Heapsort

E. Rivas

Mayo 2013

Introducción

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

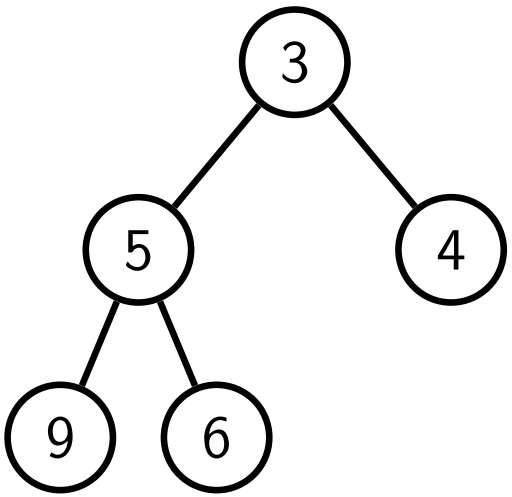
heapsort /hiːp'sɔːrt/
(computación) algoritmo de ordenamiento basado en la estructura de datos heap.

Heaps, un repaso

min(max)-heap. árbol binario completo la propiedad que el elemento de cada nodo es menor (mayor) a los elementos de sus subárboles.

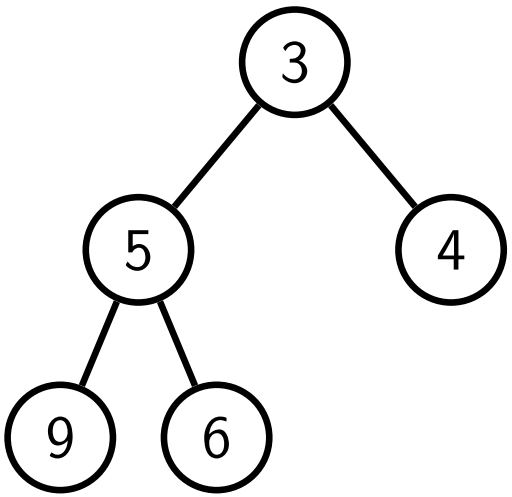
Heaps, un repaso

min(max)-heap. árbol binario completo la propiedad que el elemento de cada nodo es menor (mayor) a los elementos de sus subárboles.



Heaps, un repaso

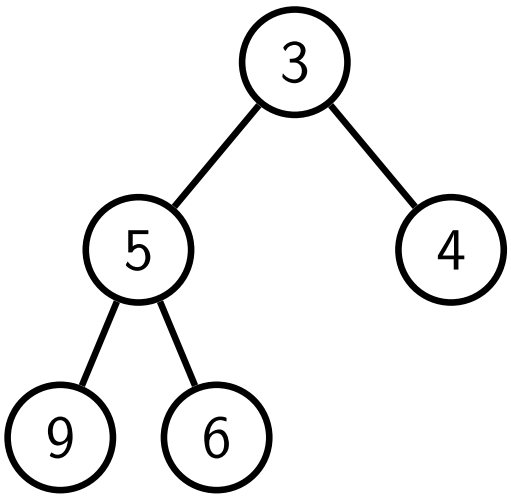
min(max)-heap. árbol binario completo la propiedad que el elemento de cada nodo es menor (mayor) a los elementos de sus subárboles.



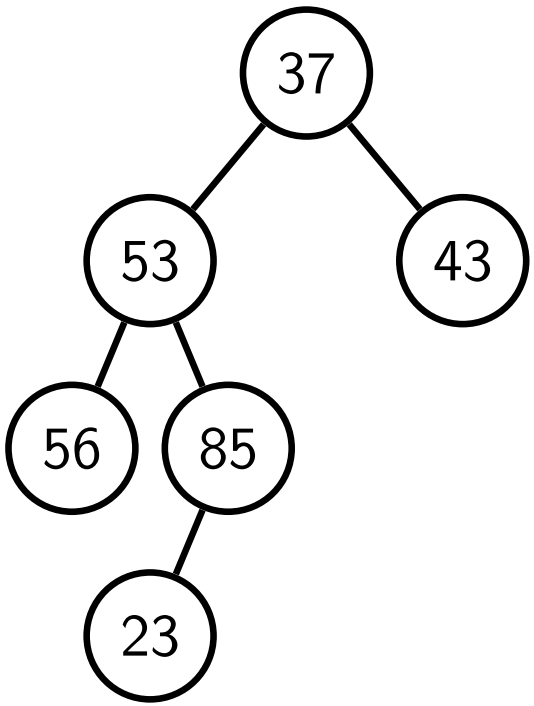
¡Es un heap!

Heaps, un repaso

min(max)-heap. árbol binario completo la propiedad que el elemento de cada nodo es menor (mayor) a los elementos de sus subárboles.

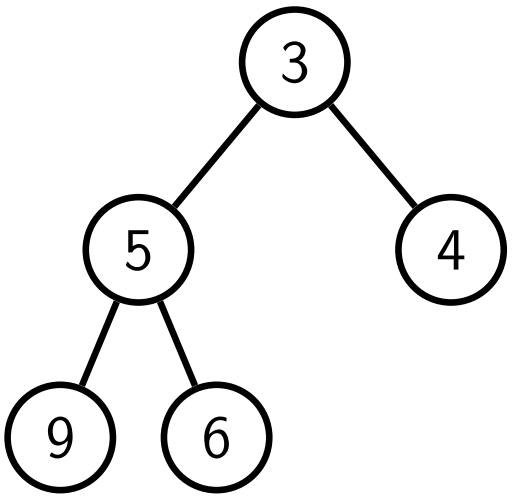


¡Es un heap!

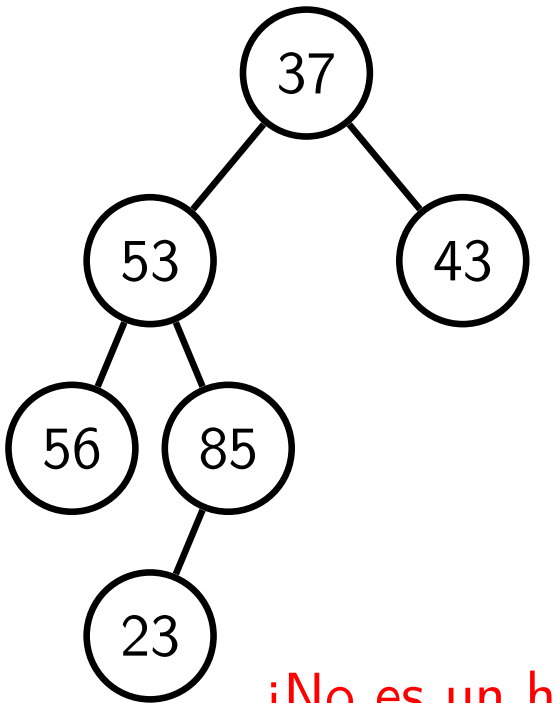


Heaps, un repaso

min(max)-heap. árbol binario completo la propiedad que el elemento de cada nodo es menor (mayor) a los elementos de sus subárboles.



