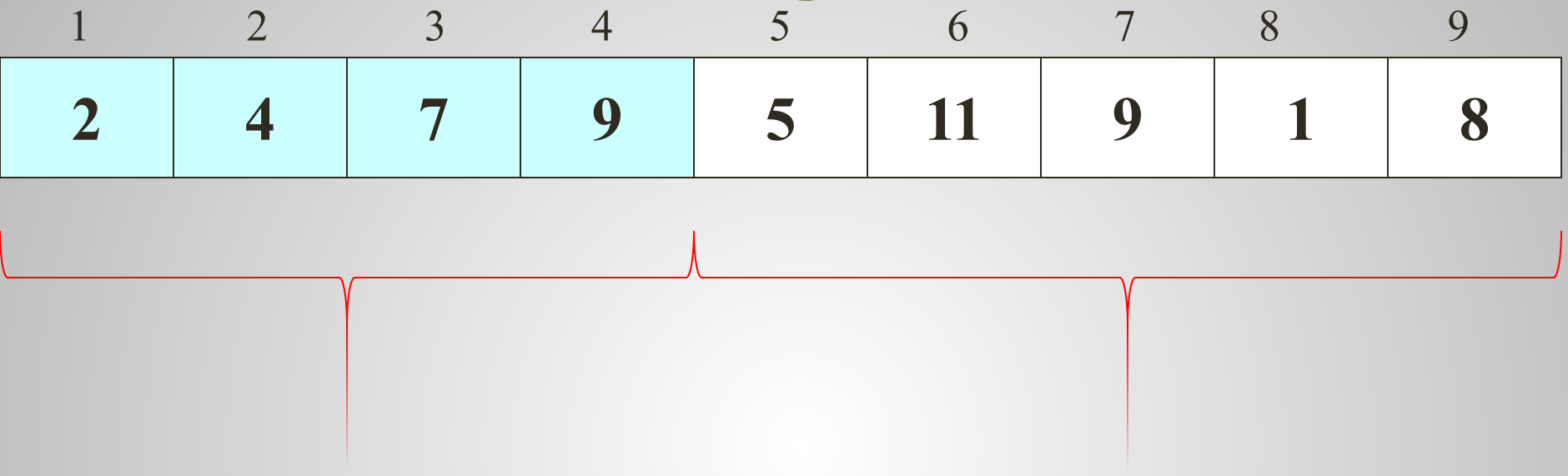
1	2	3	4	5	6	7	8	9
4	**7**	**2**	**9**	**5**	**11**	**9**	**1**	**8**

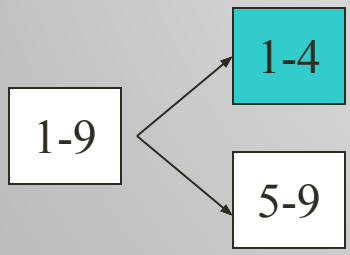


Intercalar



Merge Sort

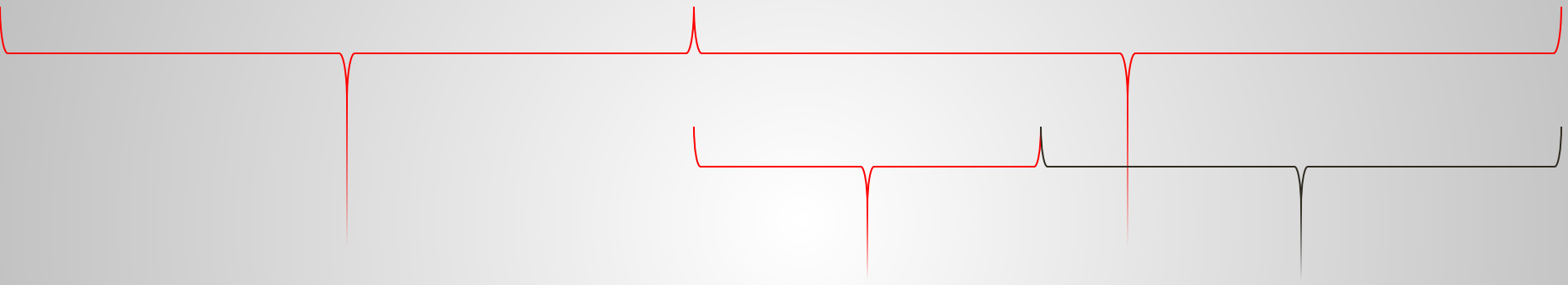


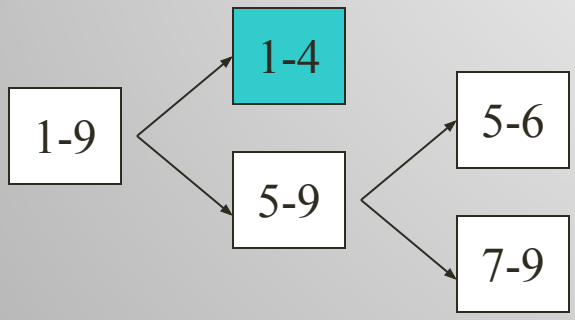


Merge Sort

5 < 11

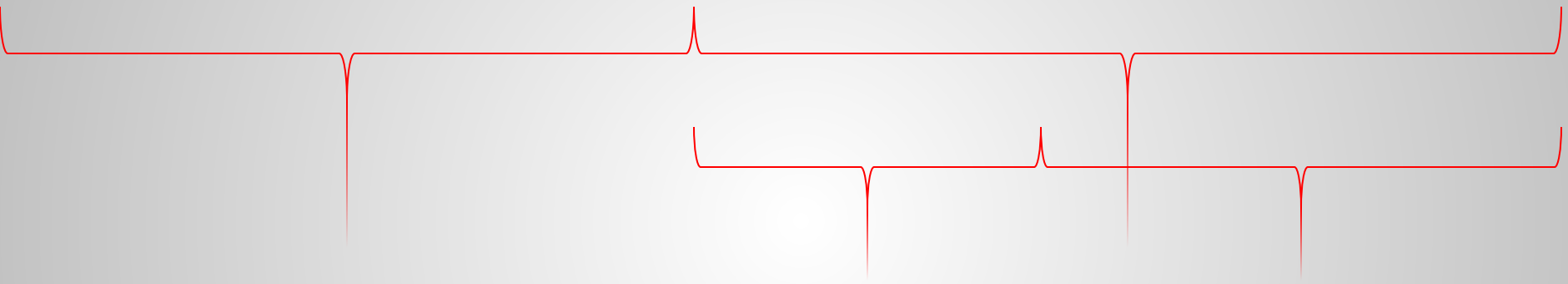
2	4	7	9	5	11	9	1	8

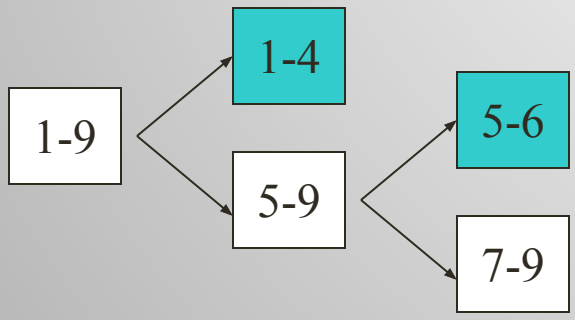




Merge Sort

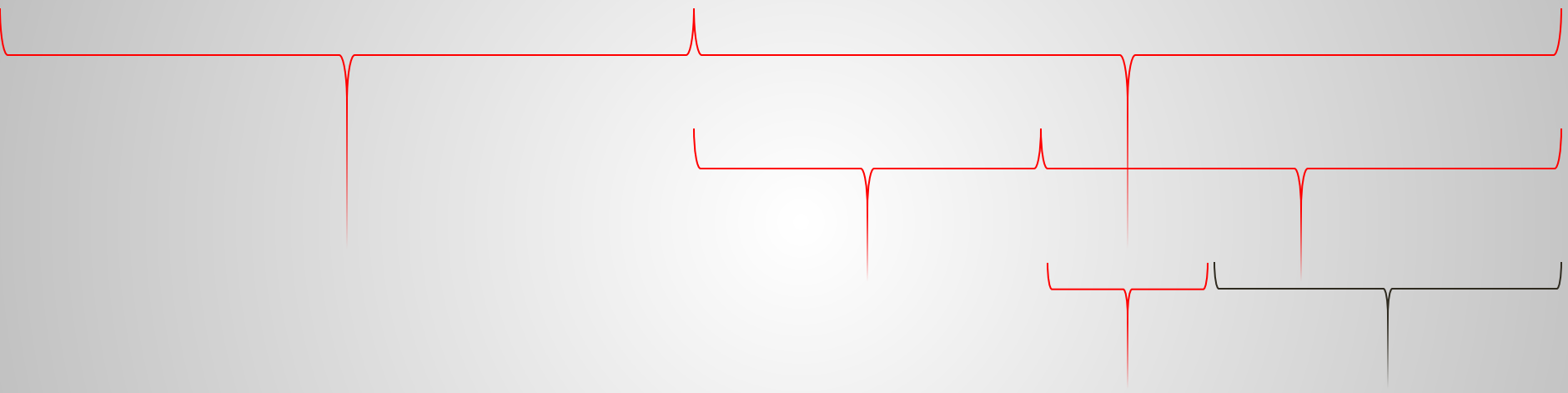
1	2	3	4	5	6	7	8	9
2	**4**	**7**	**9**	**5**	**11**	**9**	**1**	**8**

