UTEC
UNIVERSIDAD DE INGENIERÍA
Y TECNOLOGÍA

Welcome to Algorithms and Data Structures! - CS2100

Heaps

Muchas veces necesitamos un acceso rápido al elemento más grande o pequeño de un conjunto de datos.

Por ejemplo:

- Para gestionar una cola de prioridades
- Ordenar elementos de manera eficiente
- Obtener el K elemento más pequeño o grande de un conjunto

¿Cómo se les ocurre que podríamos obtener el elemento más grande o pequeño de un arreglo de números?

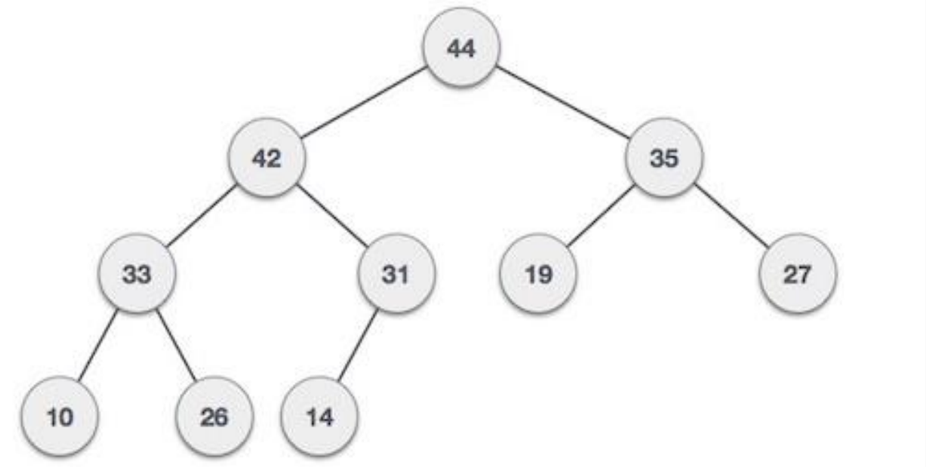
Heaps

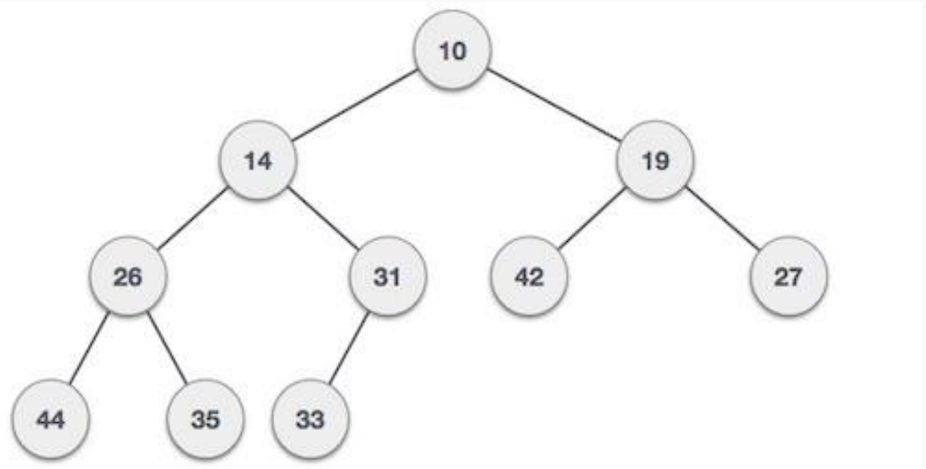
Es una estructura de datos tipo árbol binario que satisface ciertas propiedades:

- Es un árbol completo (semi-completo)
- Satisface la propiedad del heap:
  - Para todo nodo:
    - Si es **max-heap** entonces el padre es mayor o igual.
    - Si es **min-heap** entonces el padre es menor o igual.

En los ejemplos de la derecha

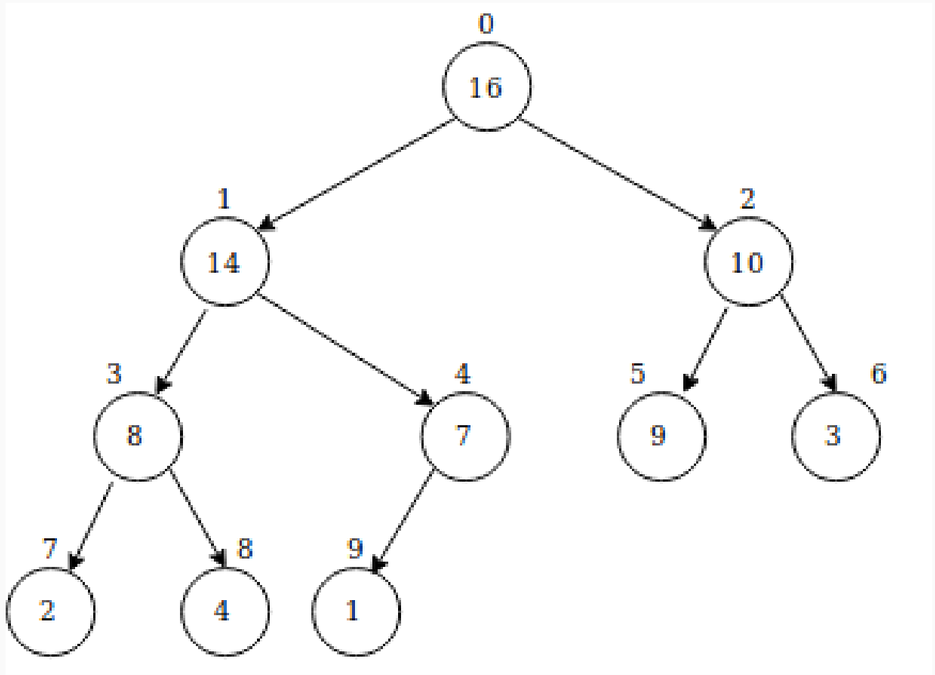
- ✓ **Min heap:** El root siempre es el elemento menor
- ✓ **Max heap:** El root siempre es el elemento mayor





Heaps

¿Cómo representamos un Heap?