¡Es un heap!



¡No es un heap!

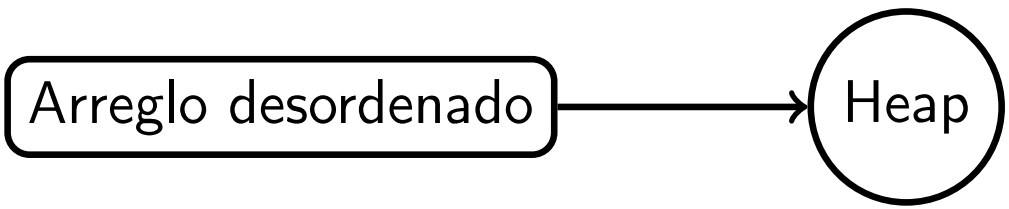
Nuestra codificación de heaps

```
typedef ... BHeap;

BHeap *bheap_create();

int bheap_is_empty(BHeap *);

BHeap *bheap_insert(BHeap *, int);

void bheap_erase_minimum(BHeap *);

int bheap_minimum(BHeap *);

void bheap_destroy(BHeap *);
```

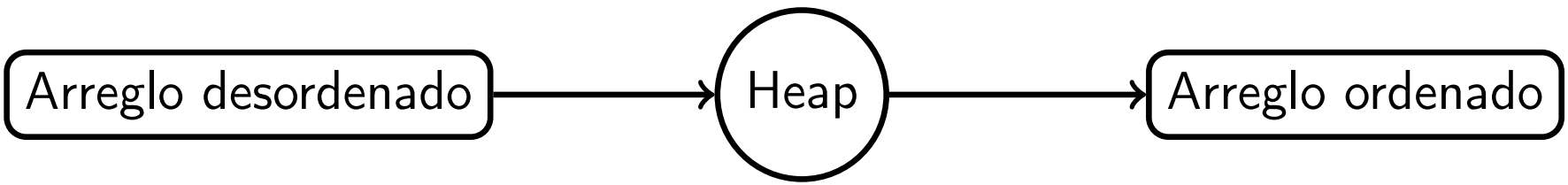
Idea

Arreglo desordenado

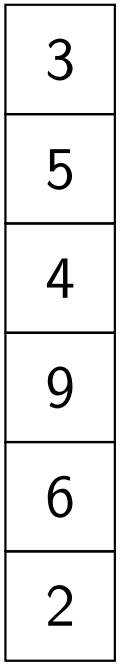
Idea



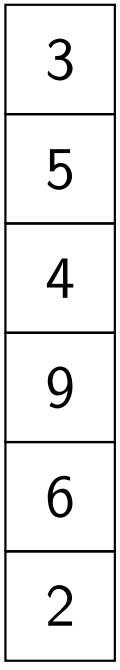
Idea



En detalle

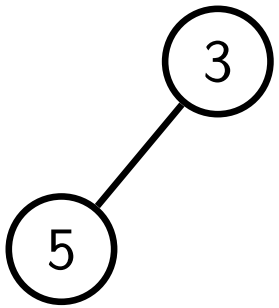