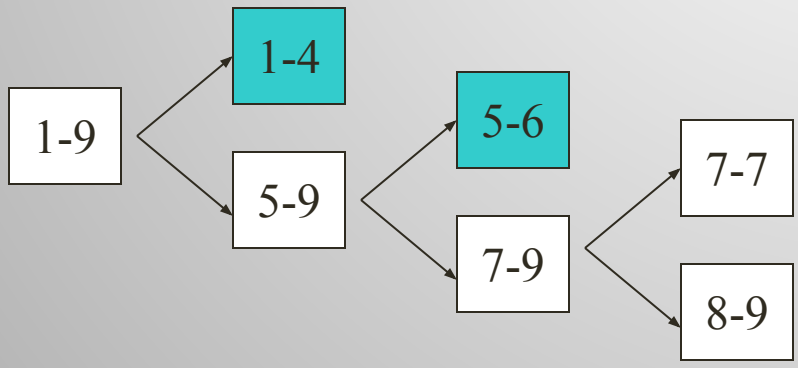




Merge Sort

1	2	3	4	5	6	7	8	9
2	**4**	**7**	**9**	**5**	**11**	**9**	**1**	**8**

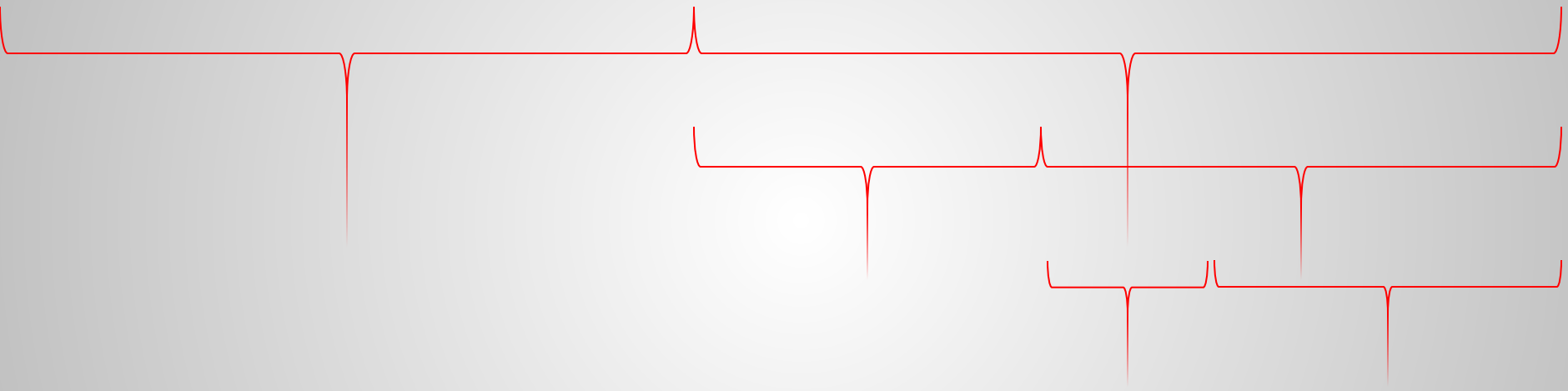


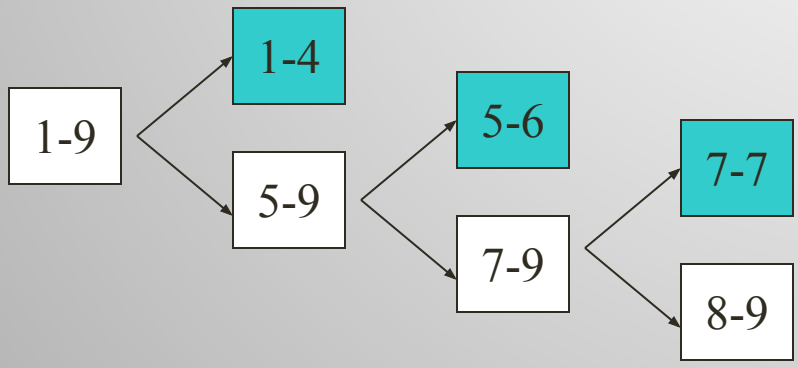


Merge Sort

1 < 8

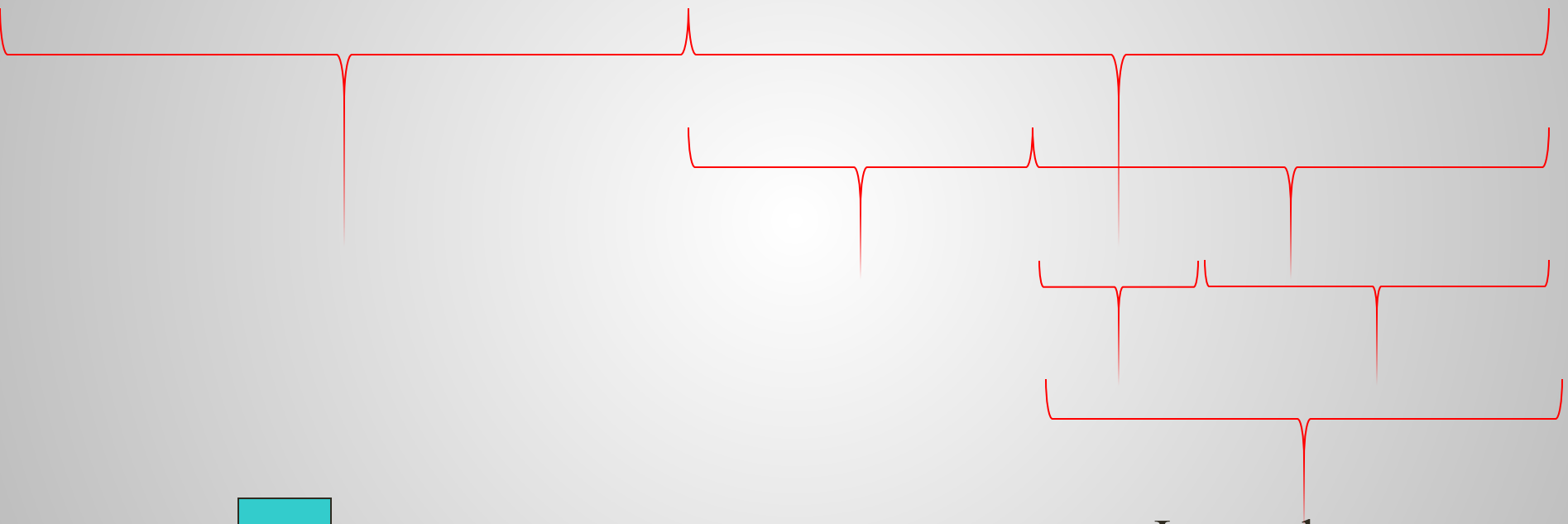
2	4	7	9	5	11	9	1	8



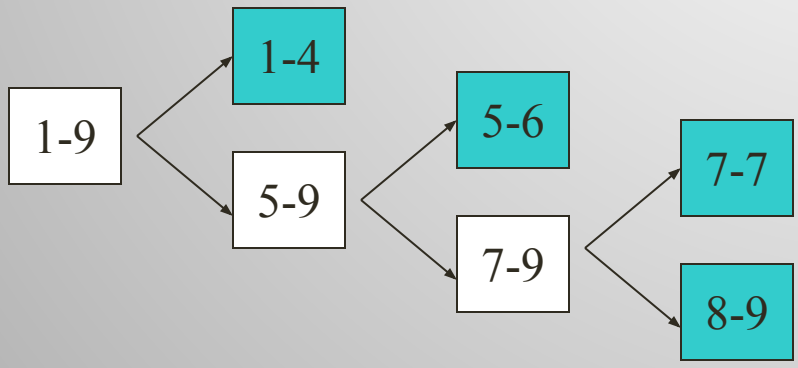


