Heaps

E. Rivas

Abril de 2014

Resumen

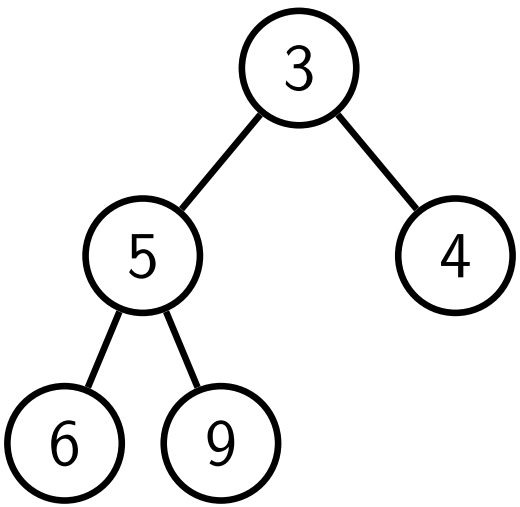
- Heaps
  - Definición y propiedades
  - Mínimo
  - Remover elemento
  - Insertar elemento
  - Implementación
- Aplicación al ordenamiento

Heaps - definición

Un **heap** es una estructura de datos basada en árboles binarios completos que satisfacen la propiedad de heap: *el valor de cada nodo es mayor o igual que el valor de su padre*.

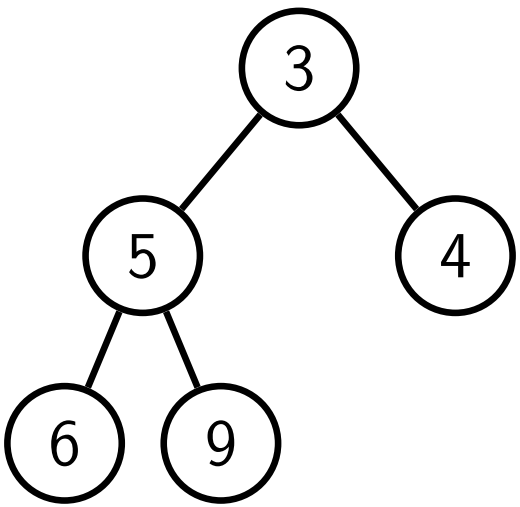
Heaps - definición

Un **heap** es una estructura de datos basada en árboles binarios completos que satisfacen la propiedad de heap: *el valor de cada nodo es mayor o igual que el valor de su padre*.



Heaps - definición

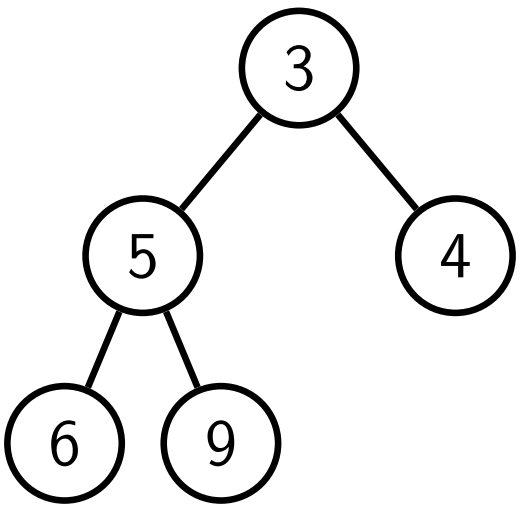
Un **heap** es una estructura de datos basada en árboles binarios completos que satisfacen la propiedad de heap: *el valor de cada nodo es mayor o igual que el valor de su padre*.



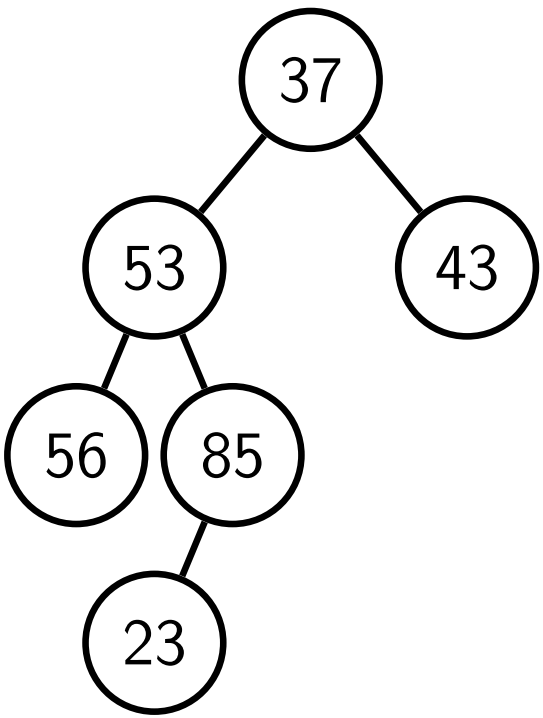
¡Es un heap!

Heaps - definición

Un **heap** es una estructura de datos basada en árboles binarios completos que satisfacen la propiedad de heap: *el valor de cada nodo es mayor o igual que el valor de su padre*.

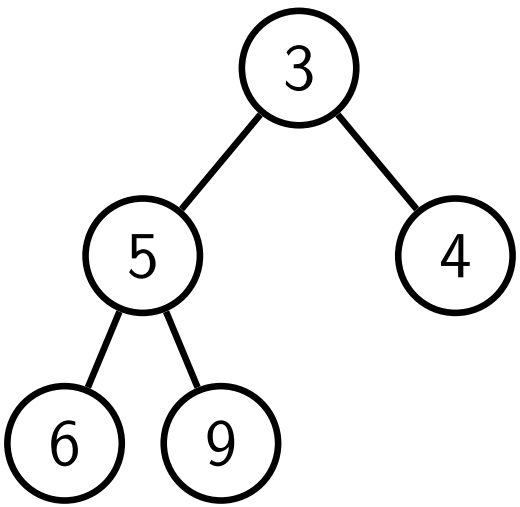


¡Es un heap!

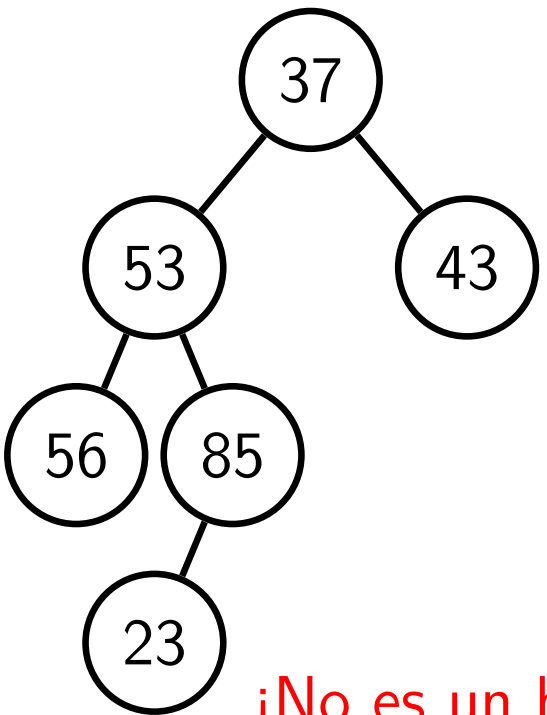


Heaps - definición

Un **heap** es una estructura de datos basada en árboles binarios completos que satisfacen la propiedad de heap: *el valor de cada nodo es mayor o igual que el valor de su padre*.



¡Es un heap!



¡No es un heap!