En detalle



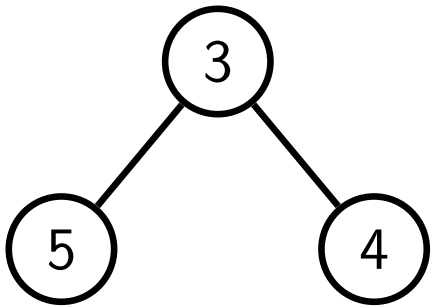
En detalle

3
5
4
9
6
2



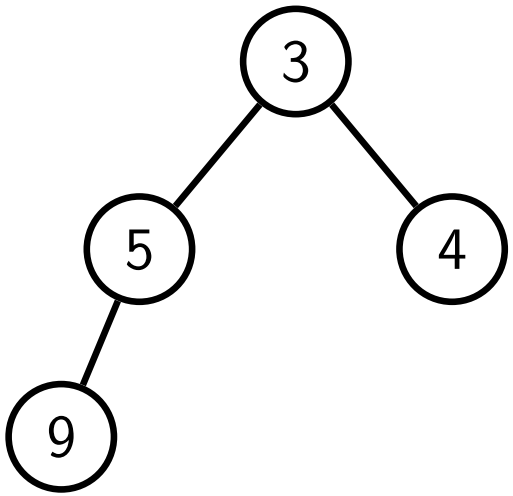
En detalle

3
5
4
9
6
2



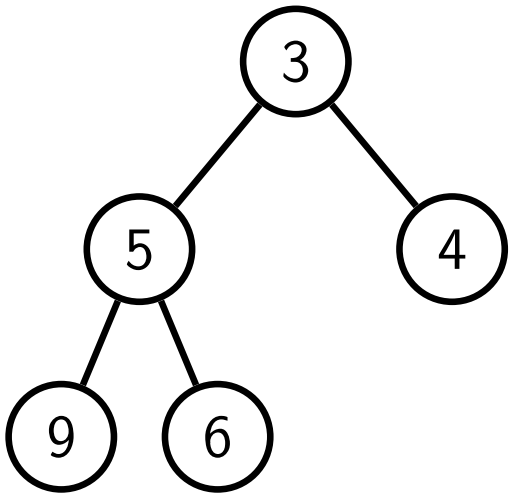
En detalle

3
5
4
9
6
2



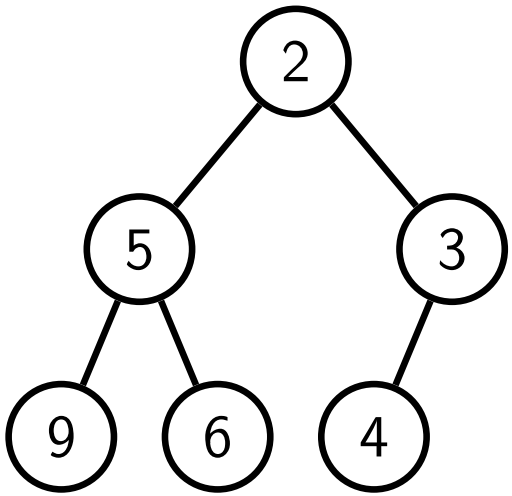
En detalle

3
5
4
9
6
2



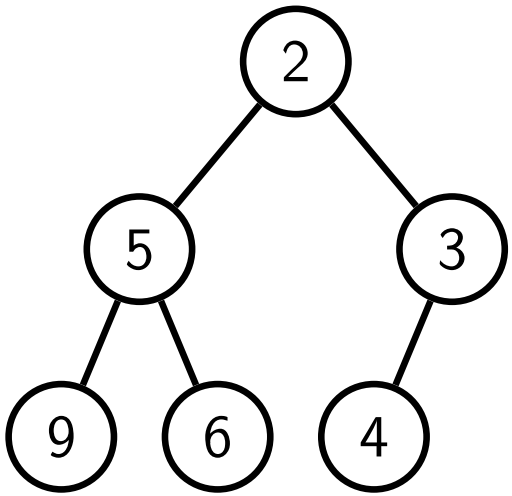
En detalle

3
5
4
9
6
2

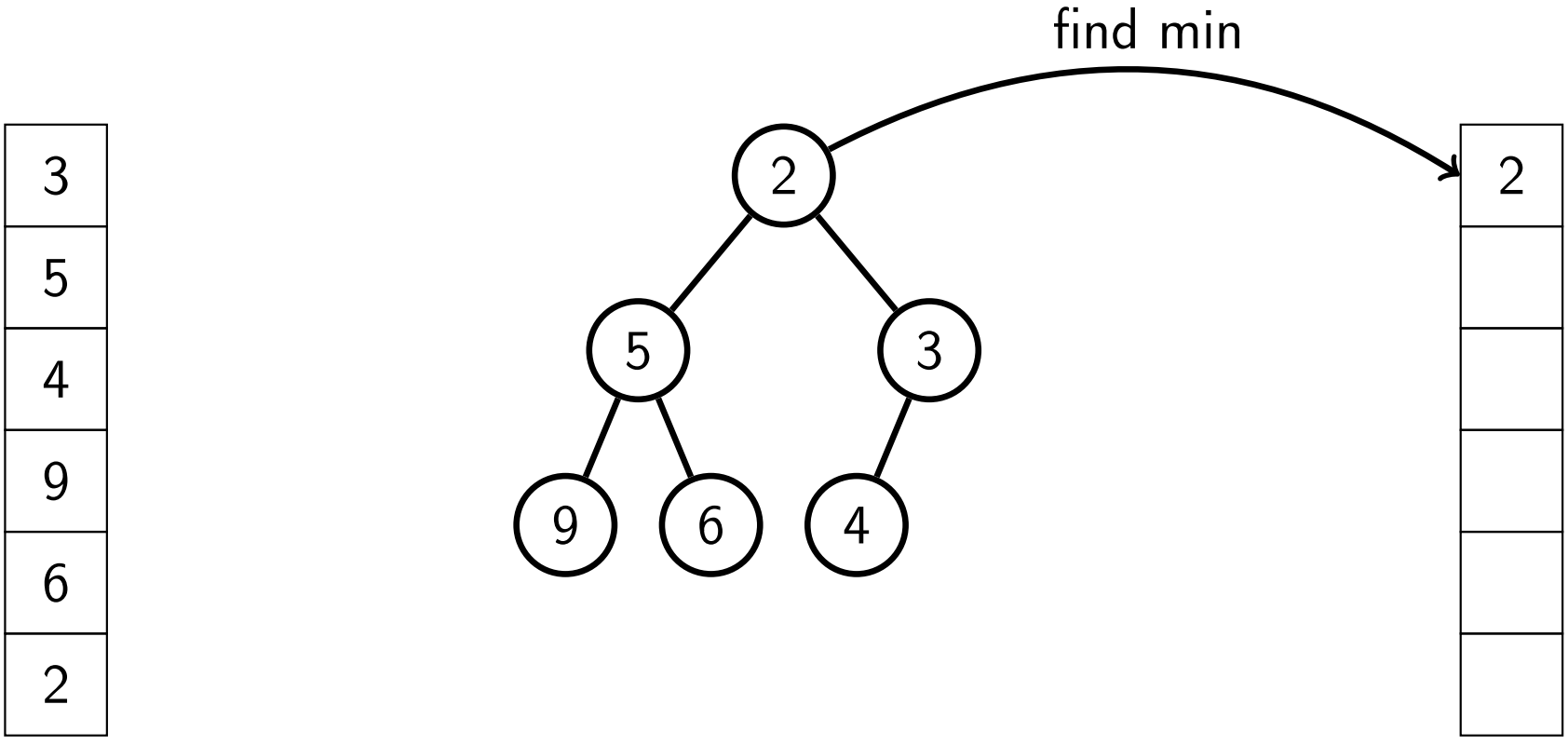


En detalle

3
5
4
9
6
2

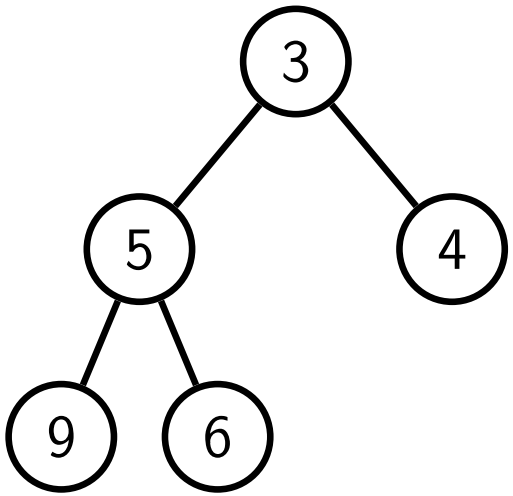


En detalle



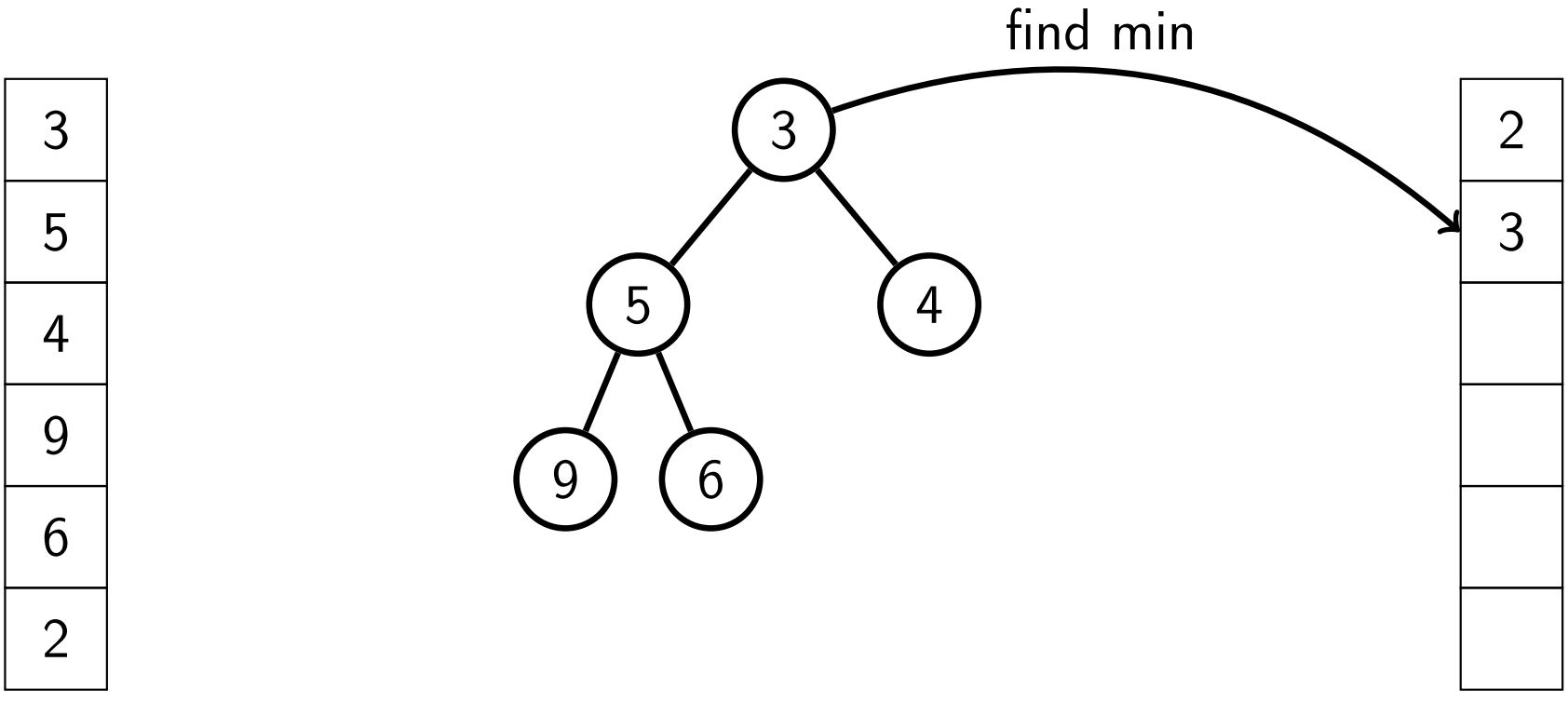
En detalle

3
5
4