Merge Sort

1	2	3	4	5	6	7	8	9
2	**4**	**7**	**9**	**5**	**11**	**9**	**1**	**8**

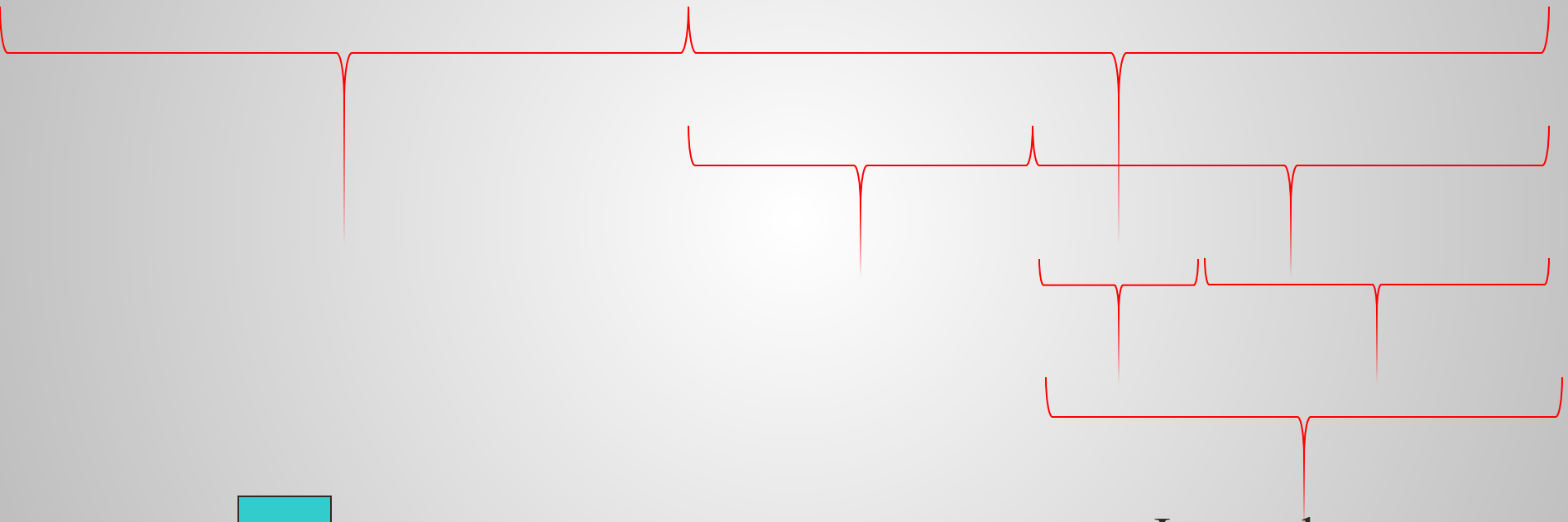


Intercalar

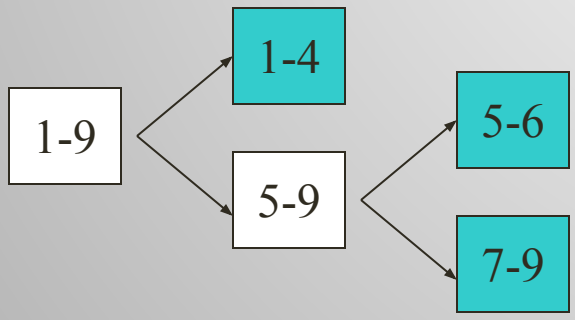


Merge Sort

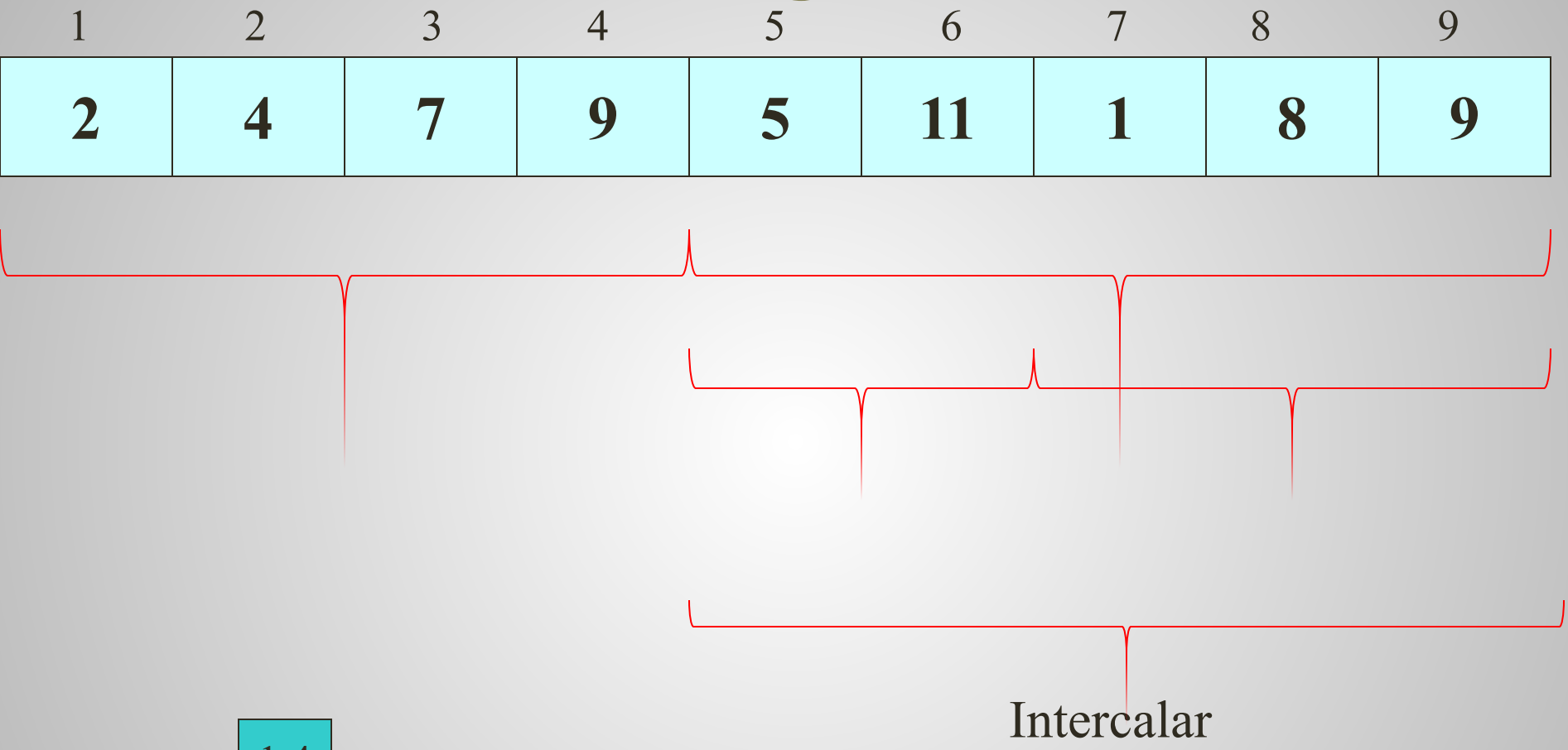
1	2	3	4	5	6	7	8	9
2	**4**	**7**	**9**	**5**	**11**	**1**	**8**	**9**