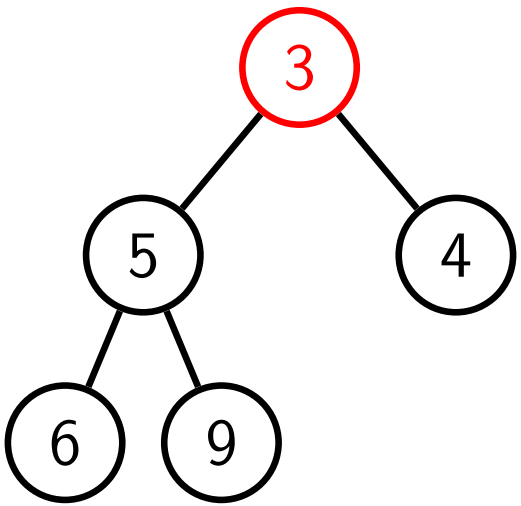
Heaps - propiedades y operaciones

En esta estructura de datos estaremos interesados en efectuar las siguientes operaciones:

- Observar el elemento mínimo: minimum.
- Quitar el elemento mínimo: erase_minimum.
- Insertar un elemento: insert.
- Verificar si está vacio: is_empty.

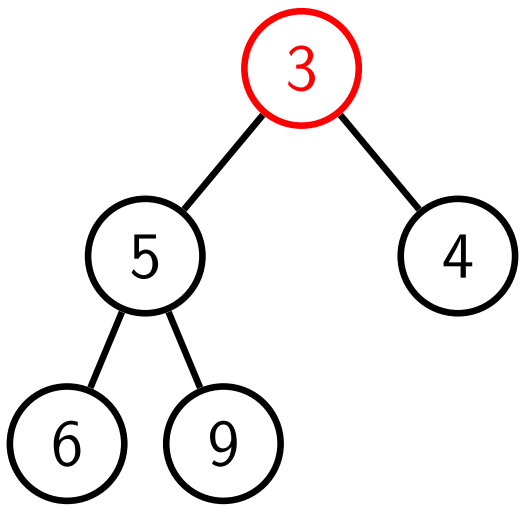
Heaps - mínimo

En esta estructura el mínimo está siempre en la raíz del árbol.



Heaps - quitar mínimo

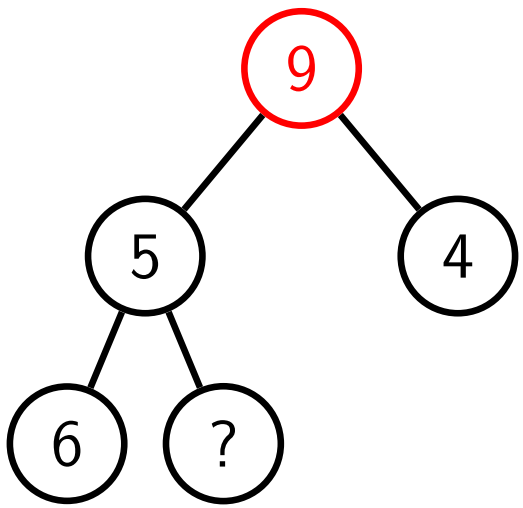
Para quitar el mínimo, reemplazaremos el elemento de la raiz por el último elemento del heap: el elemento del último nivel más a la derecha.



Luego, lo hacemos bajar hasta que no sea mayor a alguno de sus hijos.

Heaps - quitar mínimo

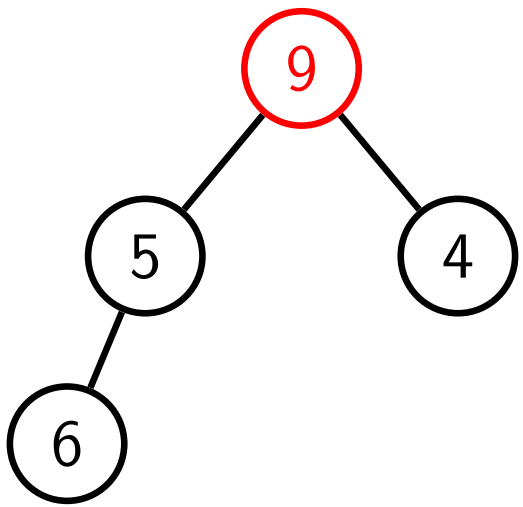
Para quitar el mínimo, reemplazaremos el elemento de la raiz por el último elemento del heap: el elemento del último nivel más a la derecha.



Luego, lo hacemos bajar hasta que no sea mayor a alguno de sus hijos.

Heaps - quitar mínimo

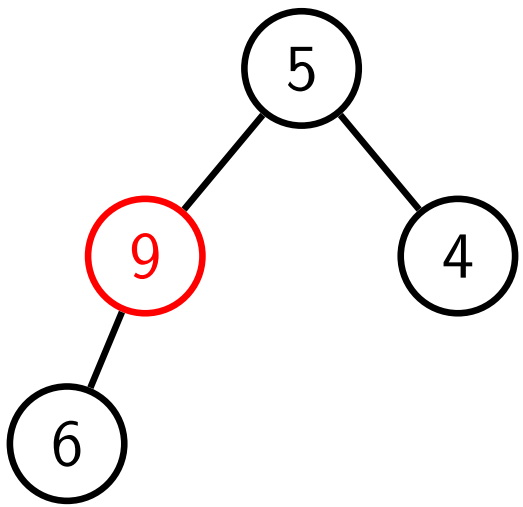
Para quitar el mínimo, reemplazaremos el elemento de la raiz por el último elemento del heap: el elemento del último nivel más a la derecha.



Luego, lo hacemos bajar hasta que no sea mayor a alguno de sus hijos.

Heaps - quitar mínimo

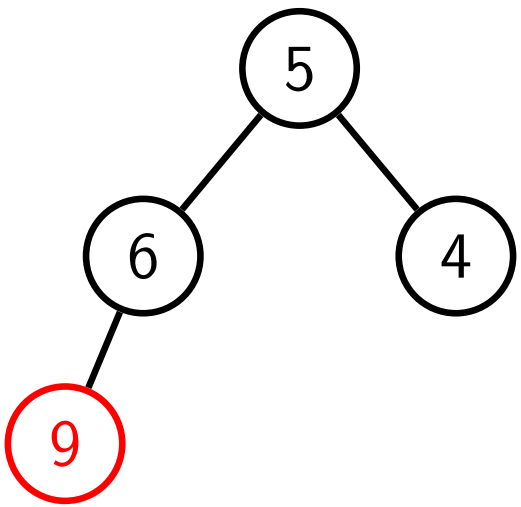
Para quitar el mínimo, reemplazaremos el elemento de la raiz por el último elemento del heap: el elemento del último nivel más a la derecha.



Luego, lo hacemos bajar hasta que no sea mayor a alguno de sus hijos.

Heaps - quitar mínimo

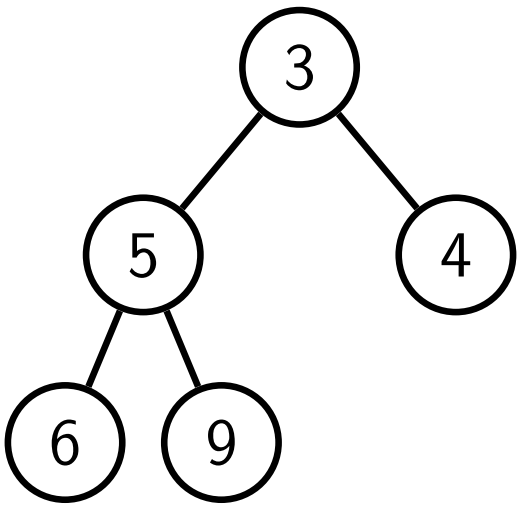
Para quitar el mínimo, reemplazaremos el elemento de la raiz por el último elemento del heap: el elemento del último nivel más a la derecha.



Luego, lo hacemos bajar hasta que no sea mayor a alguno de sus hijos.