9
6
2



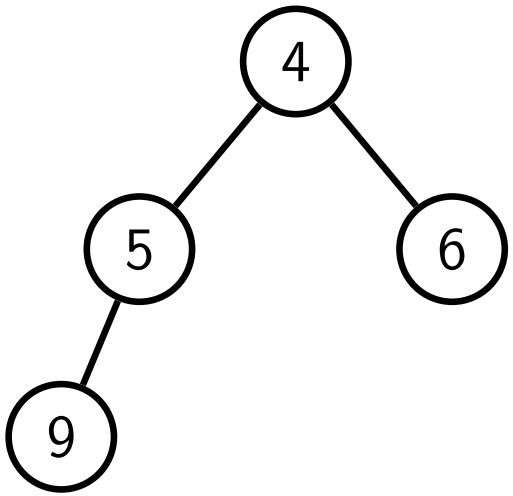
2

En detalle



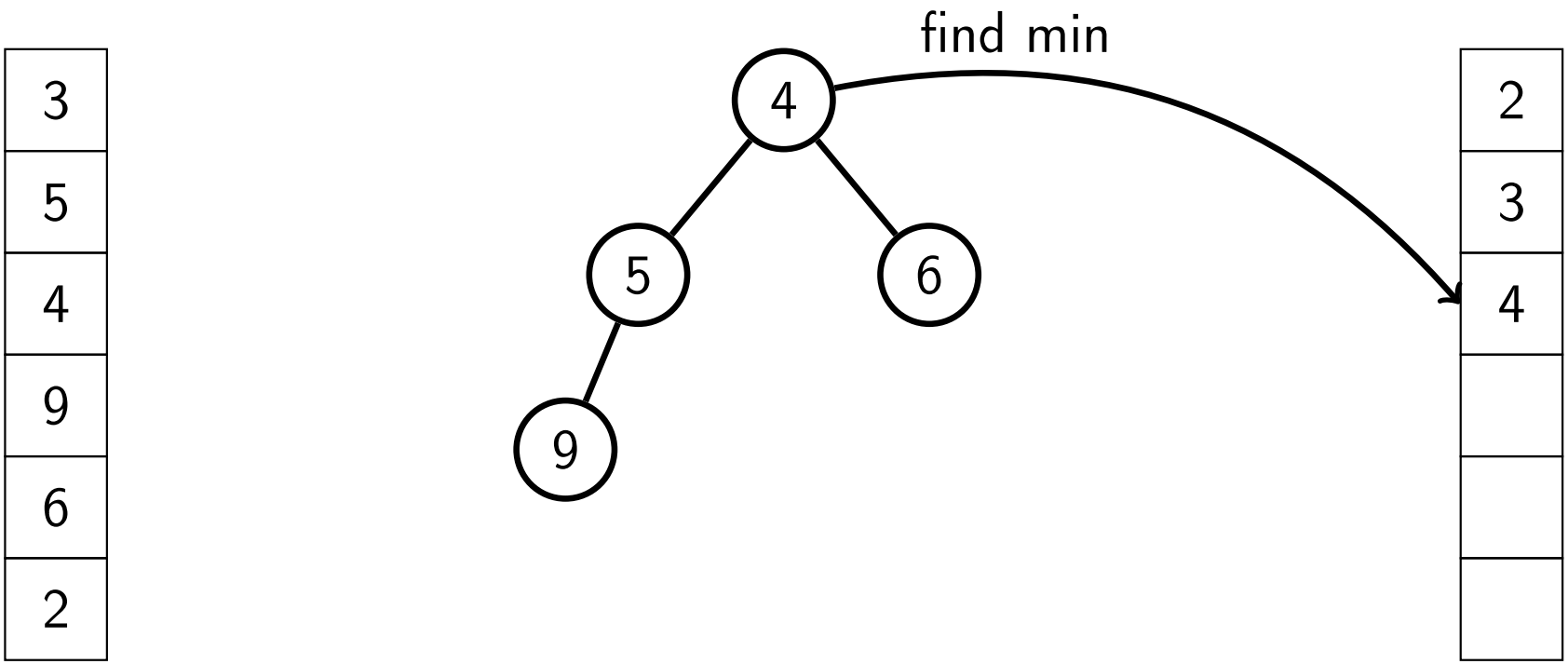
En detalle

3
5
4
9
6
2



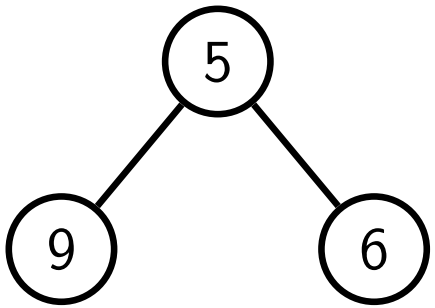
2
3

En detalle



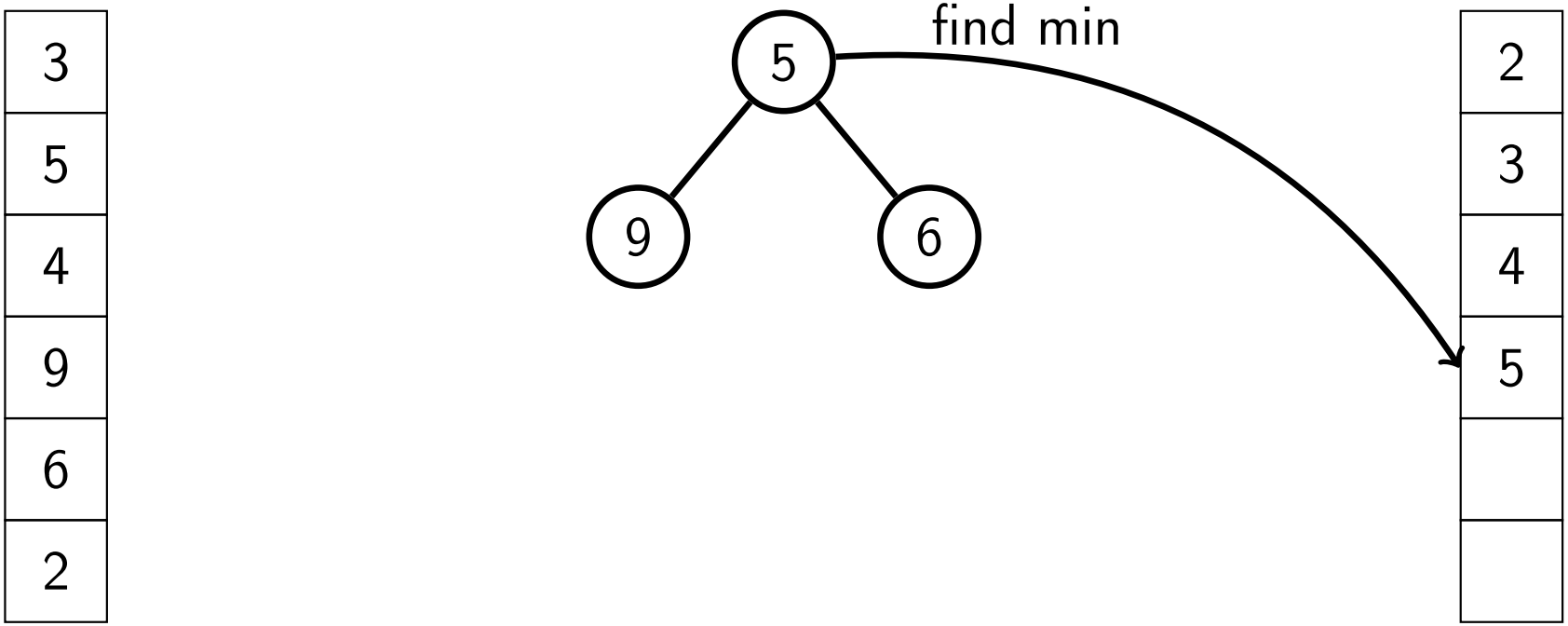
En detalle

3
5
4
9
6
2



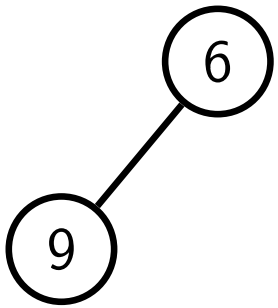
2
3
4

En detalle



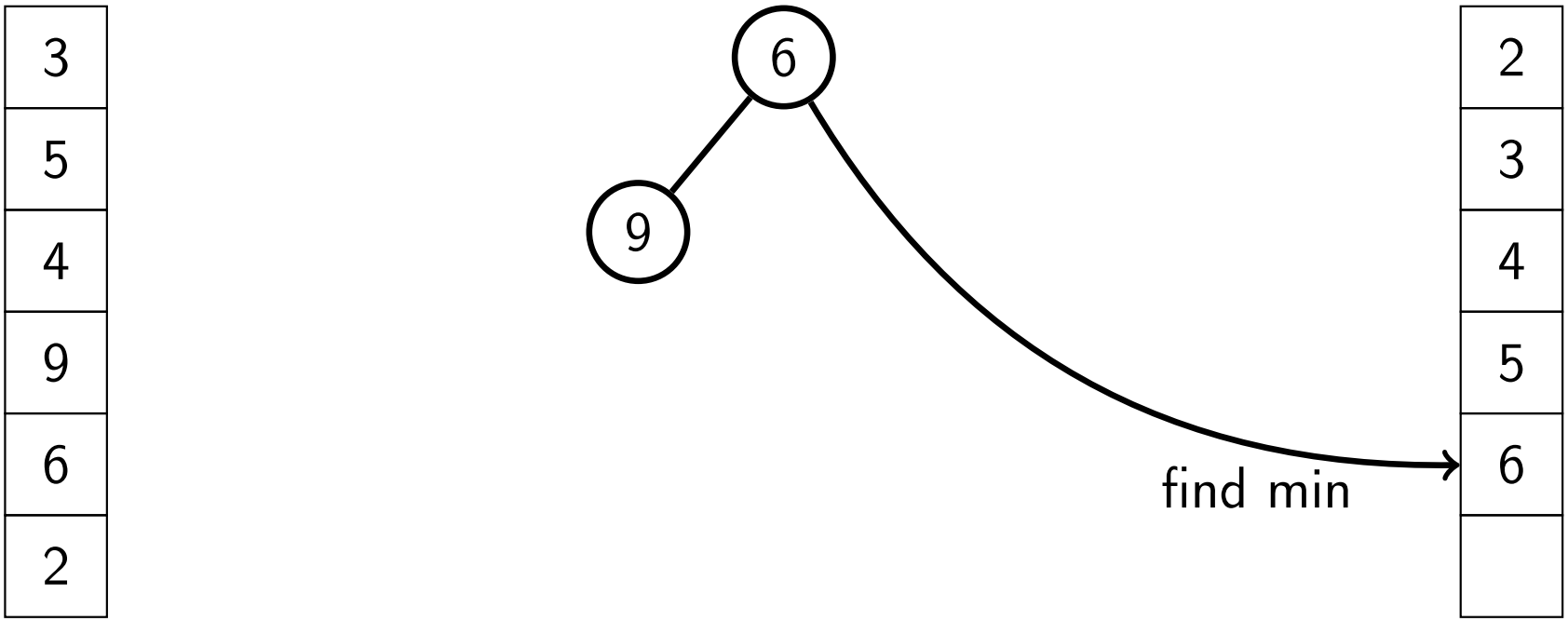
En detalle

3
5
4
9
6
2

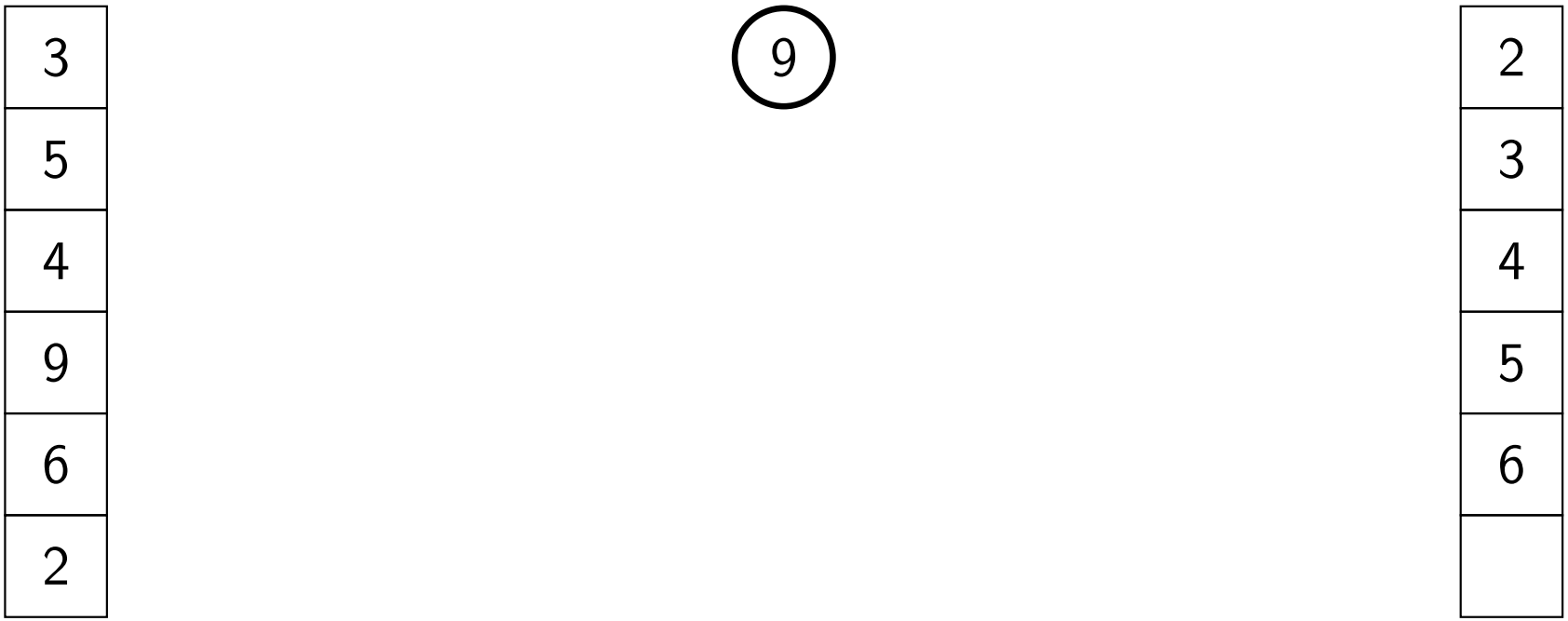


2
3
4
5

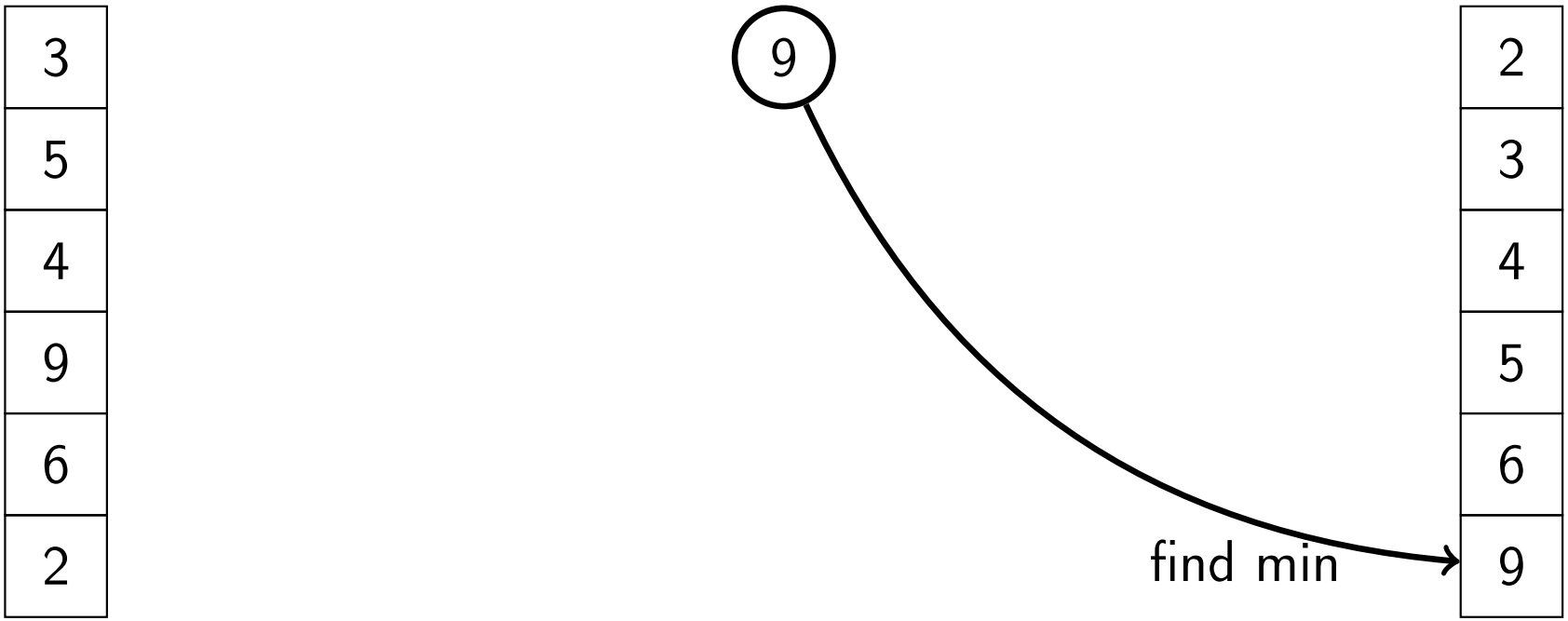
En detalle



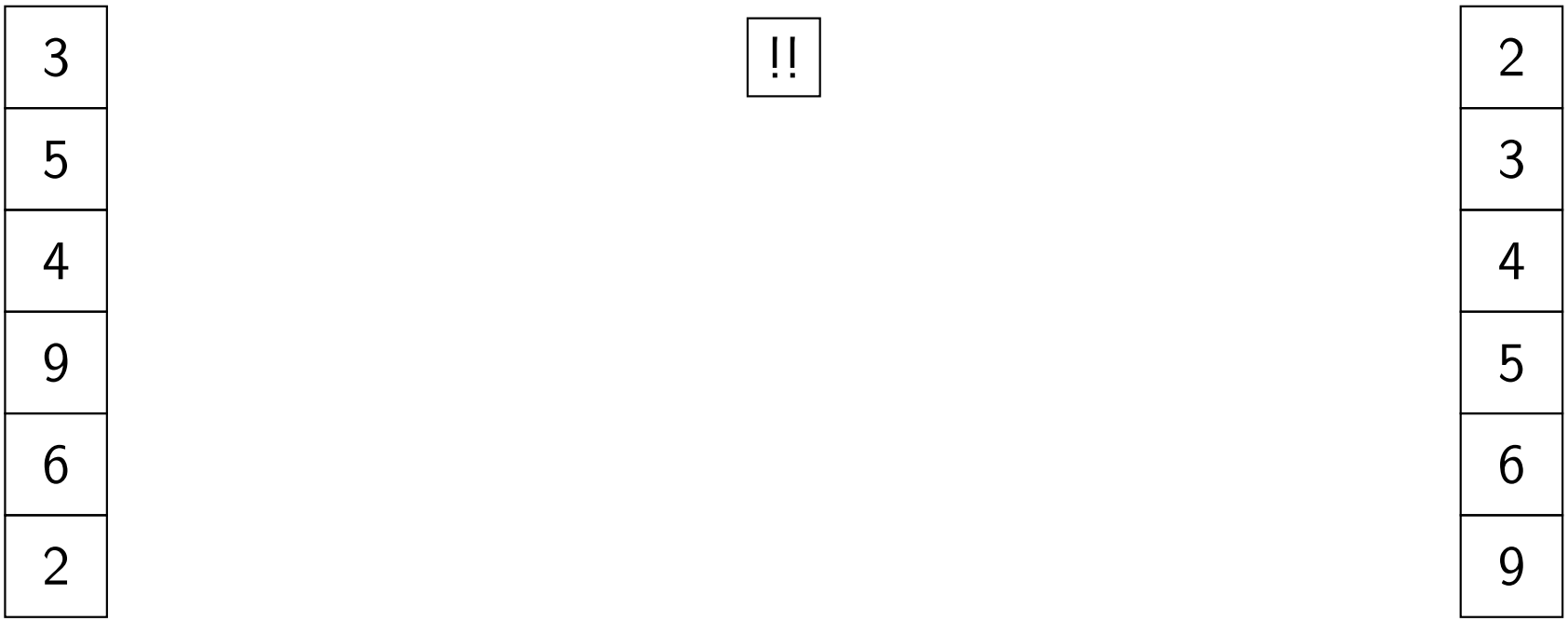