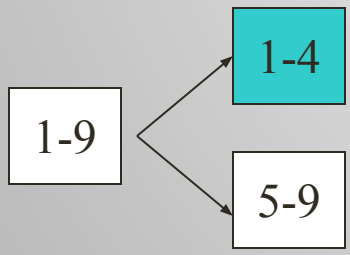


Intercalar



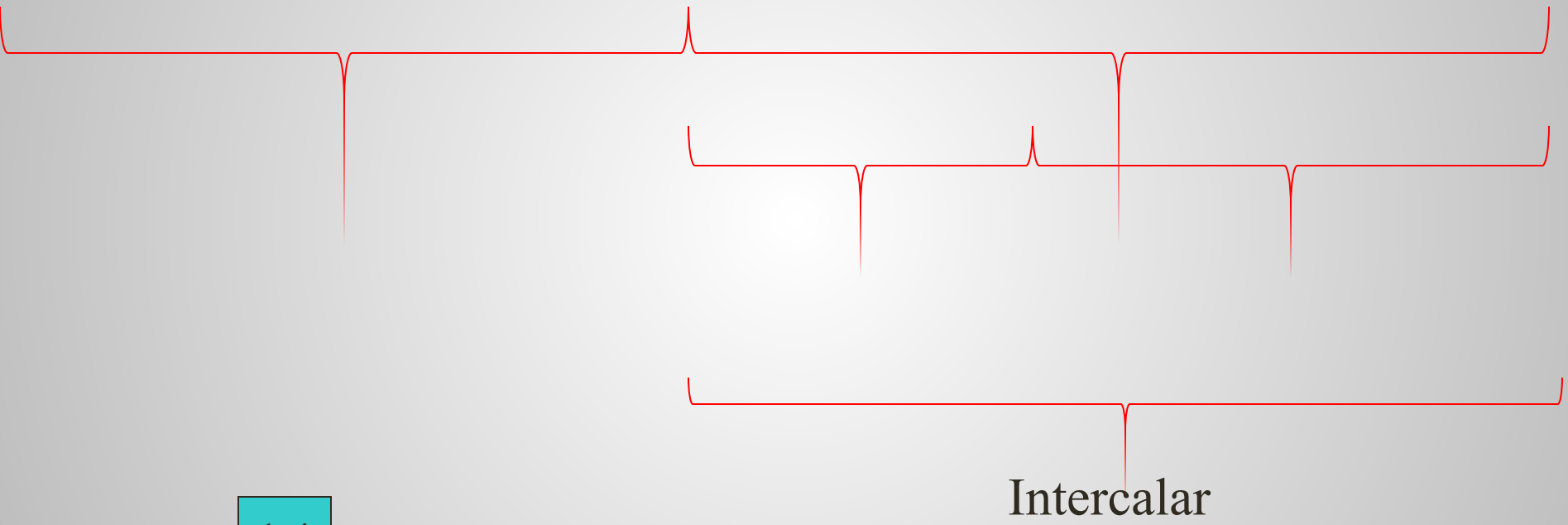
Merge Sort

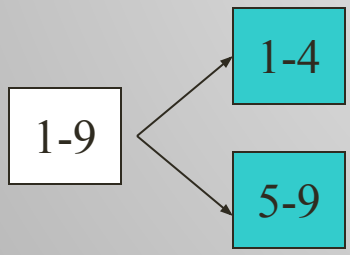




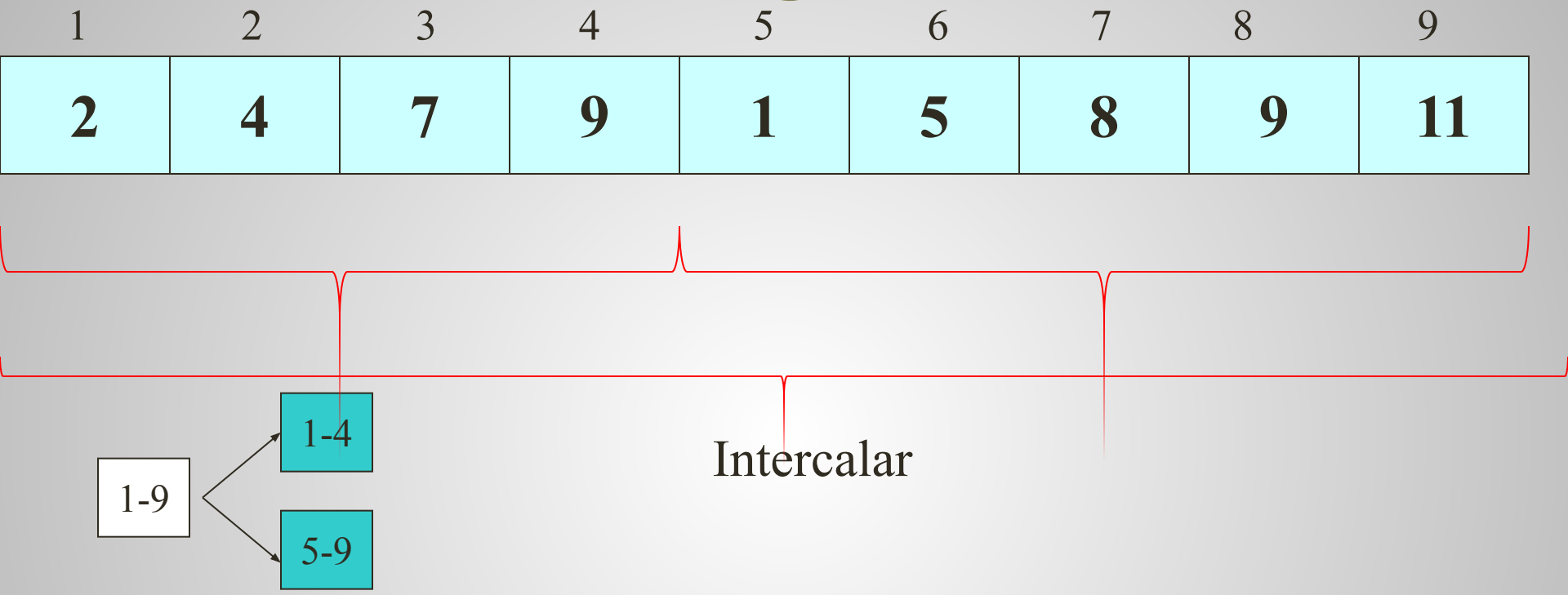
Merge Sort

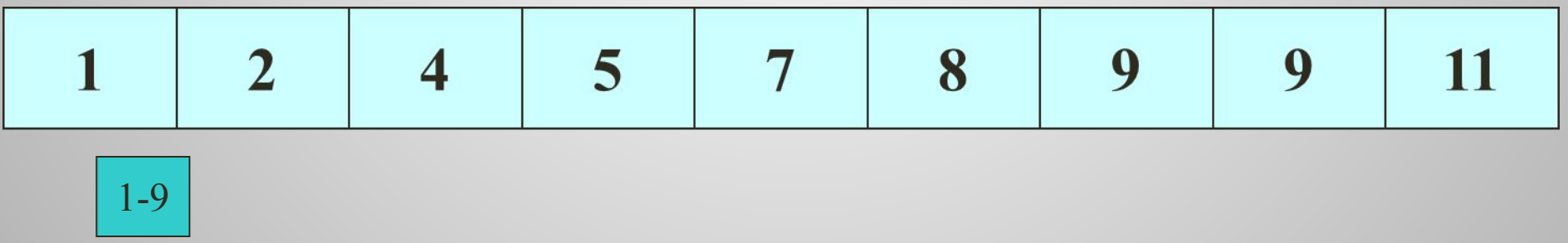
1	2	3	4	5	6	7	8	9
2	**4**	**7**	**9**	**1**	**5**	**8**	**9**	**11**





Merge Sort





Merge Sort

Algoritmo MergeSort

si la cantidad de componentes es menor o igual a 2
    si la cantidad de componentes es igual a 2
        Comparar Intercambiar
sino
    Dividir en mitades
    MergeSort primera mitad
    MergeSort segunda mitad
    Intercalar las mitades ordenadas

Merge Sort

Algoritmo MergeSort
DE ini,fin

si ini+1 = fin
    si T_{ini} es mayor que T_{fin}
        intercambiar T_{ini} , T_{fin}
sino
    si ini < fin
        Mitad = (ini+fin) / 2 *(redondeo)*
        MergeSort ini,Mitad-1
        MergeSort Mitad,fin
        Intercalar ini,Mitad-1,Mitad,fin

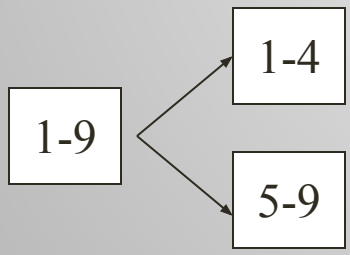
Merge Sort

1	2	3	4	5	6	7	8	9
7	**4**	**2**	**9**	**5**	**11**	**9**	**1**	**8**

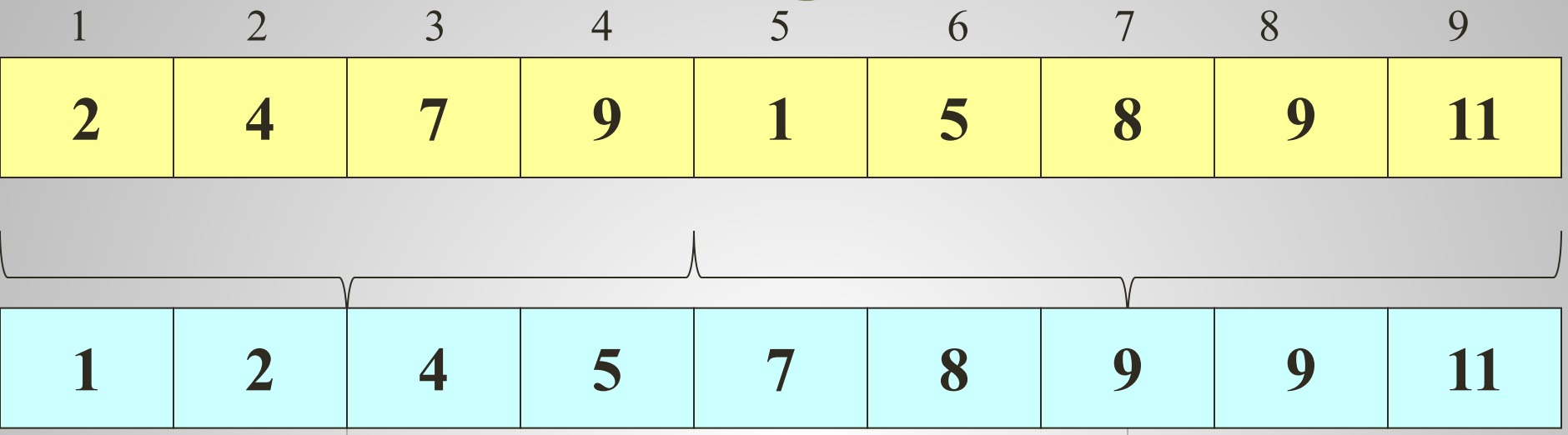
MergeSort (1,9)

MergeSort (1,4)

MergeSort (5,9)



Merge Sort

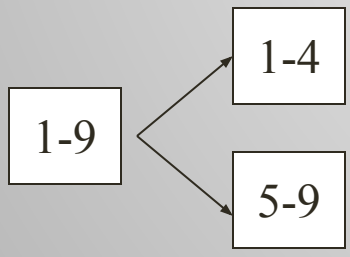


MergeSort (1,9)

MergeSort (1,4)

MergeSort (5,9)

Intercalar (1,4,5,9)



Merge Sort

El método intercalar se implementa utilizando la misma estrategia que hemos propuesto antes para intercalar ***dos*** estructuras ordenadas, solo que ahora se intercalan las dos mitades de ***una*** estructura.

Recordemos que al intercalar dos estructuras se genera una tercera en la cual los elementos se agregan al final.

En la implementación en Java este método es privado, solo es accesible dentro de la clase.