Heaps - insertar elemento

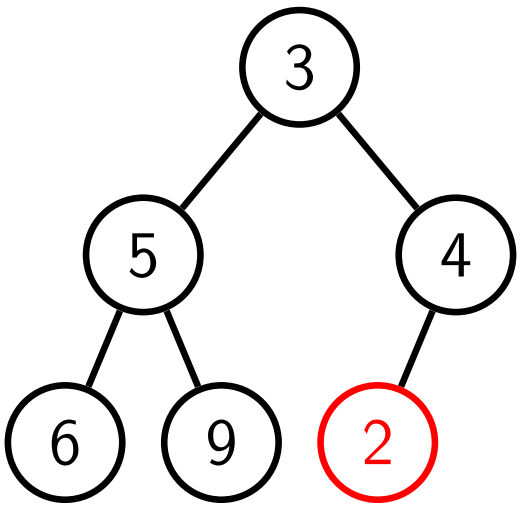
Para insertar un elemento empezamos por ponerlo en la primer posición libre del heap.



Luego, lo hacemos subir hasta que no sea menor a su padre.

Heaps - insertar elemento

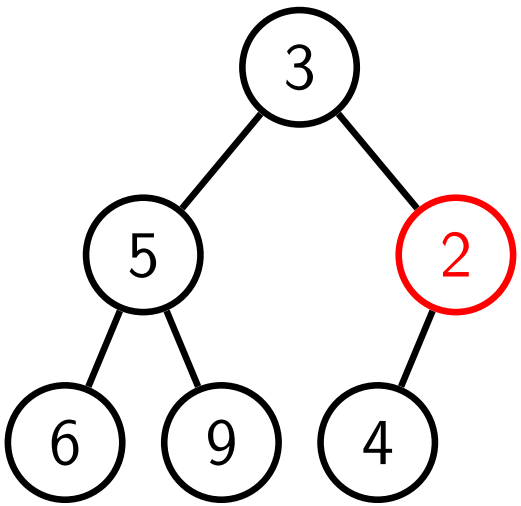
Para insertar un elemento empezamos por ponerlo en la primer posición libre del heap.



Luego, lo hacemos subir hasta que no sea menor a su padre.

Heaps - insertar elemento

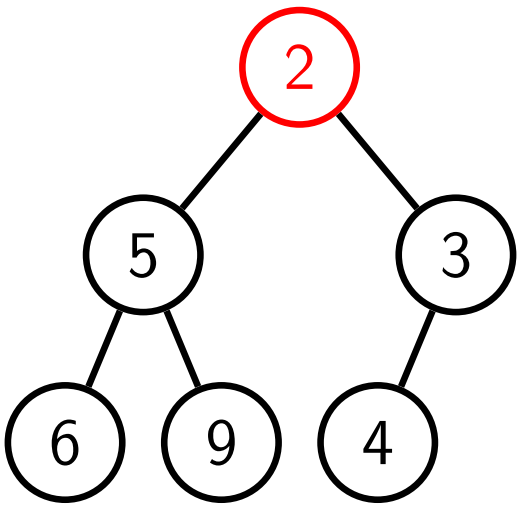
Para insertar un elemento empezamos por ponerlo en la primer posición libre del heap.



Luego, lo hacemos subir hasta que no sea menor a su padre.

Heaps - insertar elemento

Para insertar un elemento empezamos por ponerlo en la primer posición libre del heap.



Luego, lo hacemos subir hasta que no sea menor a su padre.

Heaps - implementación con arreglos

La siguiente estructura en C modela heaps usando arreglos:

```
typedef struct _BHeap {
    int    data[MAX];
    int    nelems;
} BHeap;
```

En `nelems` llevamos la cantidad de elementos alojados.

Heaps - implementación con arreglos

La siguiente estructura en C modela heaps usando arreglos:

```
typedef struct _BHeap {
    int    data[MAX];
    int    nelems;
} BHeap;
```

En `nelems` llevamos la cantidad de elementos alojados.

Ejercicio: implementar `void bheap_insert(Heap *h, int val)` que inserta un elemento un heap.

Problema de ordenamiento

Buscar una función `sort` tal que:

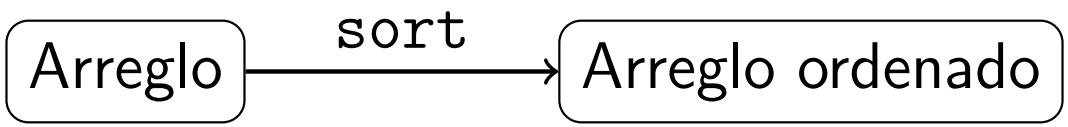
Problema de ordenamiento

Buscar una función `sort` tal que:

Arreglo

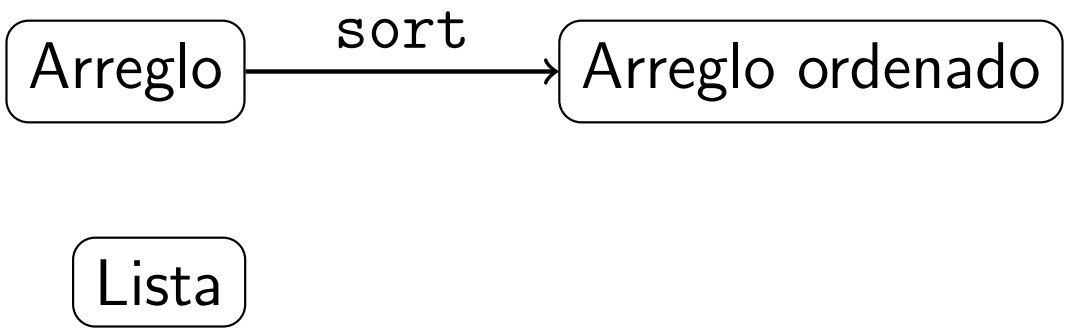
Problema de ordenamiento

Buscar una función `sort` tal que:



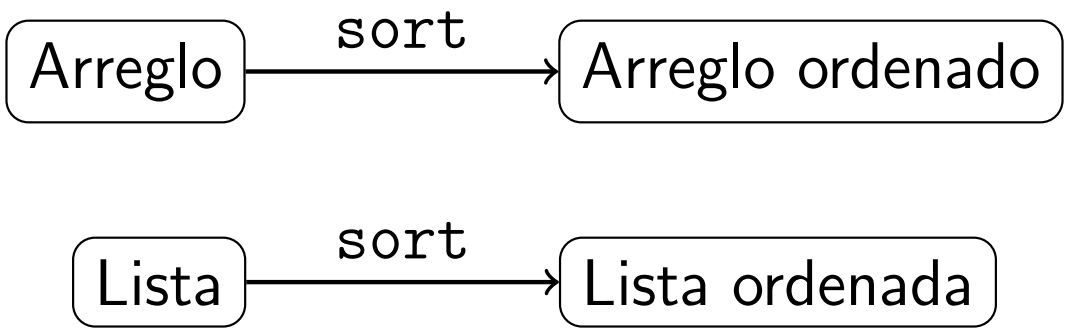
Problema de ordenamiento

Buscar una función `sort` tal que:



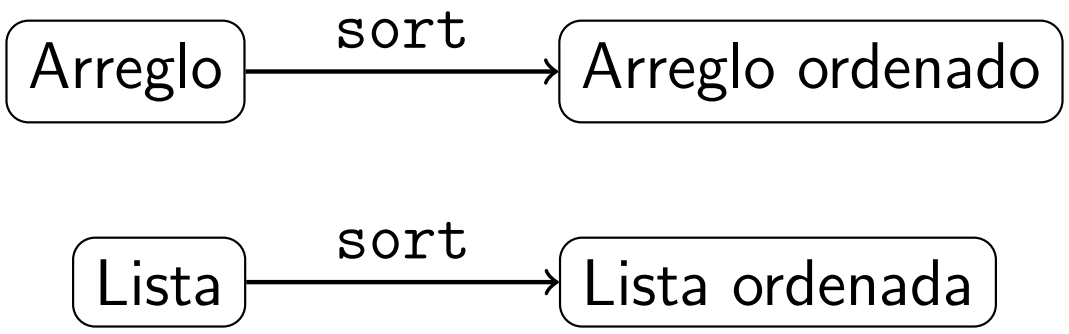
Problema de ordenamiento

Buscar una función `sort` tal que:



Problema de ordenamiento

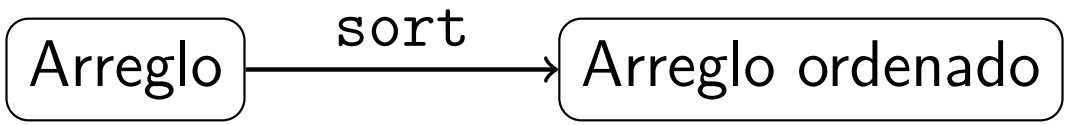
Buscar una función `sort` tal que:

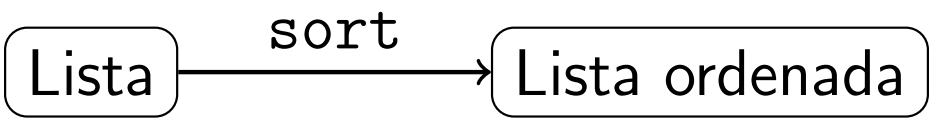


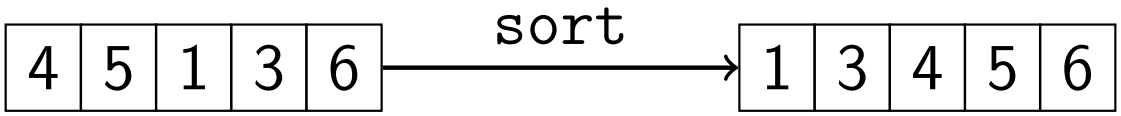
4	5	1	3	6

Problema de ordenamiento

Buscar una función `sort` tal que:







Función sort - forma

Pueden existir varios sabores para la función que buscamos:

```
void sort(int data[], int sz);

int *sort(int data[], int sz);

void sort(SList *l);

SList *sort(SList *l);
```

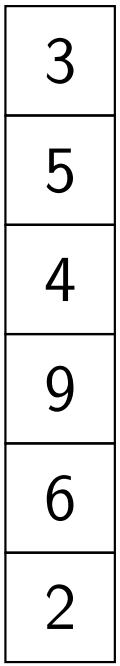
Heapsort

$$\text{heapsort} = \text{heap} + \text{sort}$$

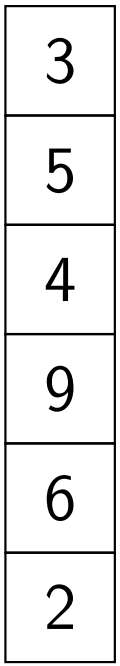
heapsort
(computación) algoritmo de ordenamiento basado en la estructura de datos heap.