En detalle



En detalle



En detalle



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;
}
```

Comparación

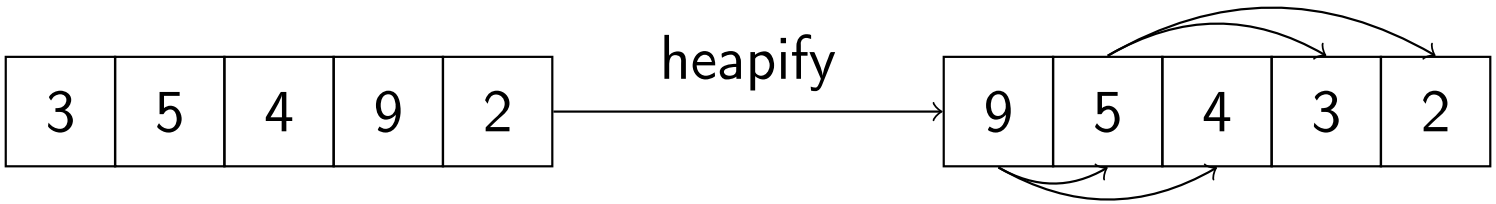
Podemos comparar a heapsort con los otros algoritmos de ordenamiento vistos:

- Heapsort tiene mejores tiempos (teóricos) en general que los ordenamientos por selección, burbuja e inserción.
- Heapsort tiene el mismo tiempo (teórico) que Quicksort.
- La implementación dada no es in-place, a diferencia de ordenamientos como burbuja o inserción.

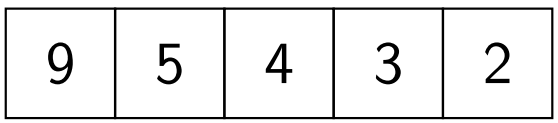