Como un árbol binario

0	1	2	3	4	5	6	7	8	9
16	14	10	8	7	9	3	2	4	1

Como un array

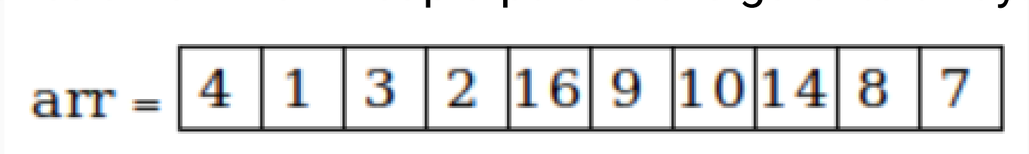
Left(i) = 2 * i + 1
Right(i) = 2 * i + 2
Parent(i) = (i – 1) / 2

Heaps

¿Cómo construir un Heap a partir de un array?

- Construimos un árbol binario inicial directamente del array.
- Luego, para cada nodo padre aplicar un **heapify-down**. Esta función, dependiendo del tipo de heap, cambia al mayor/menor de los hijos con el padre.

Ejemplo, construir un Max-Heap a partir del siguiente array.

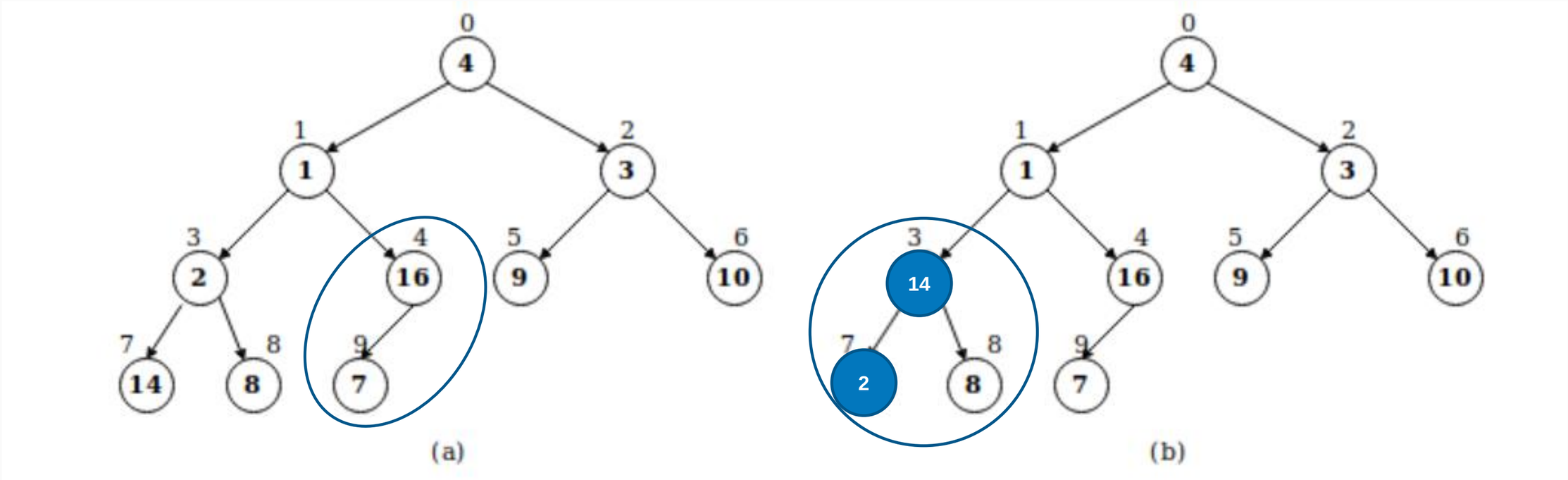


Left(i) = 2 * i + 1
Right(i) = 2 * i + 2
Parent(i) = (i – 1) / 2

Heaps

arr =	4	1	3	2	16	9	10	14	8	7

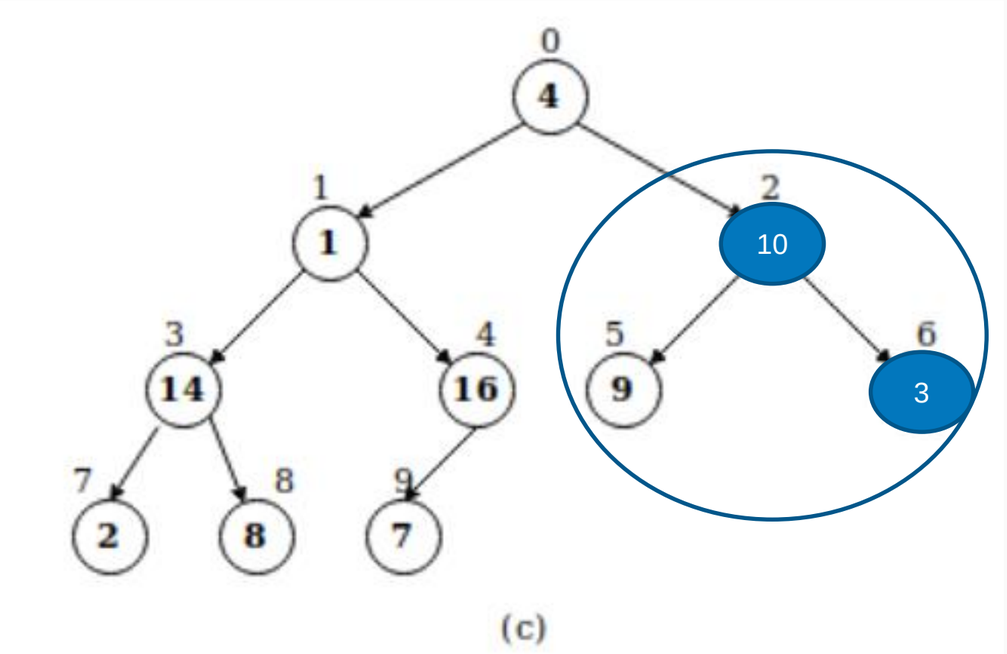
Construcción de un Heap



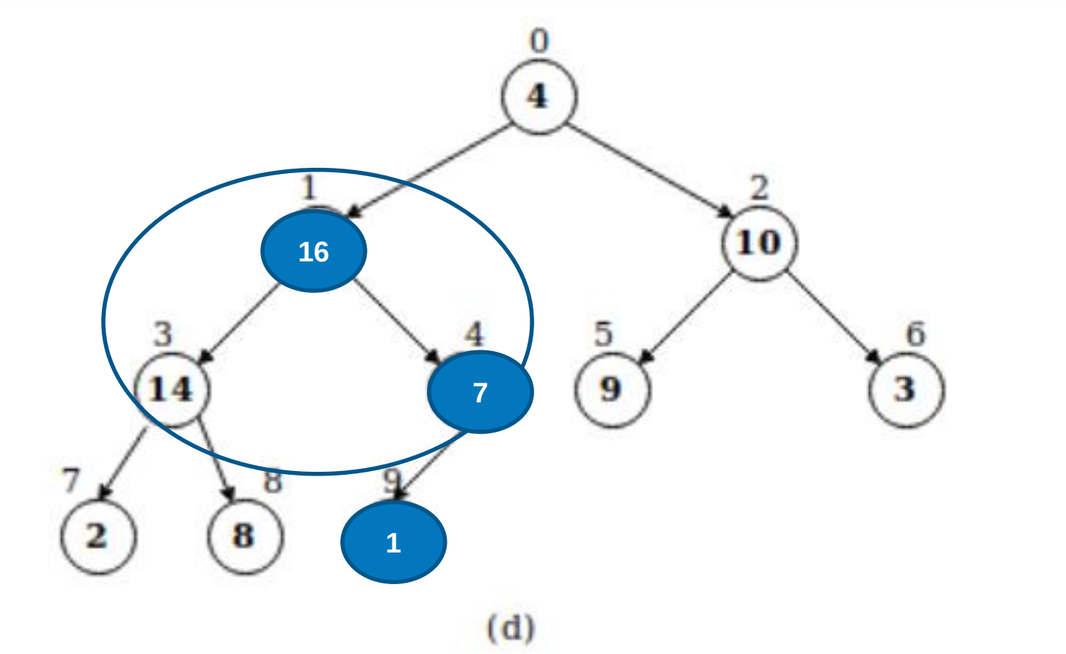
Heaps

arr =	4	1	3	2	16	9	10	14	8	7

Construcción de un Heap



(c)

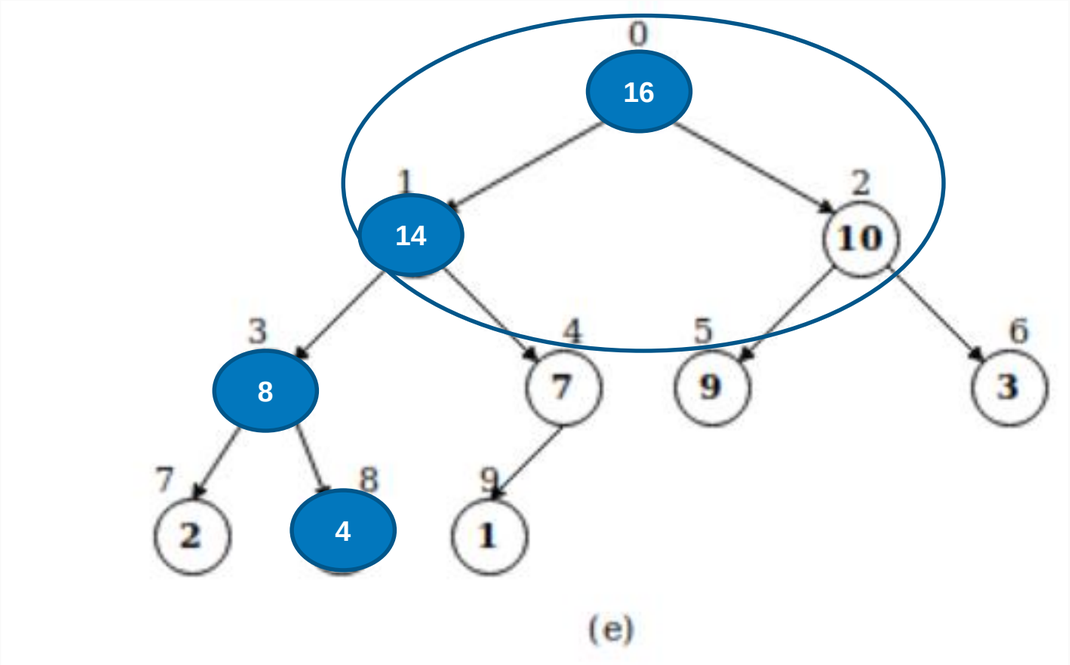


(d)

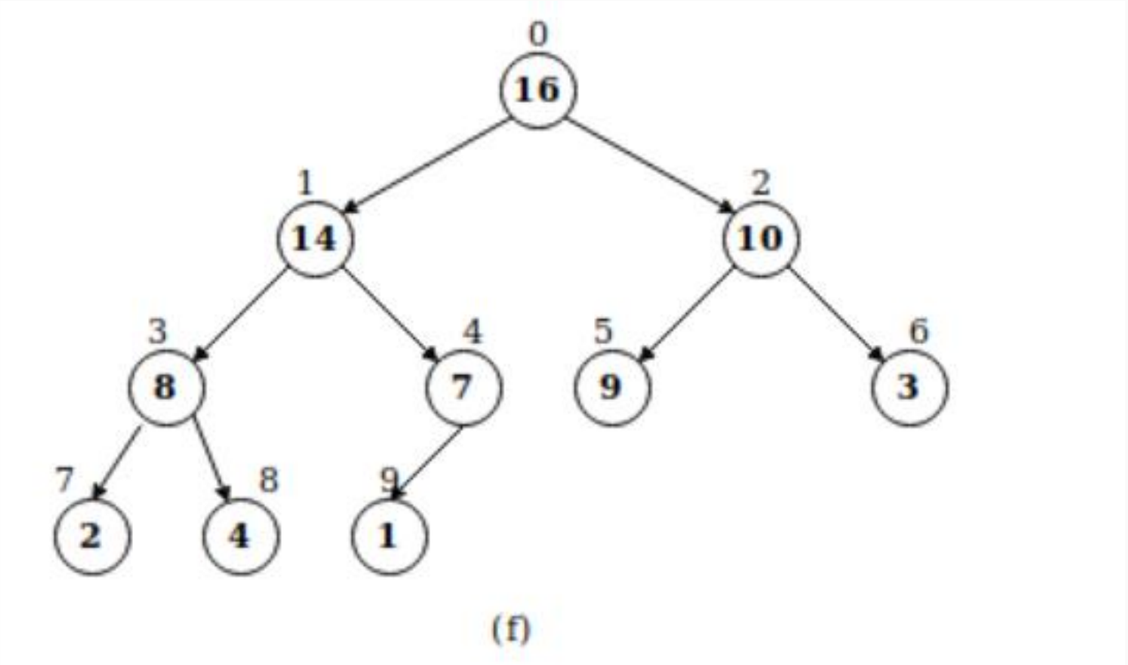
Heaps

arr =	4	1	3	2	16	9	10	14	8	7

Construcción de un Heap



(e)