Idea

En detalle

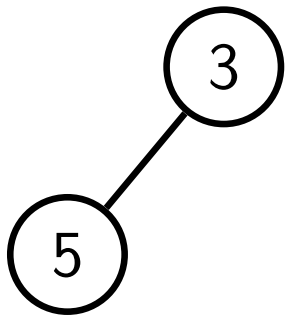


En detalle



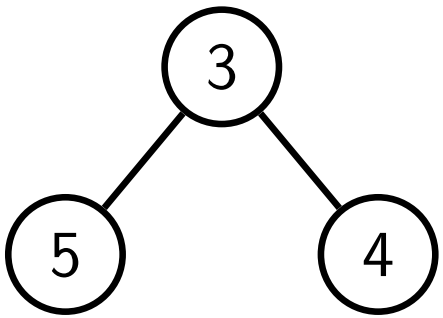
En detalle

3
5
4
9
6
2



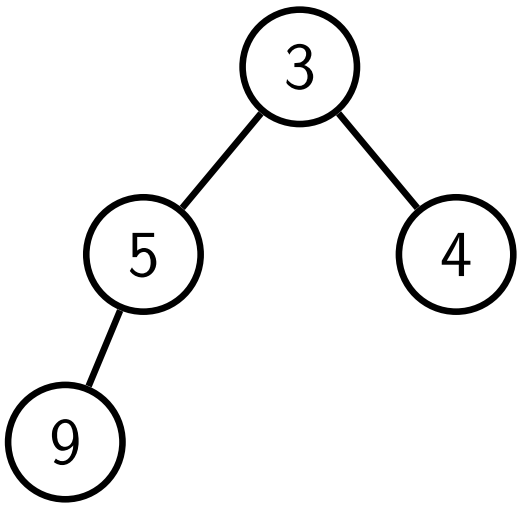
En detalle

3
5
4
9
6
2



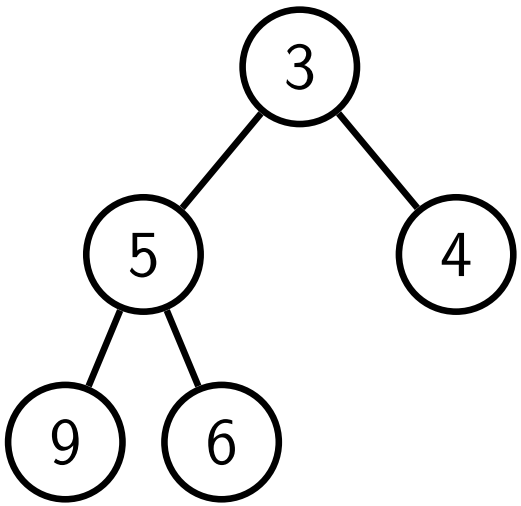
En detalle

3
5
4
9
6
2



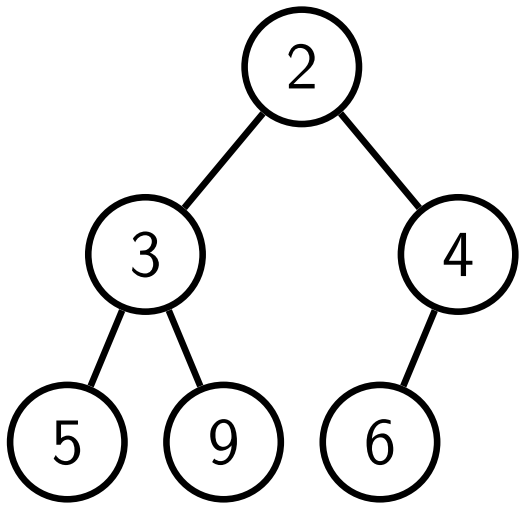
En detalle

3
5
4
9
6
2

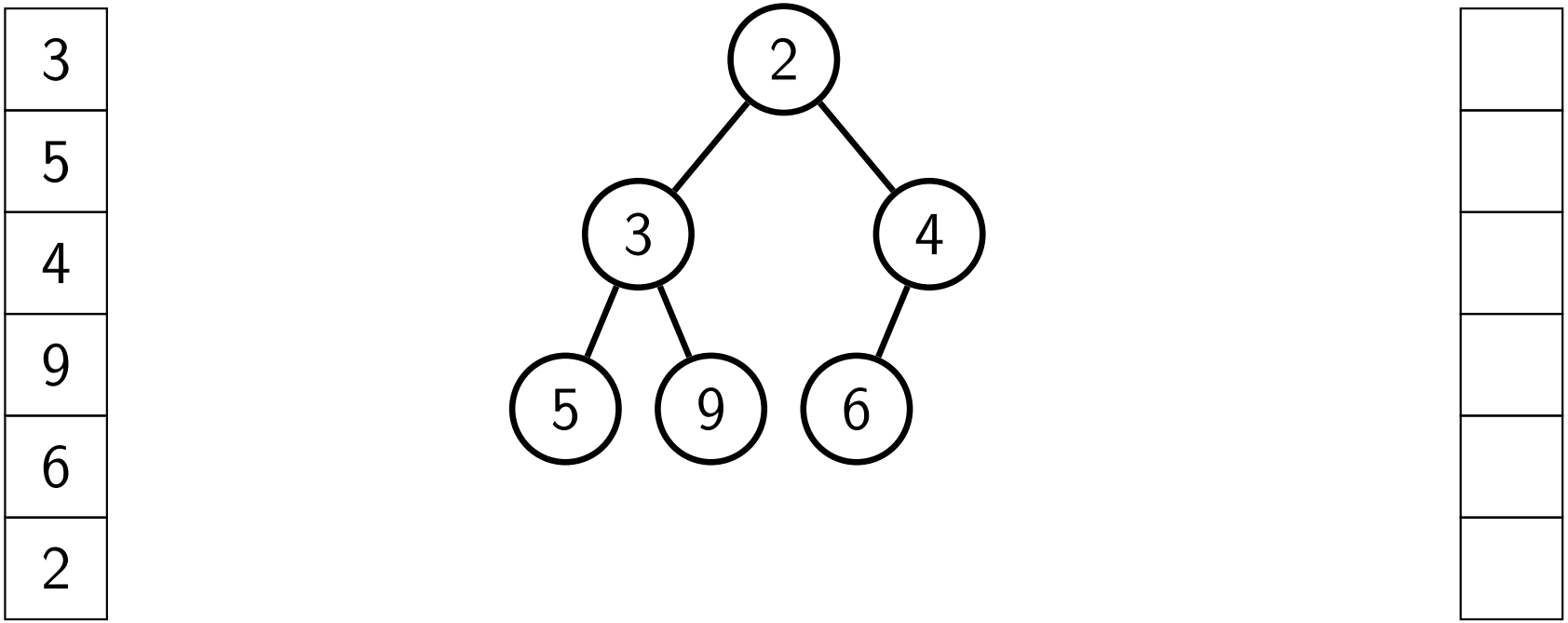