Mejorando los tiempos

En el caso de arreglos, puede ser conveniente utilizar una versión in-place del algoritmo:

- Dado un arreglo, armo un max-heap in-place



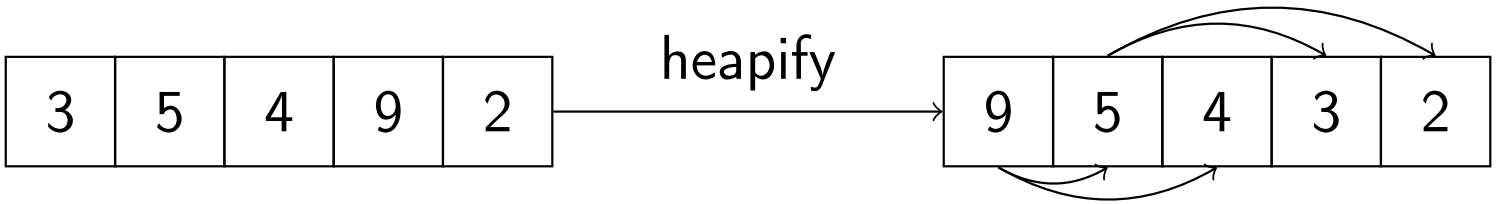
- Voy intercambiando el último elemento con la raiz (máximo del heap), y re-balanceando el heap:



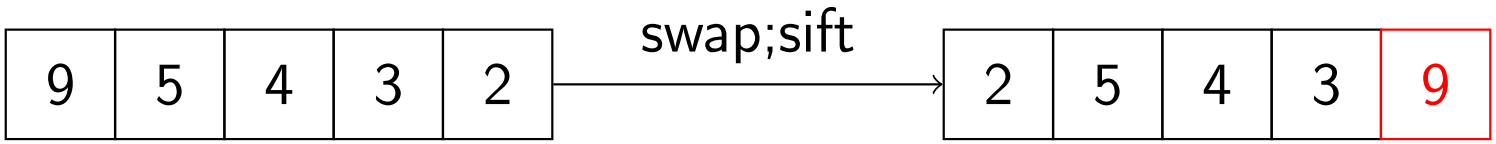
Mejorando los tiempos

En el caso de arreglos, puede ser conveniente utilizar una versión in-place del algoritmo:

- Dado un arreglo, armo un max-heap in-place



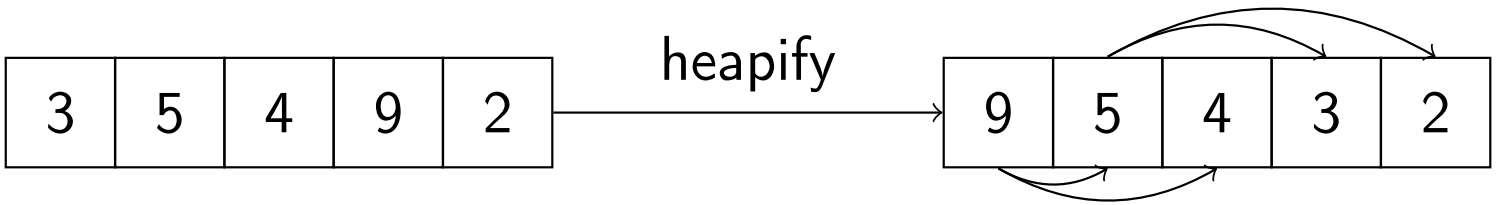