(f)

16	14	10	8	7	9	3	2	4	1

Heaps

Construcción de un Heap

¿A cuantos nodos se aplica el heapify-down?

Como el árbol es completo, el heapify se va a llamar n/2 veces.

El algoritmo final en el peor de los casos tiende a O(n log n).

Si embargo, en la mayoría de casos tiende a O(n), ya que heapify no siempre toma O(log n).

Heaps

Ejercicio1:

Creen un max-heap y un min-heap para el siguiente arreglo:

23, 10, 49, 50, 13, 12, 9, 45, 33, 17, 2, 21, 6

Heaps

Ejercicio2:

Creen un max-heap y un min-heap para el siguiente arreglo:

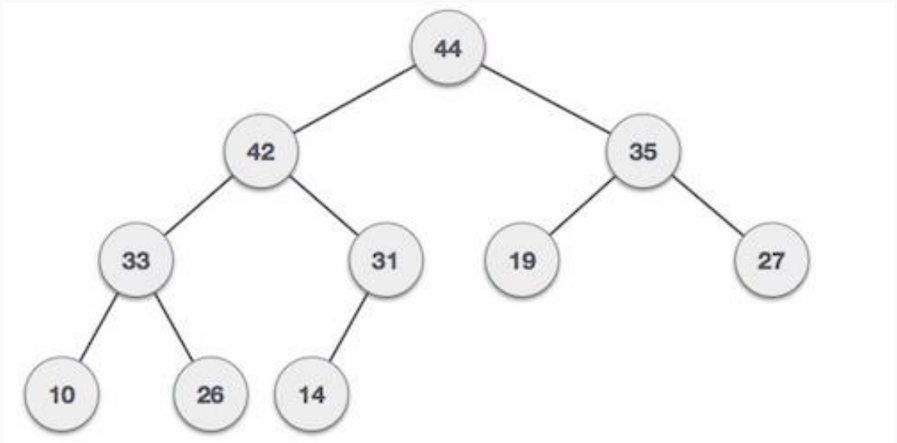
70, 12, 15, 17, 66, 54, 48, 59, 24, 13, 33, 61, 4

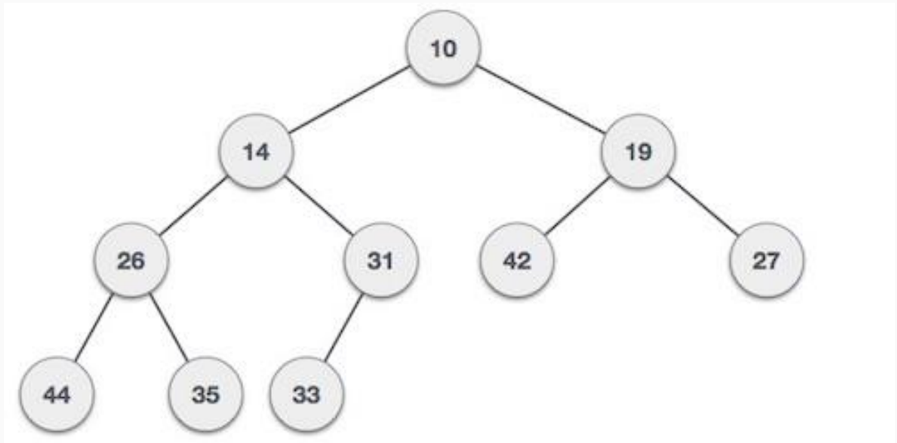
Heaps

¿Cómo se inserta un elemento en un heap?

- Se inserta siempre en el siguiente espacio vacío.
- Luego, sobre dicho nodo se realiza un **heapify-up**. Esta función compara el nodo con su padre hasta encontrar su posición correcta.