Merge Sort

Algoritmo Intercalar
DE i1,n1,i2,n2

crear aux con n2-i1+1 elementos
Ini=i1 *//para recordar donde empiezo*
mientras i1 <= n1 y i2 <= n2
    si T_{i1} es menor que T_{i2}
        agregar T_{i1} al final de aux
        i1++
    sino
        agregar T_{i2} al final de aux
        i2++
    …

Merge Sort

Algoritmo Intercalar *(continuación)*

…

mientras i1 <= n1
  agregar T_{i1} al final de aux
  i1++

mientras i2 <= n2
  agregar T_{i2} al final de aux
  i2++

  //copiar en el arreglo original

i1=ini-1
para i entre 1 y n2-i1
  $T_{i1+i} = aux_i$

Merge Sort

1	2	3	4	5	6	7	8	9
8	4	7	9	5	11	1	8	3

Algoritmo MergeSort
DE ini,fin

si ini+1 = fin
  si T_{ini} es mayor que T_{fin}
    intercambiarT_{ini} , T_{fin}
sino si ini < fin
  Mitad = (ini+fin) / 2
  MergeSort ini,Mitad-1
  MergeSort Mitad,fin
  Intercalar ini,Mitad-1,Mitad,fin

ini	fin
1	9

MergeSort(1,9)

1-9

Merge Sort

1	2	3	4	5	6	7	8	9
8	4	7	9	5	11	1	8	3

Algoritmo MergeSort
DE ini,fin

si ini+1 = fin
  si T_{ini} es mayor que T_{fin}
    intercambiarT_{ini} , T_{fin}
sino si ini < fin
  Mitad = (ini+fin) / 2
  MergeSort ini,Mitad-1
  MergeSort Mitad,fin
  Intercalar ini,Mitad-1,Mitad,fin

ini	fin	mitad
1	9	5

MergeSort(1,9)

1-9

Merge Sort

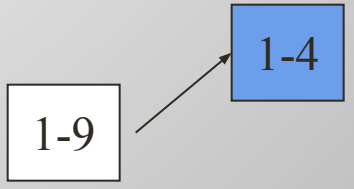
1	2	3	4	5	6	7	8	9
8	4	7	9	5	11	1	8	3

Algoritmo MergeSort
DE ini,fin

si ini+1 = fin
  si T_{ini} es mayor que T_{fin}
    intercambiarT_{ini} , T_{fin}
sino si ini < fin
  Mitad = (ini+fin) / 2
  MergeSort ini,Mitad-1
  MergeSort Mitad,fin
  Intercalar ini,Mitad-1,Mitad,fin

ini	fin	mitad
1	9	5
1	4	

MergeSort(1,9)



Merge Sort

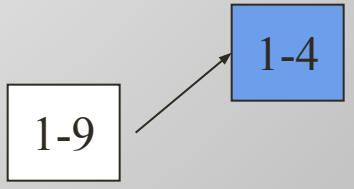
1	2	3	4	5	6	7	8	9
8	4	7	9	5	11	1	8	3

Algoritmo MergeSort
DE ini,fin

si ini+1 = fin
  si T_{ini} es mayor que T_{fin}
    intercambiarT_{ini} , T_{fin}
sino si ini < fin
  Mitad = (ini+fin) / 2
  MergeSort ini,Mitad-1
  MergeSort Mitad,fin
  Intercalar ini,Mitad-1,Mitad,fin

ini	fin	mitad
1	9	5
1	4	3

MergeSort(1,4)



Merge Sort

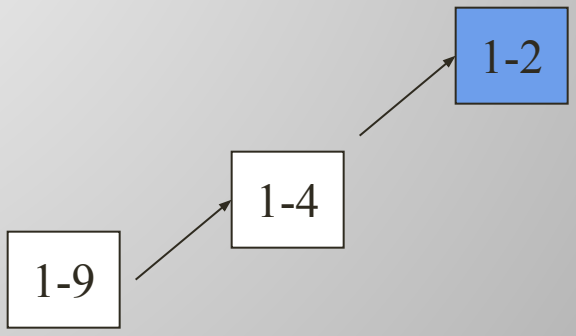
1	2	3	4	5	6	7	8	9
8	4	7	9	5	11	1	8	3

Algoritmo MergeSort
DE ini,fin

si ini+1 = fin
  si T_{ini} es mayor que T_{fin}
    intercambiarT_{ini} , T_{fin}
sino si ini < fin
  Mitad = (ini+fin) / 2
  MergeSort ini,Mitad-1
  MergeSort Mitad,fin
  Intercalar ini,Mitad-1,Mitad,fin

ini	fin	mitad
1	9	5
1	4	3
1	2	

MergeSort(1,4)



Merge Sort

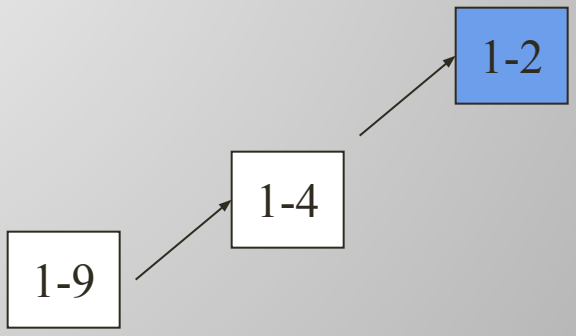
1	2	3	4	5	6	7	8	9
4	8	7	9	5	11	1	8	3

Algoritmo MergeSort
DE ini,fin

si ini+1 = fin
  si T_{ini} es mayor que T_{fin}
    intercambiarT_{ini} , T_{fin}
sino si ini < fin
  Mitad = (ini+fin) / 2
  MergeSort ini,Mitad-1
  MergeSort Mitad,fin
  Intercalar ini,Mitad-1,Mitad,fin

ini	fin	mitad
1	9	5
1	4	3
1	2	

MergeSort(1,2)



Merge Sort

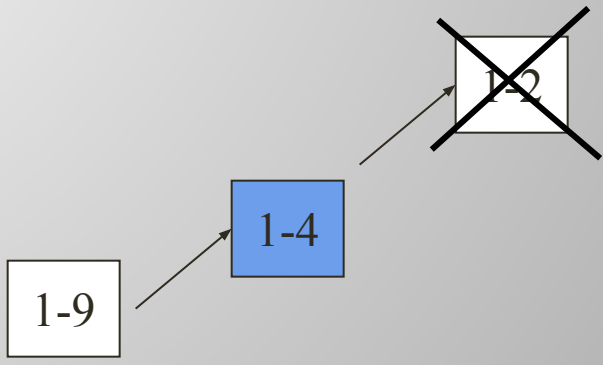
1	2	3	4	5	6	7	8	9
4	8	7	9	5	11	1	8	3

Algoritmo MergeSort
DE ini,fin

si ini+1 = fin
  si T_{ini} es mayor que T_{fin}
    intercambiarT_{ini} , T_{fin}
sino si ini < fin
  Mitad = (ini+fin) / 2
  MergeSort ini,Mitad-1
  MergeSort Mitad,fin
  Intercalar ini,Mitad-1,Mitad,fin

ini	fin	mitad
1	9	5
1	4	3

MergeSort(1,4)



Merge Sort

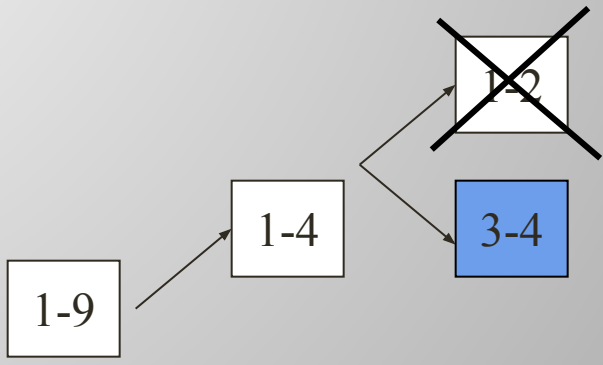
1	2	3	4	5	6	7	8	9
4	8	7	9	5	11	1	8	3

Algoritmo MergeSort
DE ini,fin

si ini+1 = fin
  si T_{ini} es mayor que T_{fin}
    intercambiarT_{ini} , T_{fin}
sino si ini < fin
  Mitad = (ini+fin) / 2
  MergeSort ini,Mitad-1
  MergeSort Mitad,fin
  Intercalar ini,Mitad-1,Mitad,fin

ini	fin	mitad
1	9	5
1	4	3
3	4	

MergeSort(3,4)



Merge Sort

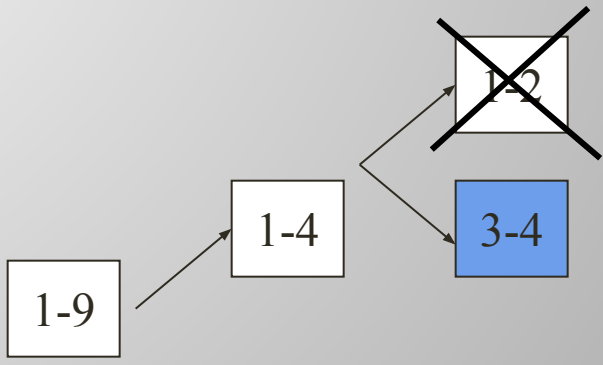
1	2	3	4	5	6	7	8	9
4	8	7	9	5	11	1	8	3

Algoritmo MergeSort
DE ini,fin

si ini+1 = fin
  si T_{ini} es mayor que T_{fin}
    intercambiarT_{ini} , T_{fin}
sino si ini < fin
  Mitad = (ini+fin) / 2
  MergeSort ini,Mitad-1
  MergeSort Mitad,fin
  Intercalar ini,Mitad-1,Mitad,fin

ini	fin	mitad
1	9	5
1	4	3
3	4	

MergeSort(3,4)