En detalle

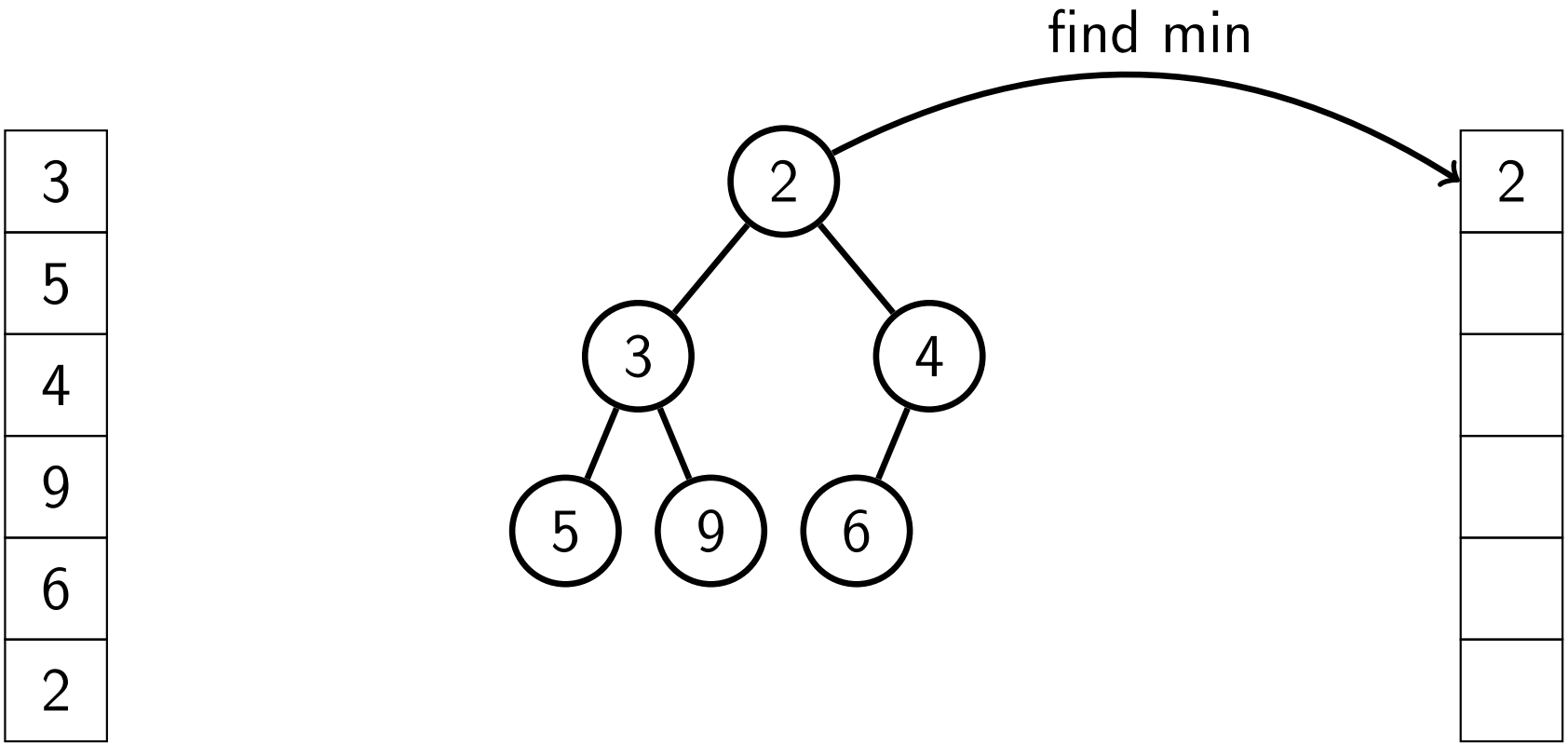
3
5
4
9
6
2



En detalle

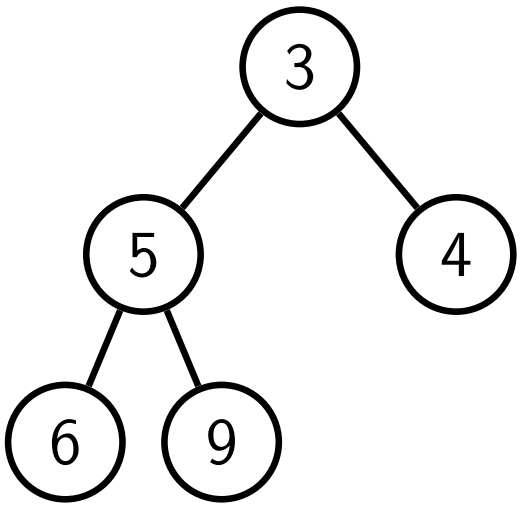


En detalle



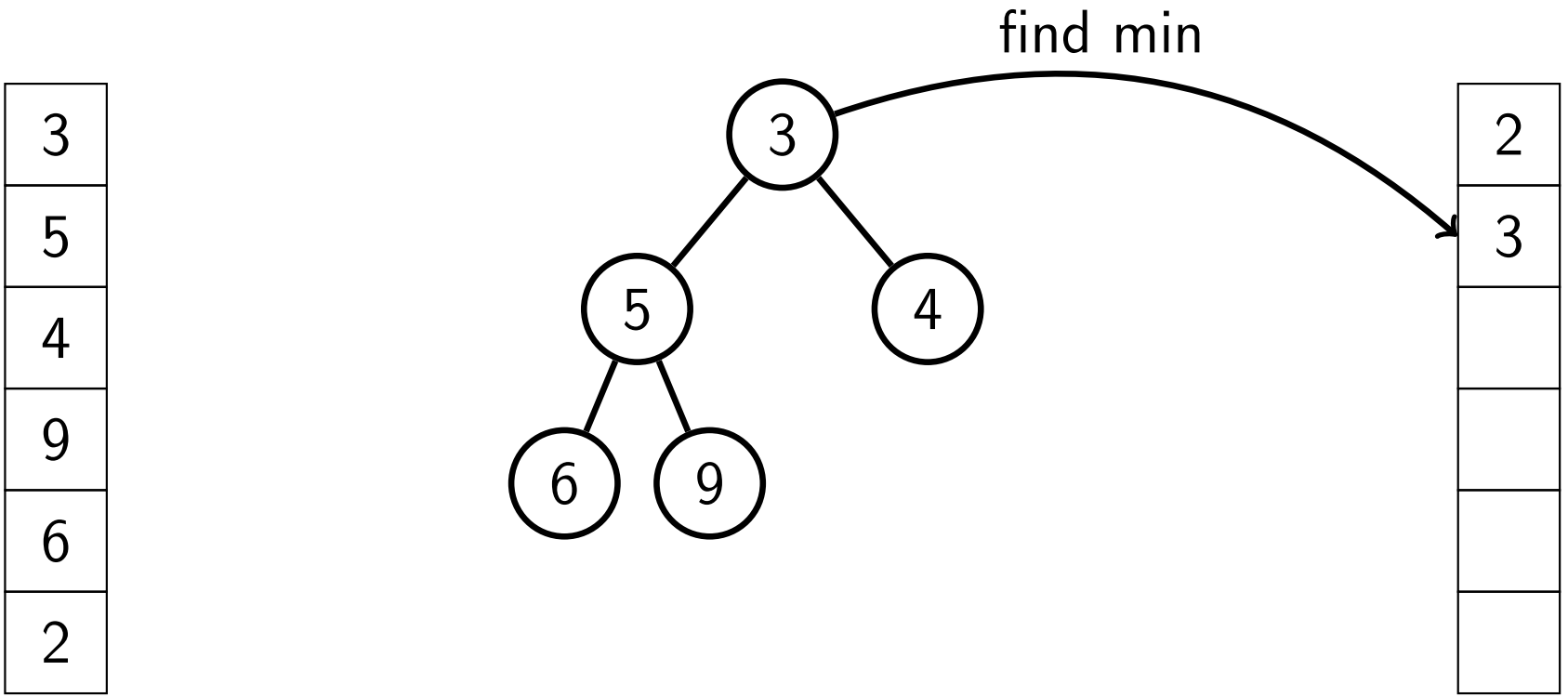
En detalle

3
5
4
9
6
2



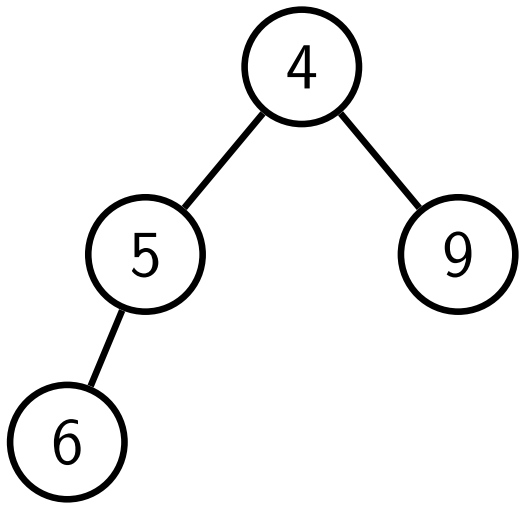
2

En detalle



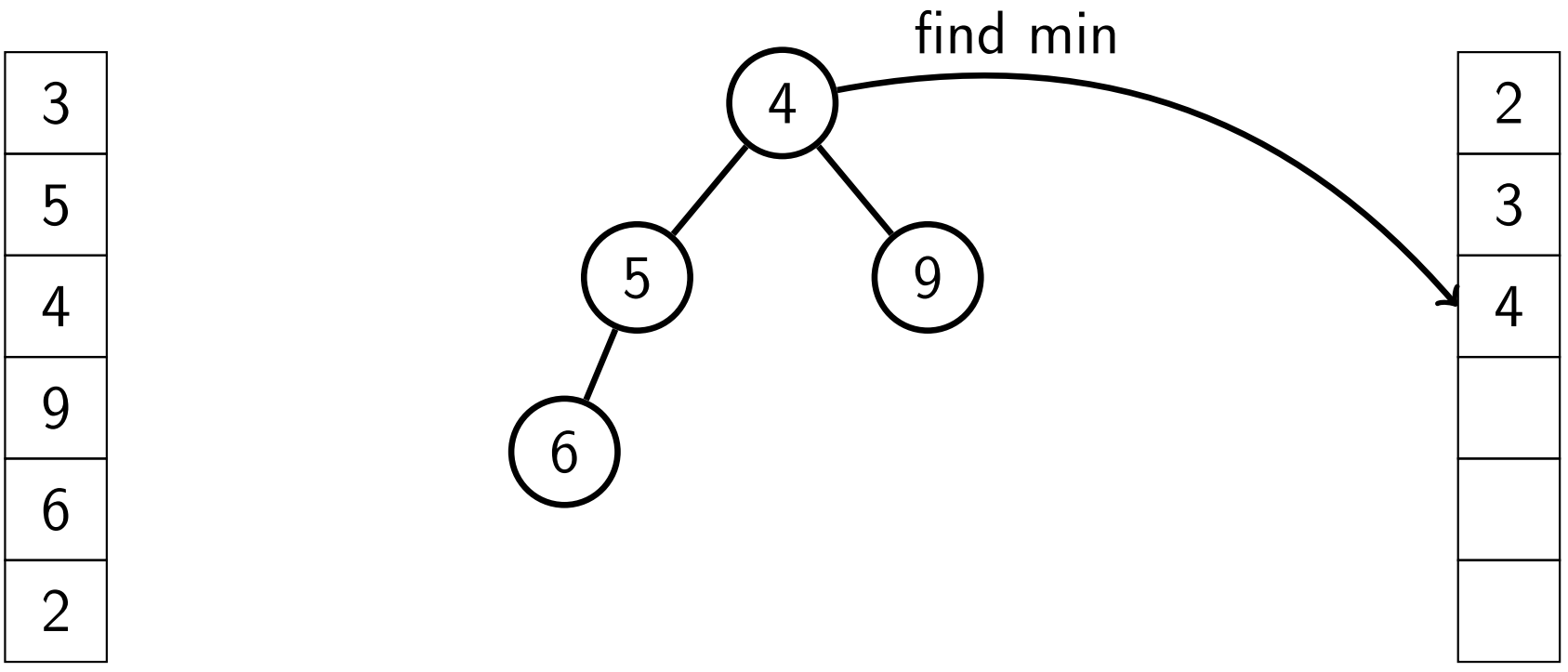
En detalle

3
5
4
9
6
2



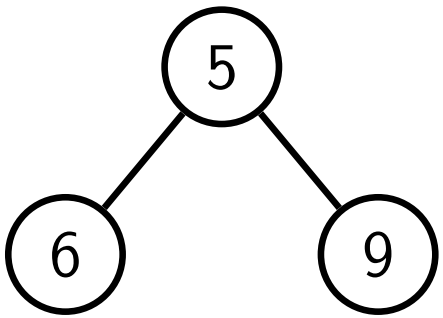