Por ejemplo: Insertar 38 en el max heap, y 5 en el min heap



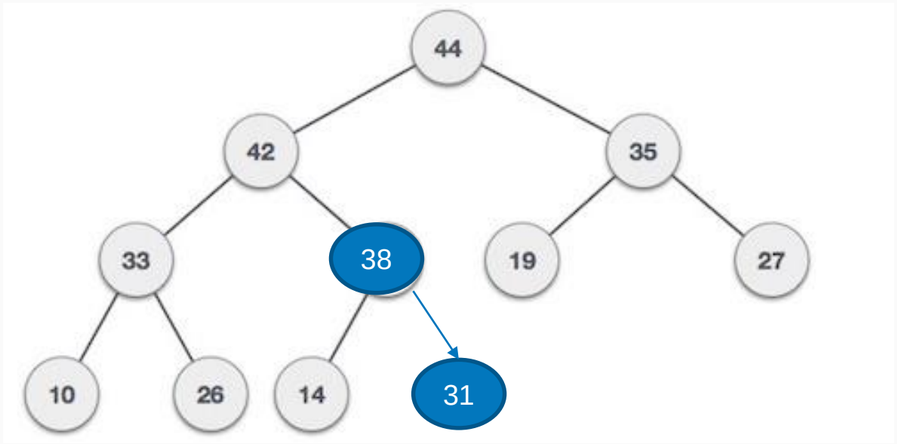


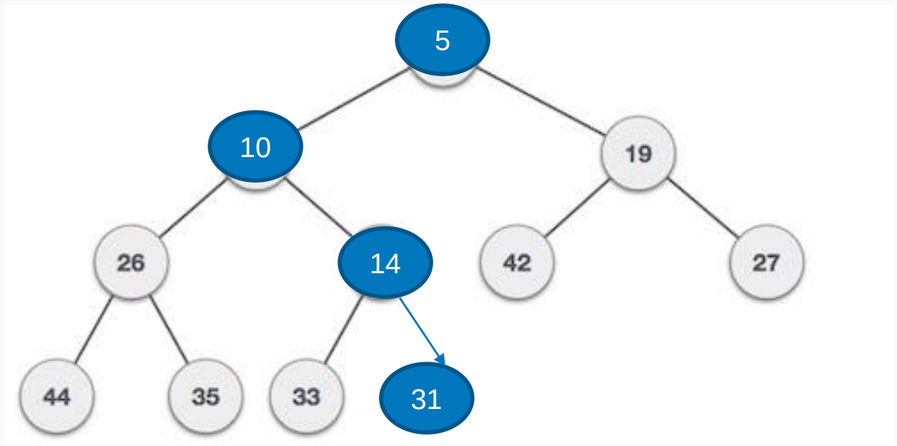
Heaps

¿Cómo se inserta un elemento en un heap?

- Se inserta siempre en el siguiente espacio vacío.
- Luego, sobre dicho nodo se realiza un **heapify-up**. Esta función compara el nodo con su padre hasta encontrar su posición correcta.

Por ejemplo: Insertar 38 en el max heap, y 5 en el min heap

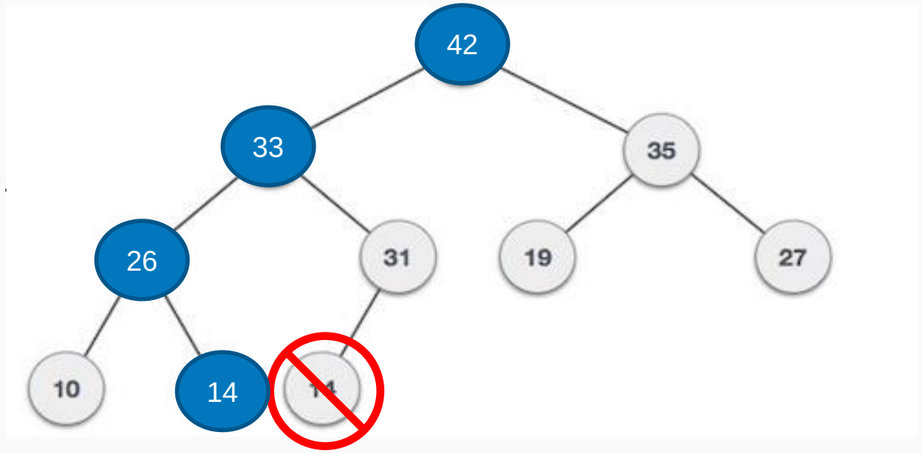




Heaps

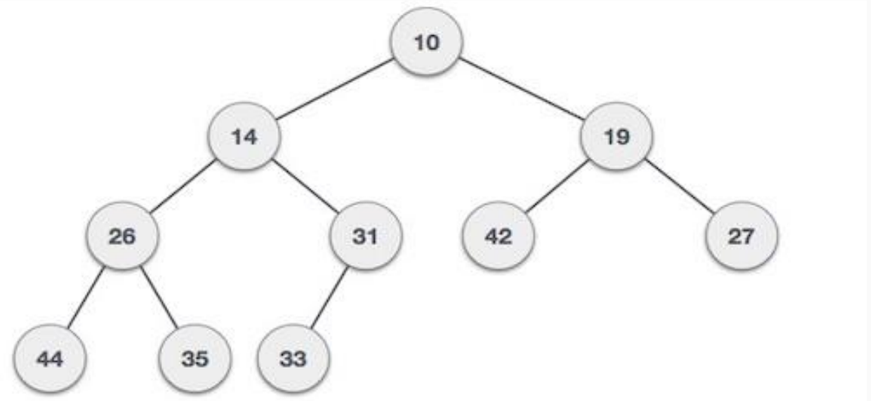
¿Cómo haríamos el pop?

- Swap del root (0) con el último (n-1) y se remueve el último (n--).
- Se ubica el nuevo valor del root en su posición correcta usando **heapify-down.** Esta función compara y cambia (de ser necesario) al nodo con alguno de sus hijos.



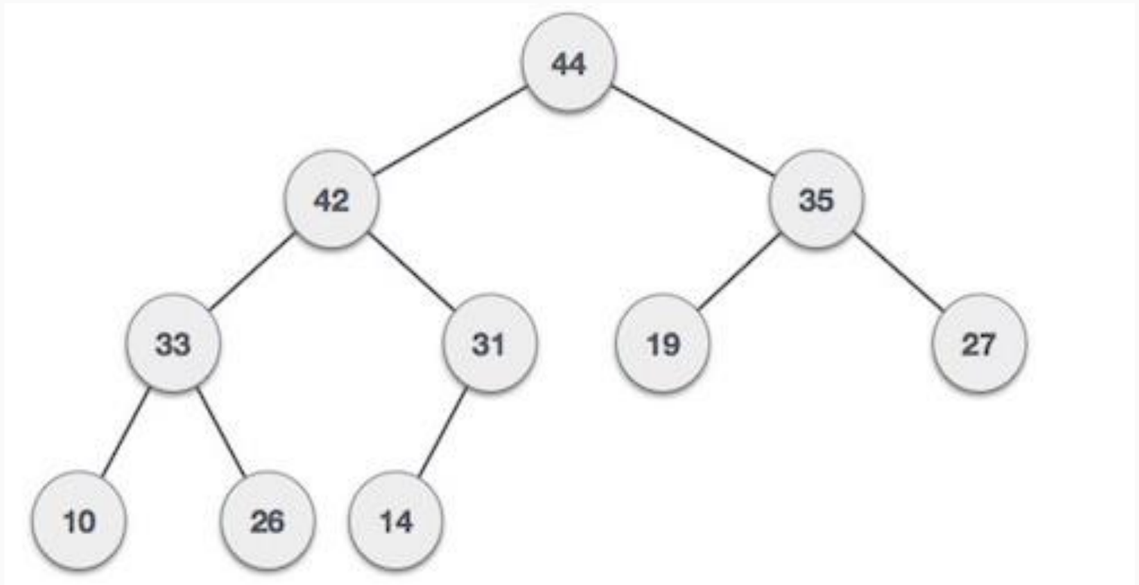
En el caso de un max, se cambiará con el hijo mayor. Min cambiará con el hijo menor. Al hacer un cambio se llamará recursivamente a la función.