Merge Sort

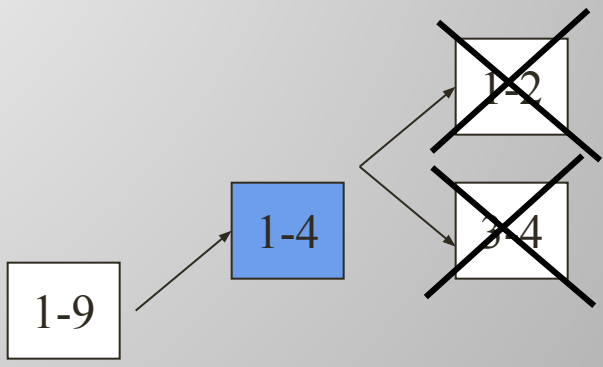
1	2	3	4	5	6	7	8	9
4	8	7	9	5	11	1	8	3

Algoritmo MergeSort
DE ini,fin

si ini+1 = fin
  si T_{ini} es mayor que T_{fin}
    intercambiarT_{ini} , T_{fin}
sino si ini < fin
  Mitad = (ini+fin) / 2
  MergeSort ini,Mitad-1
  MergeSort Mitad,fin
  Intercalar ini,Mitad-1,Mitad,fin

ini	fin	mitad
1	9	5
1	4	3

MergeSort(1,4)



Merge Sort

1	2	3	4	5	6	7	8	9
4	8	7	9	5	11	1	8	3

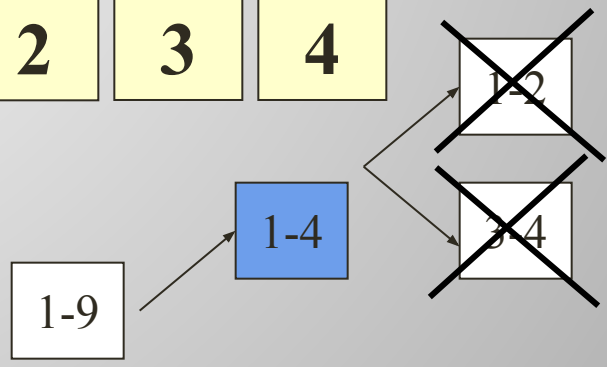
Algoritmo MergeSort
DE ini,fin

si ini+1 = fin
  si T_{ini} es mayor que T_{fin}
    intercambiarT_{ini} , T_{fin}
sino si ini < fin
  Mitad = (ini+fin) / 2
  MergeSort ini,Mitad-1
  MergeSort Mitad,fin
  Intercalar ini,Mitad-1,Mitad,fin

ini	fin	mitad
1	9	5
1	4	3

i1	n1	i2	n2
1	2	3	4

MergeSort(1,4)



Merge Sort

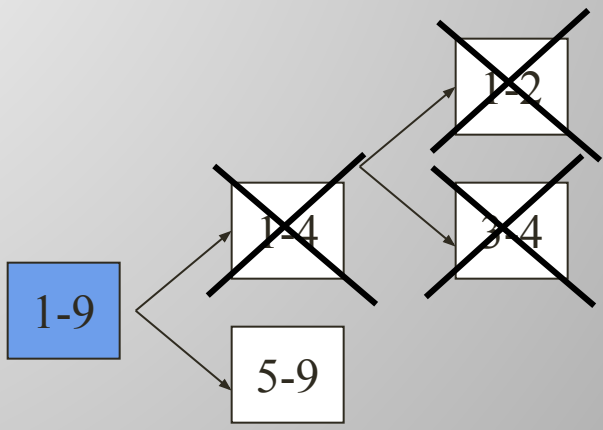
1	2	3	4	5	6	7	8	9
4	7	8	9	5	11	1	3	8

Algoritmo MergeSort
DE ini,fin

si ini+1 = fin
  si T_{ini} es mayor que T_{fin}
    intercambiarT_{ini} , T_{fin}
sino si ini < fin
  Mitad = (ini+fin) / 2
  MergeSort ini,Mitad-1
  MergeSort Mitad,fin
  Intercalar ini,Mitad-1,Mitad,fin

ini	fin	mitad
1	9	5

MergeSort(1,9)



Merge Sort

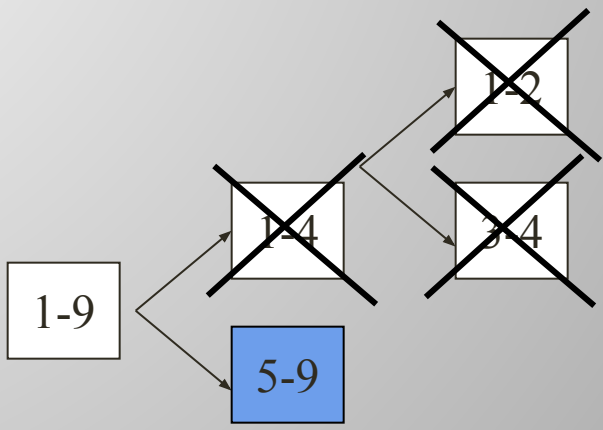
1	2	3	4	5	6	7	8	9
4	7	8	9	5	11	1	3	8

Algoritmo MergeSort
DE ini,fin

si ini+1 = fin
  si T_{ini} es mayor que T_{fin}
    intercambiarT_{ini} , T_{fin}
sino si ini < fin
  Mitad = (ini+fin) / 2
  MergeSort ini,Mitad-1
  MergeSort Mitad,fin
  Intercalar ini,Mitad-1,Mitad,fin

ini	fin	mitad
1	9	5
5	9	7

MergeSort(1,9)



Ordenamiento: Quick Sort

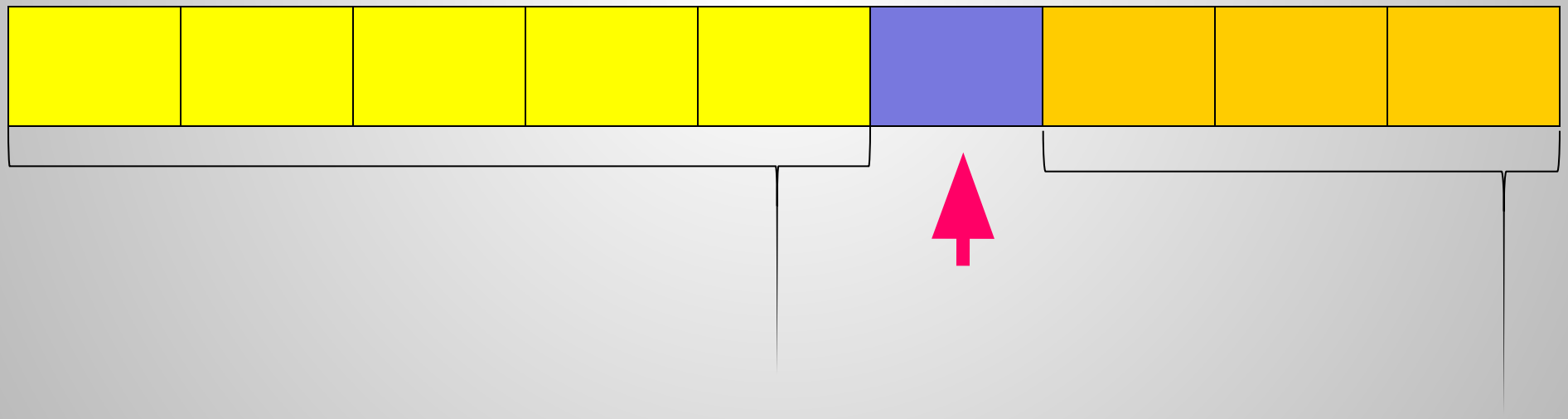
El método de Quick Sort consiste en acomodar un elemento llamado **Pivot** en su posición definitiva y luego ordenar la estructura que queda a su izquierda y la que queda a su derecha.

Todos los elementos en posiciones menores al Pivot son menores que él.

Todos los elementos en posiciones mayores al Pivot son mayores que él.

Las dos estructuras se ordenan aplicando el mismo método.

Ordenamiento: Quick Sort



Ordenamiento: Quick Sort

Acomodar un elemento llamado Pivot
Ordenar a la izquierda del Pivot
Ordenar a la derecha del Pivot

Ordenamiento: Quick Sort

Algoritmo QuickSort

si hay más de un elemento
    Acomodar Pivot
    QuickSort a la izquierda del Pivot
    QuickSort la derecha del Pivot

Ordenamiento: Quick Sort

```
Algoritmo QuickSort
DE ini,Fin
si ini < fin
  pospivot ← AcomodarPivot ini,fin
  QuickSort ini,pospivot-1
  QuickSort pospivot+1,fin
```

Es un algoritmo genérico, no depende del tipo de los elementos.