2
3

En detalle



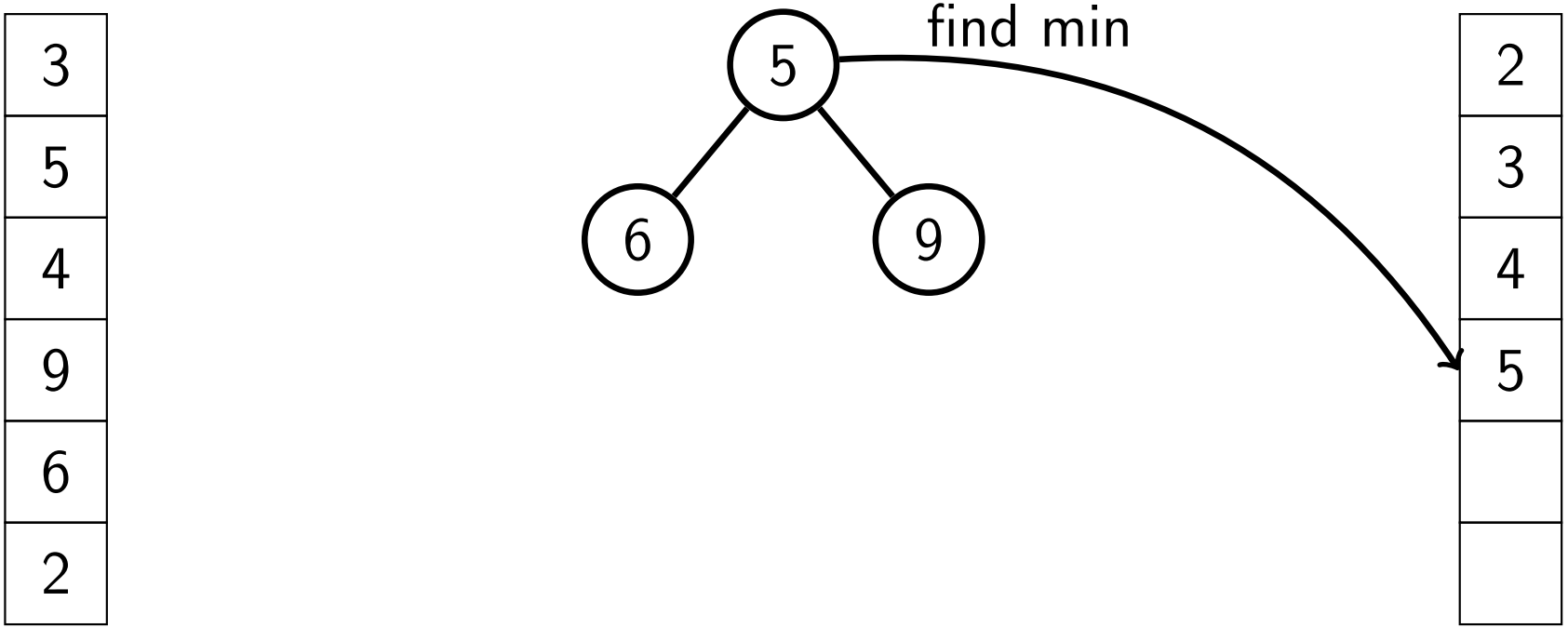
En detalle

3
5
4
9
6
2



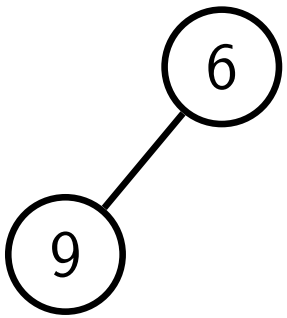
2
3
4

En detalle



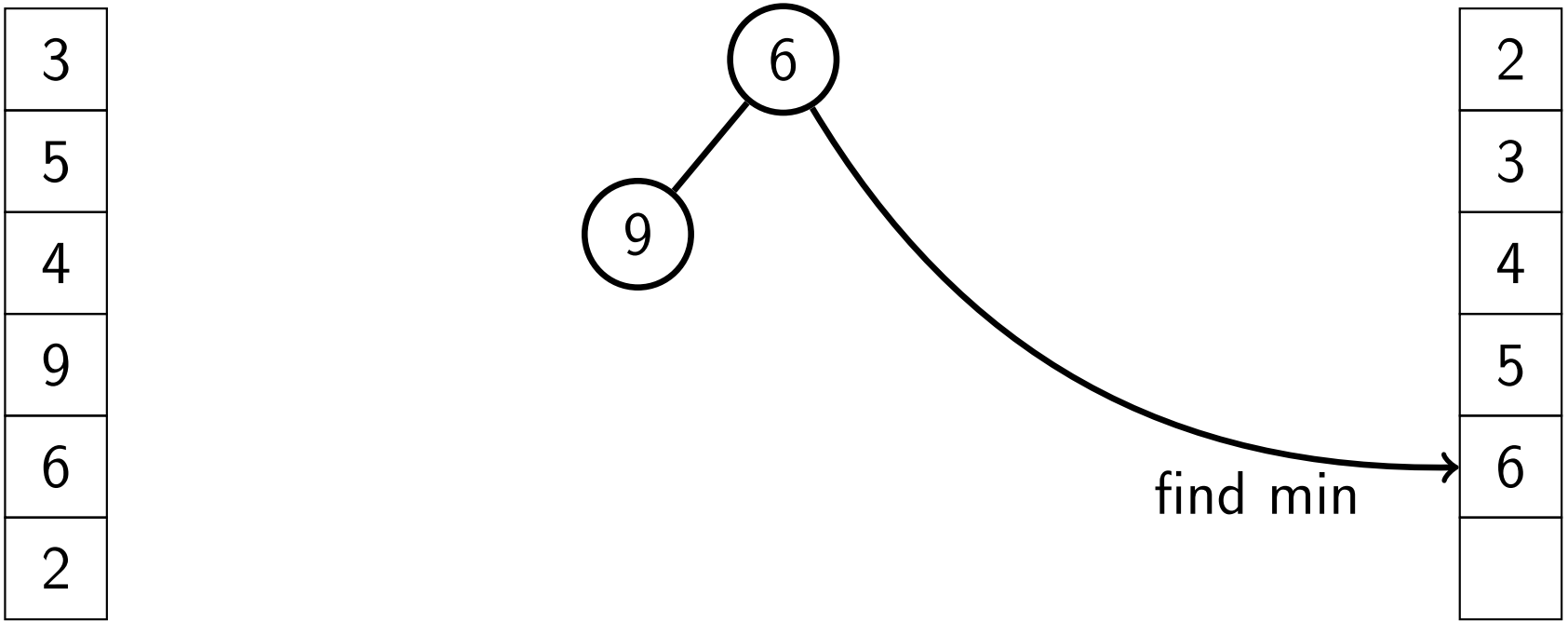
En detalle

3
5
4
9
6
2

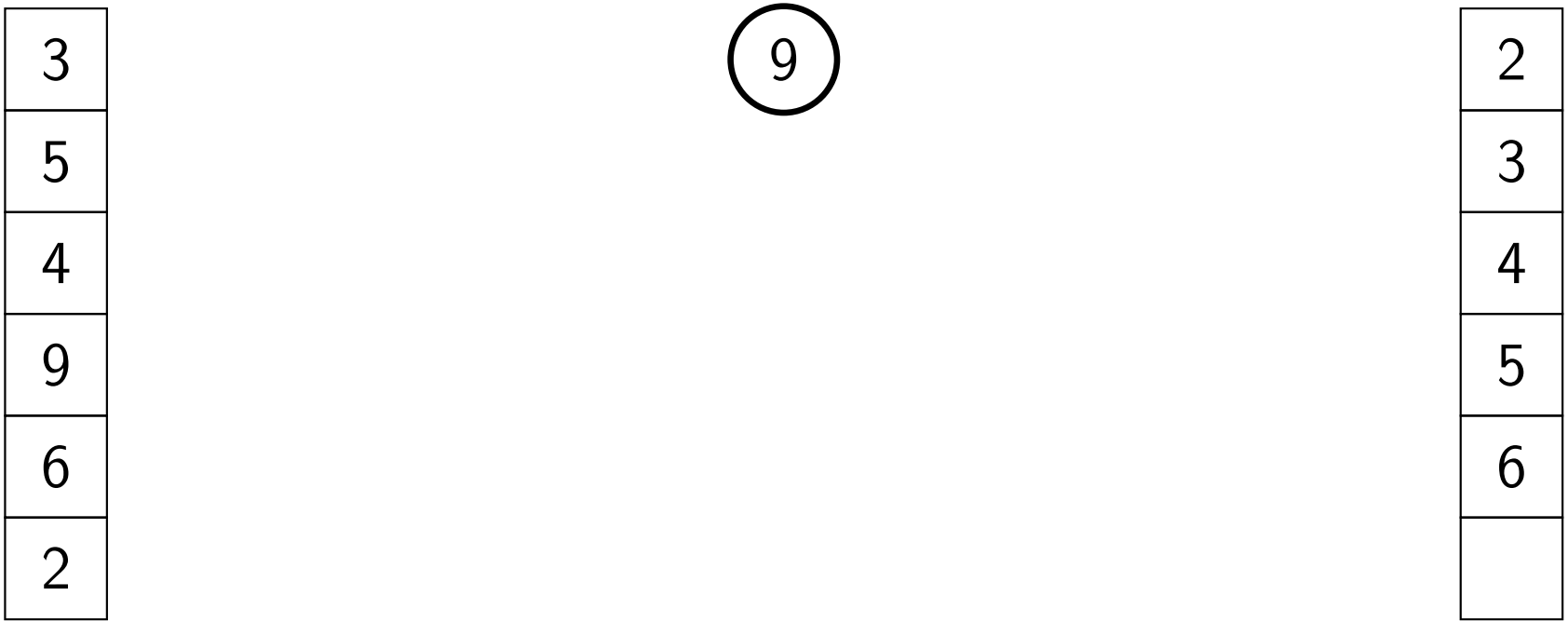


2
3
4
5

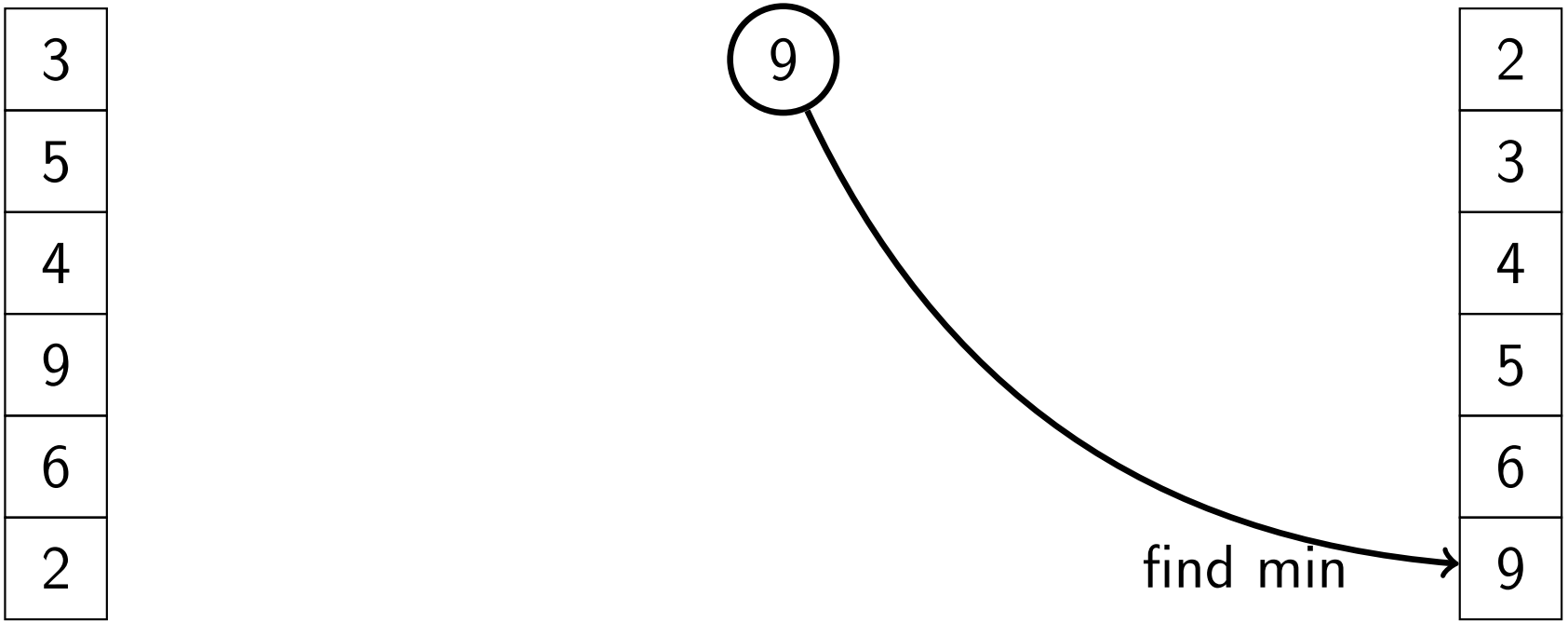
En detalle



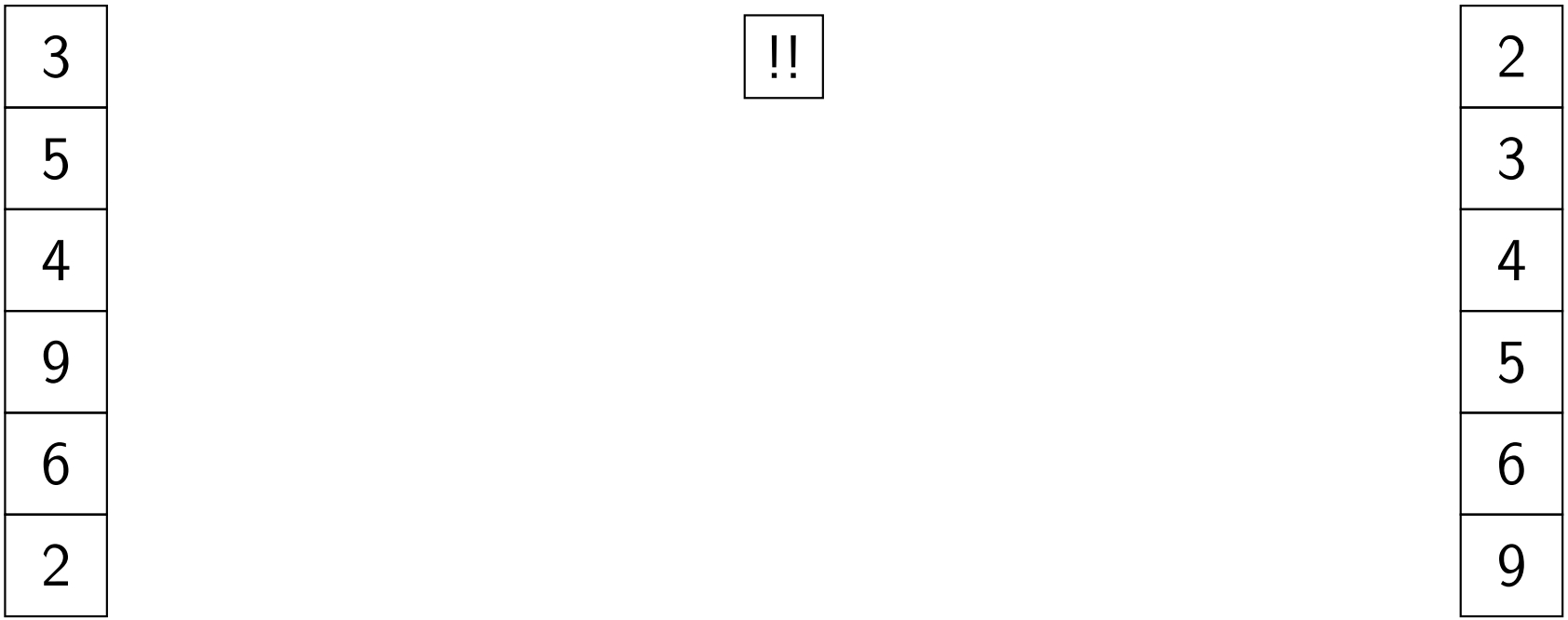
En detalle



En detalle



En detalle



Implementación en C

```
int *heapsort(int data[], int sz) {
  int *l = malloc(sizeof(int)*sz);
  BHeap *h = bheap_create();

  for (i = 0; i < sz; i++)
    h = bheap_insert(h, data[i]);

  i = 0;
  while (!bheap_is_empty(h)) {
    l[i++] = bheap_minimum(h);
    h = bheap_erase_minimum(h);
  }
  bheap_destroy(h);

  return l;
}
```