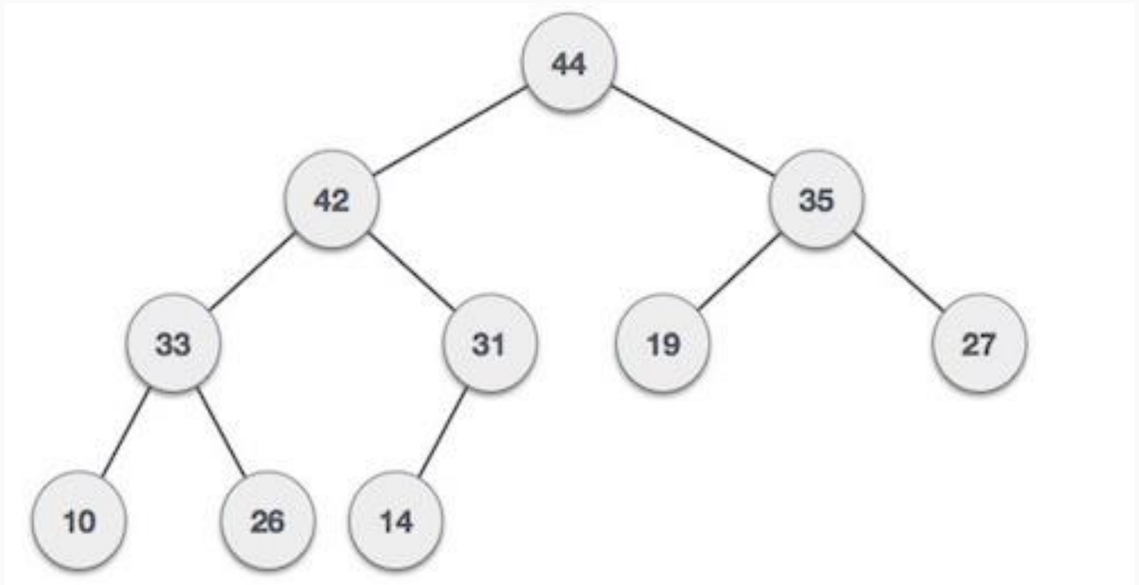
Por ejemplo: aplicar pop en los heaps de las imágenes de la derecha.



Heaps

POP en un Max-Heap

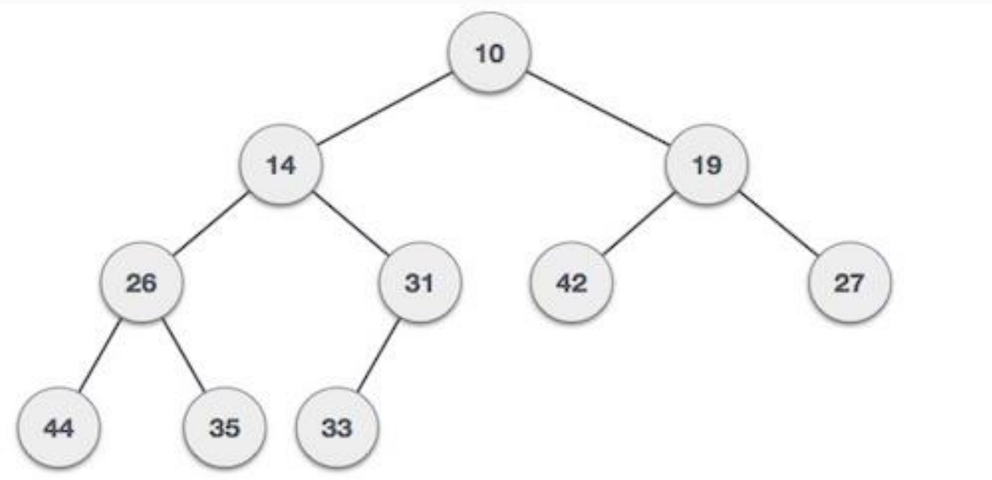


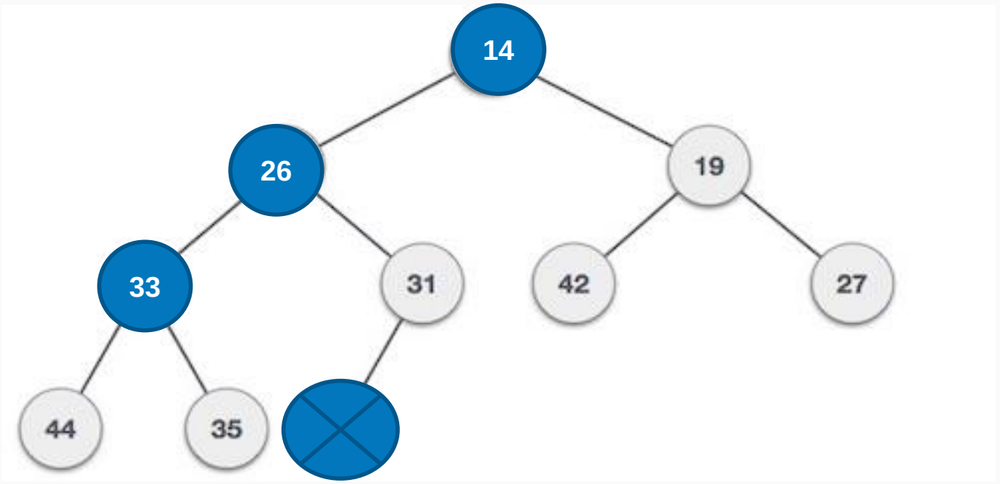


- Swap del root (0) con el último (n-1) y se remueve el último (n--).
- Se ubica el nuevo valor del root en su posición correcta usando **heapify-down.** Esta función compara y cambia (de ser necesario) al nodo con alguno de sus hijos.