Ordenamiento: Quick Sort

```
Algoritmo AcomodarPivot
DE ini,fin
DS pos
   pos ← avanzar ini,fin
```

Ordenamiento: Quick Sort

```
Algoritmo avanzar
DE izq, der
DS posPiv
si izq >= der
    posPiv ← izq
sino
    si T izq >= T izq+1
        intercambiar izq izq+1
        posPiv ← avanzar izq+1,der
    sino
        posPiv ← retroceder izq,der
```

Ordenamiento: Quick Sort

```
Algoritmo retroceder
DE izq, der
DS posPiv
si izq >= der
    posPiv ← izq
sino
    si T izq <= T der
       posPiv ← retroceder izq,der-1
   sino
      intercambiar izq+1,der
      posPiv ← avanzar izq,der-1
```

Es un algoritmo genérico, no depende del tipo de los elementos.

TDA Matriz Racionales

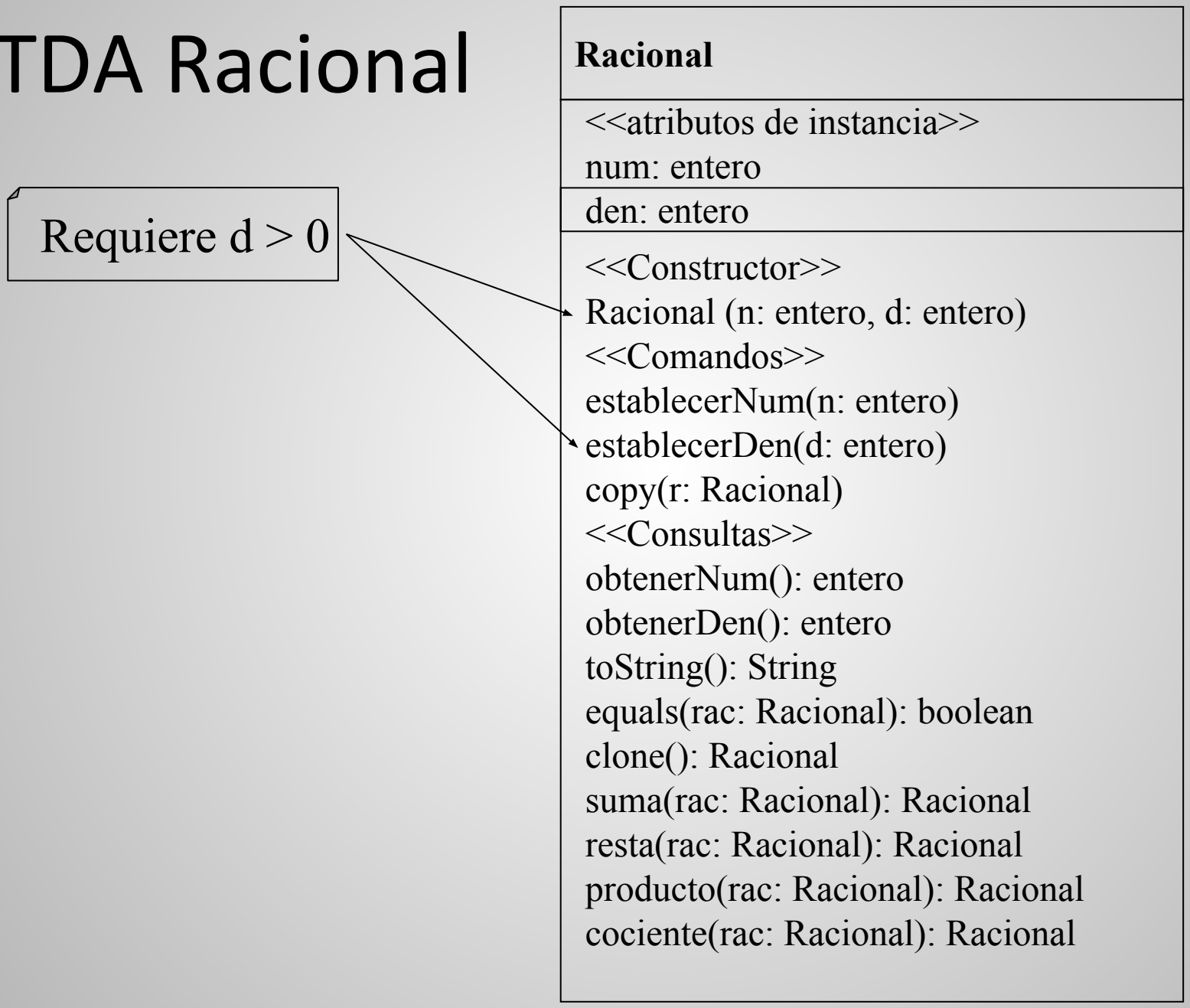
TDA Matriz Racionales

Implementar un TDA MatrizRac que brinde operaciones para calcular el producto de un escalar por una matriz, la suma de dos matrices, establecer la matriz identidad, decidir si una matriz es cuadrada, decidir si una matriz es la matriz identidad, decidir si es una matriz simétrica.

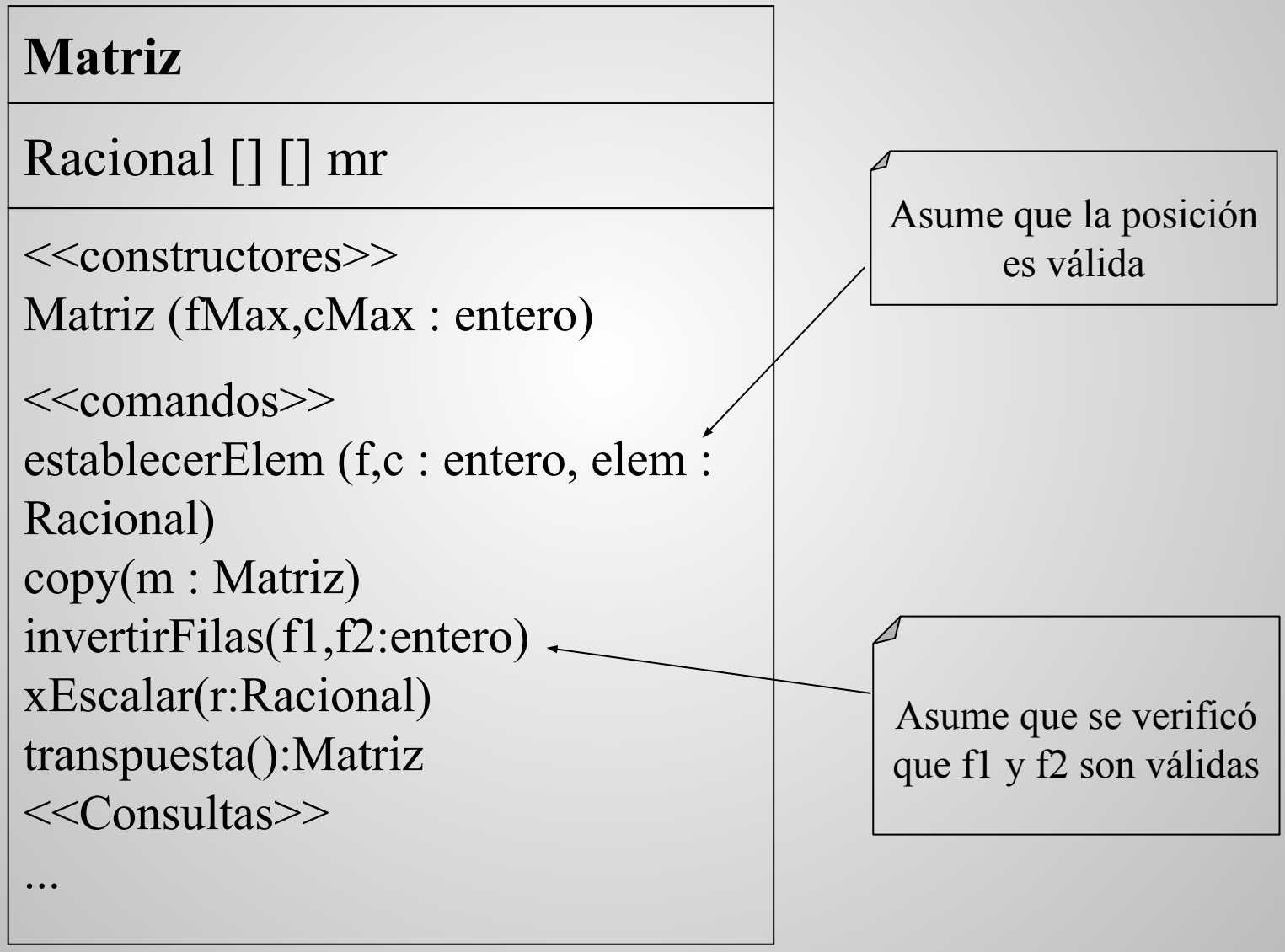
La matriz se representa mediante un arreglo de dos dimensiones de números racionales

La clase que encapsula al arreglo brinda operaciones para establecer y obtener un elemento y para comparar, copiar y clonar matrices.