- Voy intercambiando el último elemento con la raiz (máximo del heap), y re-balanceando el heap:



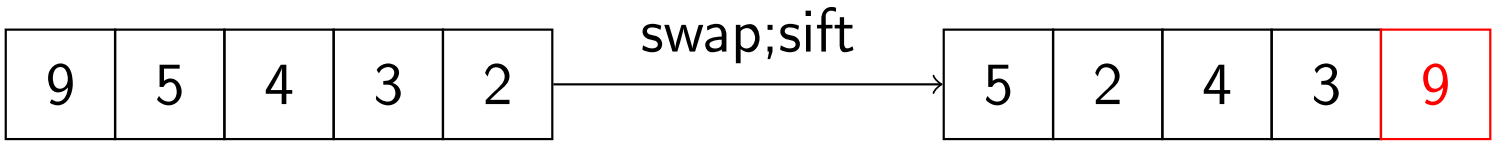
Mejorando los tiempos

En el caso de arreglos, puede ser conveniente utilizar una versión in-place del algoritmo:

- Dado un arreglo, armo un max-heap in-place



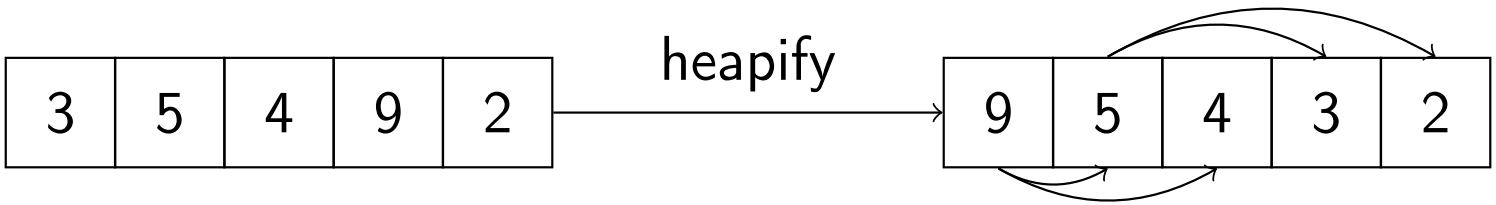
- Voy intercambiando el último elemento con la raiz (máximo del heap), y re-balanceando el heap:



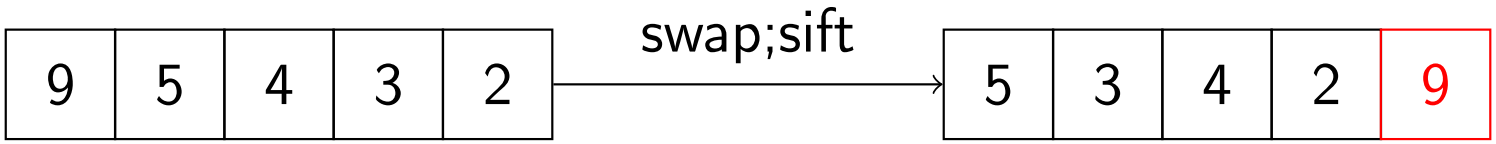
Mejorando los tiempos

En el caso de arreglos, puede ser conveniente utilizar una versión in-place del algoritmo:

- Dado un arreglo, armo un max-heap in-place



- Voy intercambiando el último elemento con la raiz (máximo del heap), y re-balanceando el heap:



Resumen

- heapsort = heap + sort
- Para ordenar una lista:
  - Insertar sus elementos en un heap
  - Encontrar el mínimo del heap, y guardarlo en una lista
  - Borrar el mínimo del heap, y volver al pasa anterior hasta vaciar el heap
- El algoritmo queda impuesto por la estructura intermedia usada (heap).