Heaps

POP en un Min-Heap

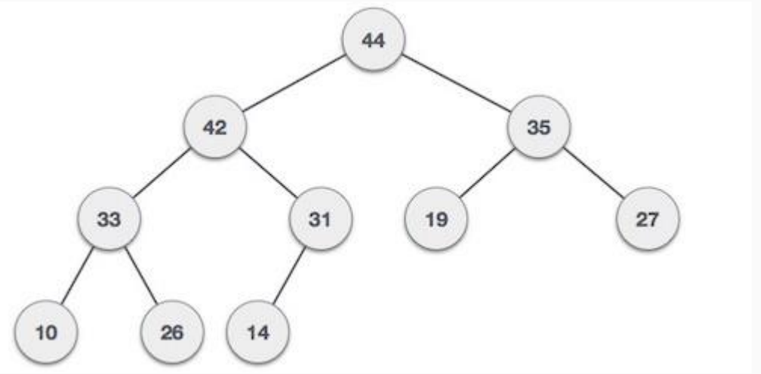




- Swap del root (0) con el último (n-1) y se remueve el último (n--).
- Se ubica el nuevo valor del root en su posición correcta usando **heapify-down.** Esta función compara y cambia (de ser necesario) al nodo con alguno de sus hijos.

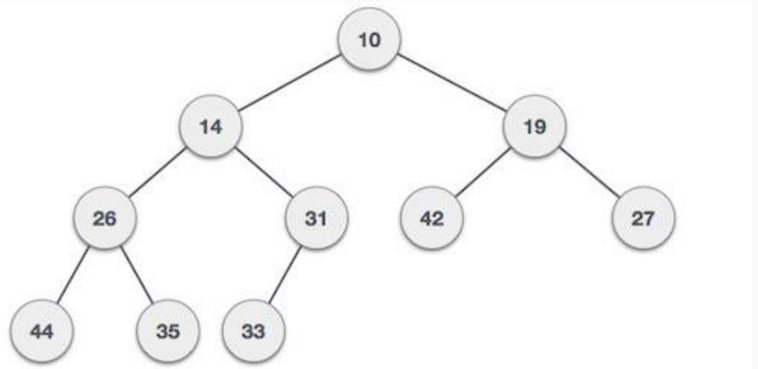
Heaps

¿Cómo están almacenados los nodos en el array?



¿Cómo podríamos saber si un nodo tiene hijo izquierdo o hijo derecho?

Bastaría con validar que el índice del hijo sea menor al tamaño del array

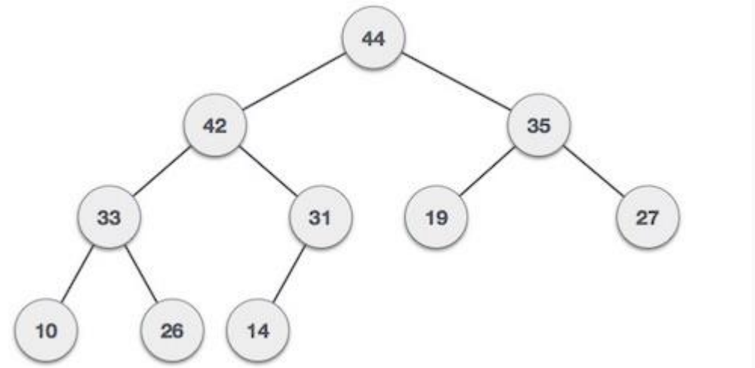


Heaps

¿Cuáles son los tiempos de ejecución para sus funciones?

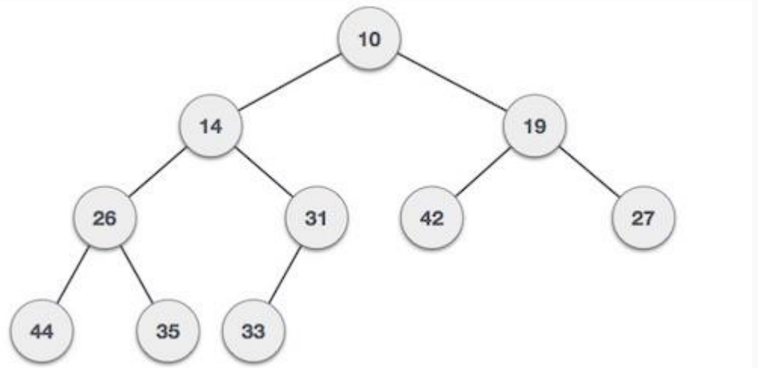
insertar: O(log n)
remover: O(log n)
construir el heap: O(n log n)
encontrar el máximo o mínimo: O(1)

Nota: Un árbol completo, la altura es aproximadamente log(n).



¿Cuáles serían los usos de los heaps?

Colas de prioridad, scheduling, prim o kruskal, ordenamientos, etc.



Heapsort

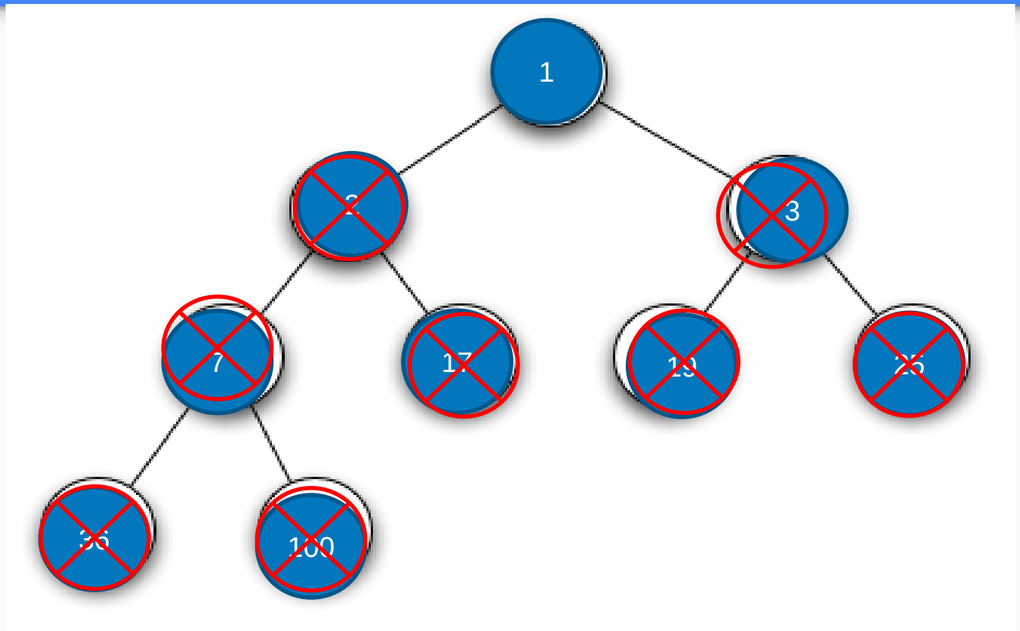
¿Cómo usarían los heaps para ordenamiento?