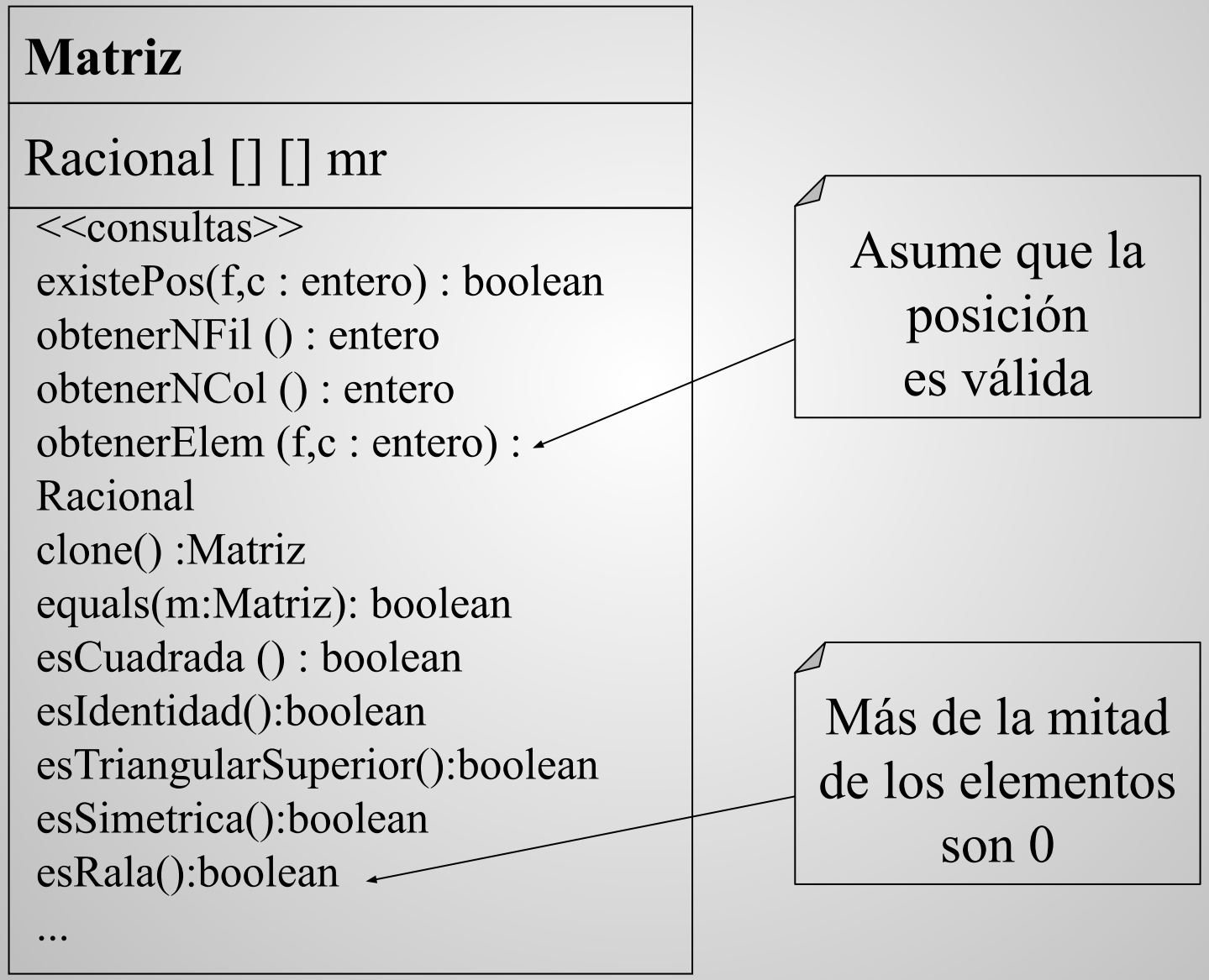
TDA Racional



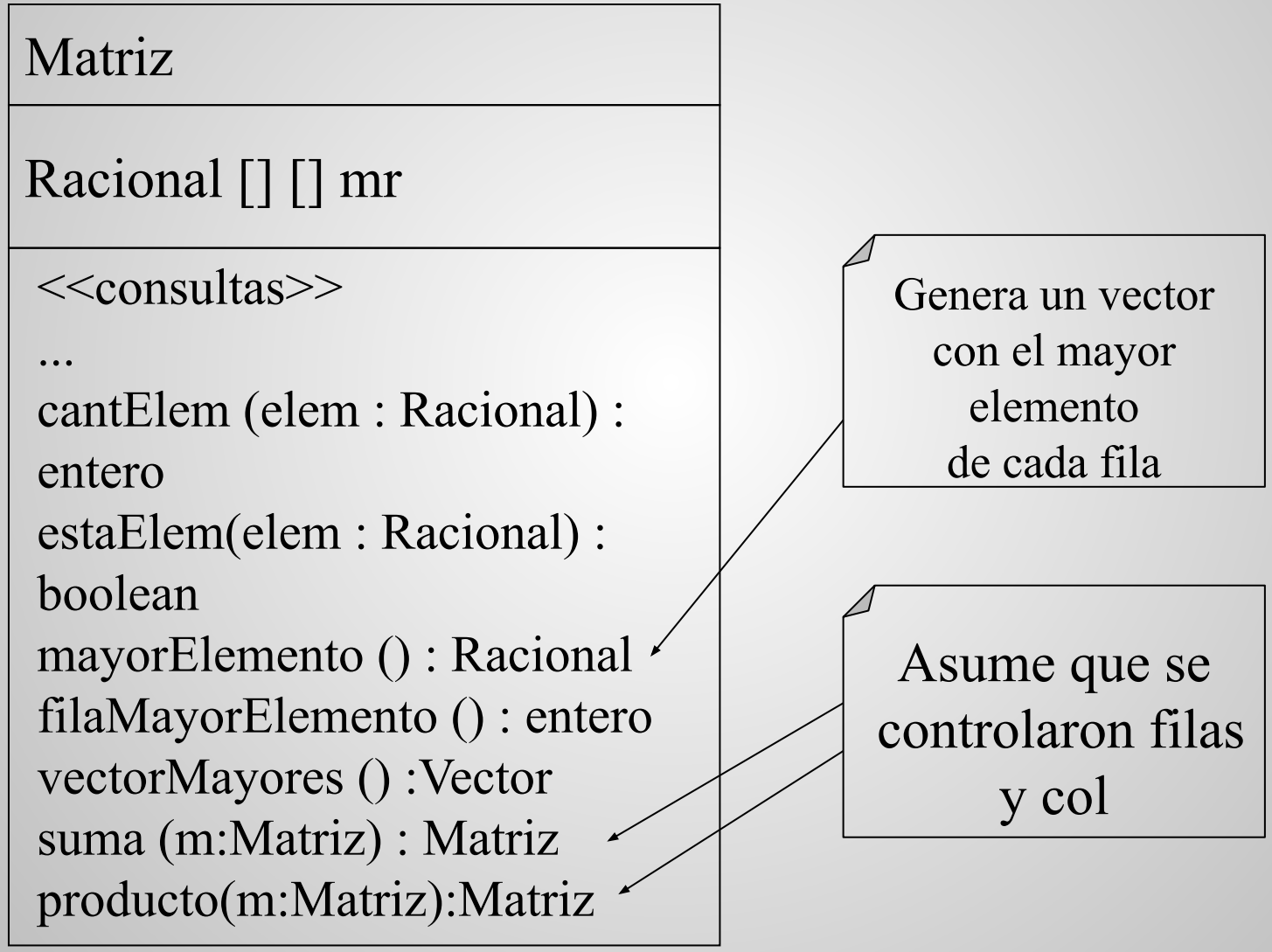
TDA Matriz Racionales - Diseño



TDA Matriz Racionales - Diseño



TDA Matriz Racionales - Diseño



TDA Matriz Racionales – Multiplicación de Matrices

- Dadas dos matrices A *(mxn)* y B *(nxp)*, tales que el numero de columnas de la matriz A es igual al numero de filas de la matriz B.
- El resultado de AB es una nueva matriz C *(mxp)* .

$$C = AB = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & \cdots & b_{1p} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{np} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + \cdots + a_{1n}b_{n1} & \cdots & a_{11}b_{1p} + \cdots + a_{1n}b_{np} \\ \vdots & \ddots & \vdots \\ a_{m1}b_{11} + \cdots + a_{mn}b_{n1} & \cdots & a_{m1}b_{1p} + \cdots + a_{mn}b_{np} \end{pmatrix}$$

$$C_{i,j} = a_{i,1}b_{1,j} + \ldots + a_{i,n}b_{n,j} = \Sigma(a_{i,k}\, b_{n,j}) \text{ para } k=1 \text{ hasta } k=n$$

TDA Matriz Racionales - Implementación

```
public Matriz producto(Matriz b){
    Matriz c = new Matriz(this.obtenerNfil(), b.obtenerNcol())
    for(int i = 0; i< c.obtenerNfil() ; i++){
        for(int j = 0; j < c.obtenerNcol() ; j++){

            //multiplicar fila i por columna j
        }
    }
    return c;
}
```

TDA Matriz Racionales - Implementación

```
//multiplicar fila i por columna j


_________ acumulador = ______0.0________;
for (int n = 0; n < this.obtenerNcol() ; n++){
    _________ prod = this.obtenerElem(i,
n)____X____(b.obtenerElem(n,j);
     acumulador = acumulador__+__(prod) ;
               }
c.establecerElem(i,j, acumulador);
```

TDA Matriz Racionales - Implementación

```
//multiplicar fila i por columna j

Racional componente = new Racional(0,1);
for (int n = 0; n < this.obtenerNcol() ; n++){
    Racional prod = this.obtenerElem(i,
n).producto(b.obtenerElem(n,j);
    componente = componente.suma(prod) ;
            }
c.establecerElem(i,j, Componente);
```