Sea el array:

36	3	25	2	19	100	1	17	7

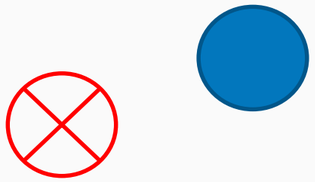
Ordenar de forma ascendente usando Max-Heap

- Construir un max-heap
- Aplicar n operaciones de pop
  - Cada elemento que se extrae se deja en el final del array



n = 1

1	2	3	7	17	19	25	36	100
0	1	2	3	4	5	6	7	8



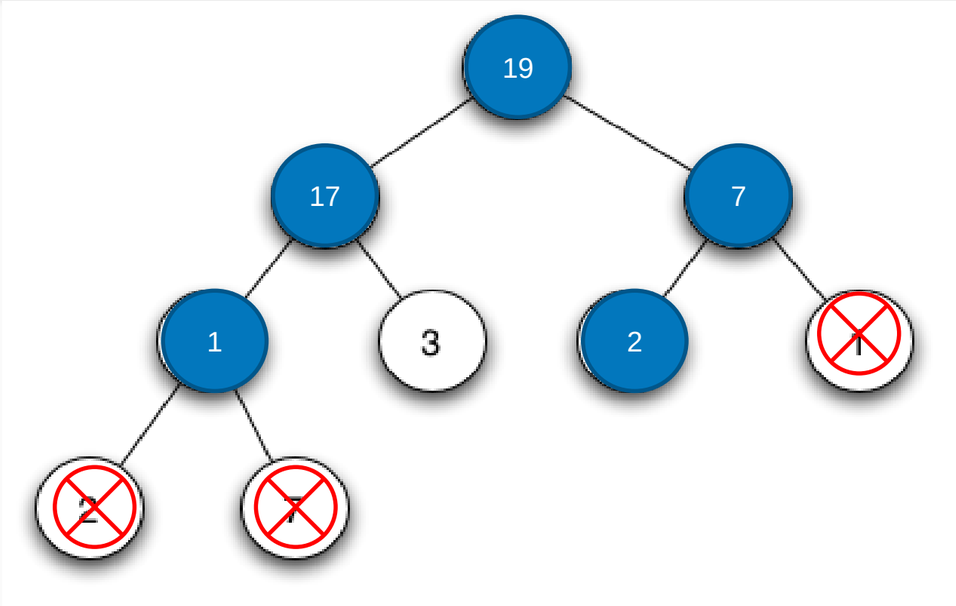
Heapsort

¿Cómo usarían los heaps para ordenamiento?

Sea el array:

36	3	25	2	19	100	1	17	7

Ordenar de forma ascendente usando Max-Heap



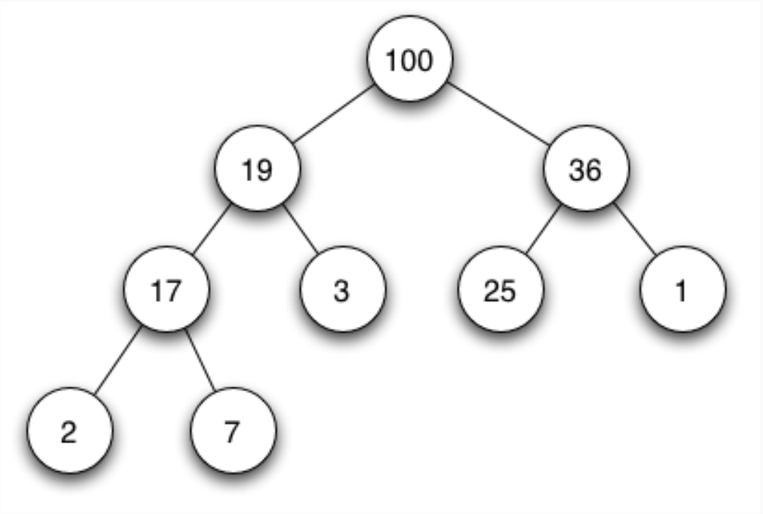
n = 5

2	17	7	1	3	19	25	36	100
0	1	2	3	4	5	6	7	8

Heapsort

¿Cómo usarían los heaps para ordenamiento?

Primero construímos un max-heap desde el array **O(n),** por ejemplo de: 36, 3, 25, 2, 19, 100, 1, 17, 7 → **100, 19, 36, 17, 3, 25, 1, 2, 7**



100, 19, 36, 17, 3, 25, 1, 2, 7

Segundo, removemos el primer elemento del max-heap y lo cambiamos por la última posición **O(1)**, quedando el siguiente array:

7, 19, 36, 17, 3, 25, 1, 2, 100

Tercero, volvemos a armar el heap sin el último elemento **O(log n)**, y se continúa hasta que solo quede un elemento en el heap **O(n)**:

36, 19, 25, 17, 3, 7, 1, 2, 100 → … → 1, 2, 3, 7, 17, 19, 25, 36, 100

Heapsort

Ejercicios:

Ordene los siguientes elementos utilizando heapsort, explique el proceso

- 13, 75, 79, 54, 2, 3 (de mayor a menor)

  79, 75, 13, 54, 2, 3

- 13, 75, 79, 54, 2, 3 (de menor a mayor)

  2, 15, 3, 54, 75, 79

UTEC
UNIVERSIDAD DE INGENIERÍA
Y TECNOLOGÍA

Welcome to Algorithms and Data Structures! - CS2100