Montículos Binarios

AED II

2020

UBP UNIVERSIDAD BLAS PASCAL

montículo binario

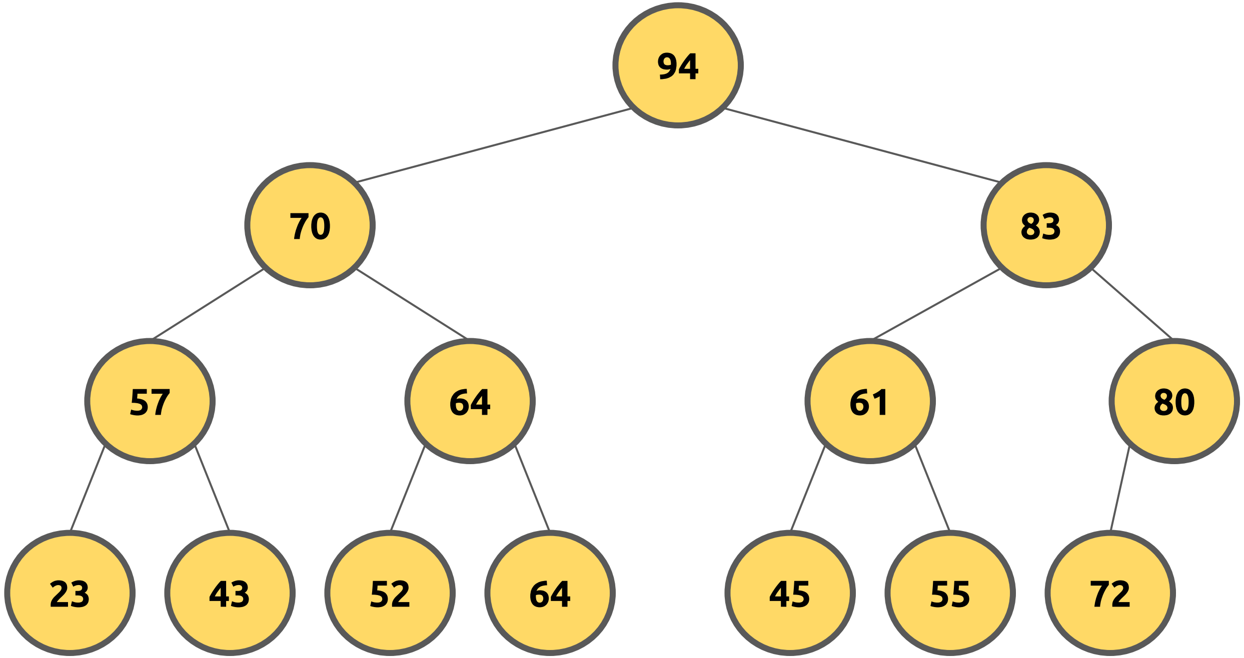
Un **montículo binario** es un árbol binario con las siguientes restricciones:

1. Cada nodo padre es siempre mayor o igual (max-heap) (maximal) o menor o igual (min-heap) (minimal) que sus descendientes
2. Es un árbol **completo**

Otras características que tiene como consecuencia de su definición son:

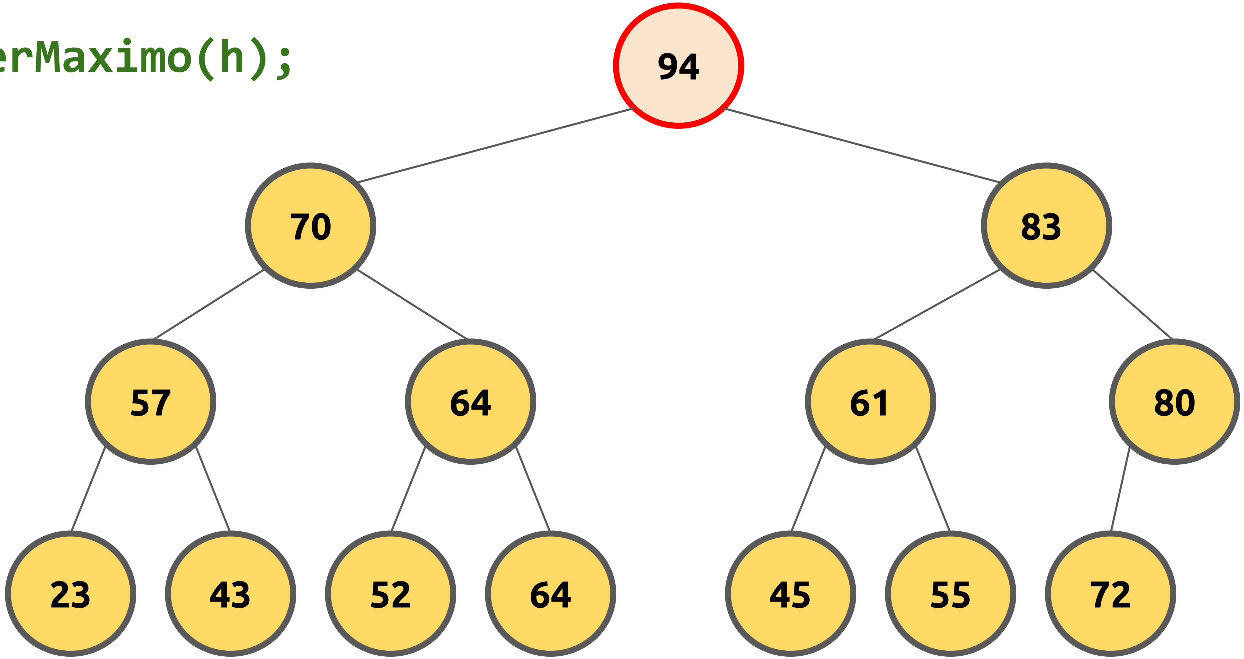
- Admite repeticiones
- Puede ser implementado en un array

montículo binario: ejemplo



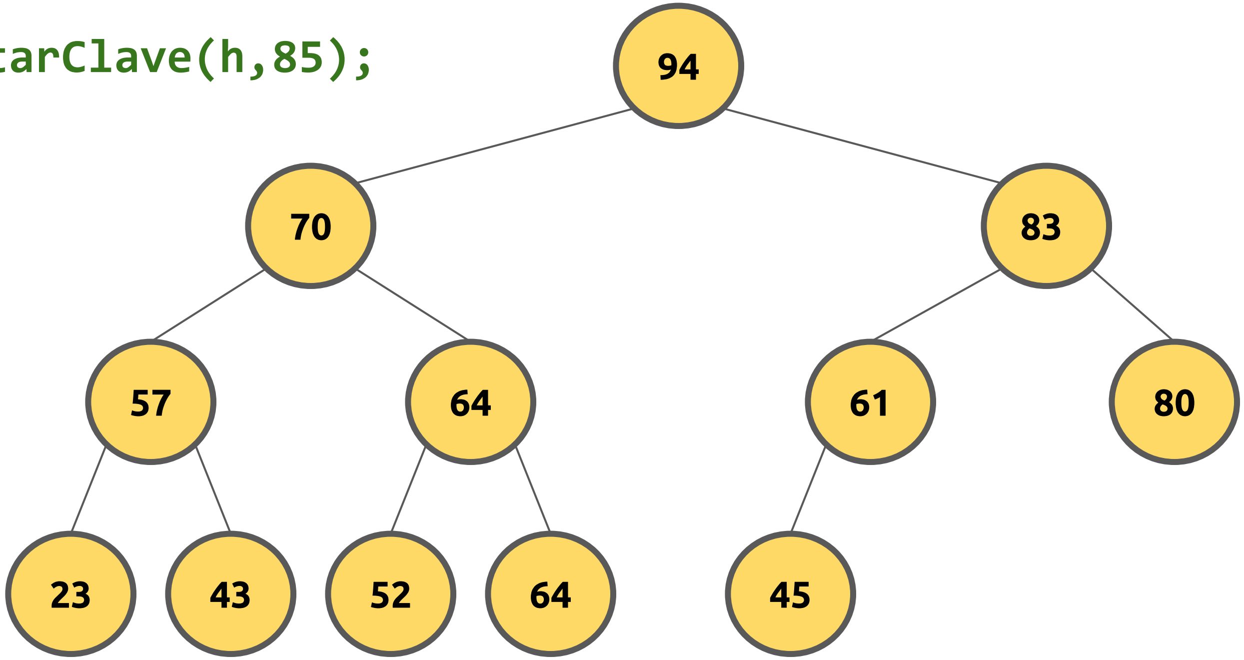
operaciones: obtener el máximo

```
obtenerMaximo(h);
```



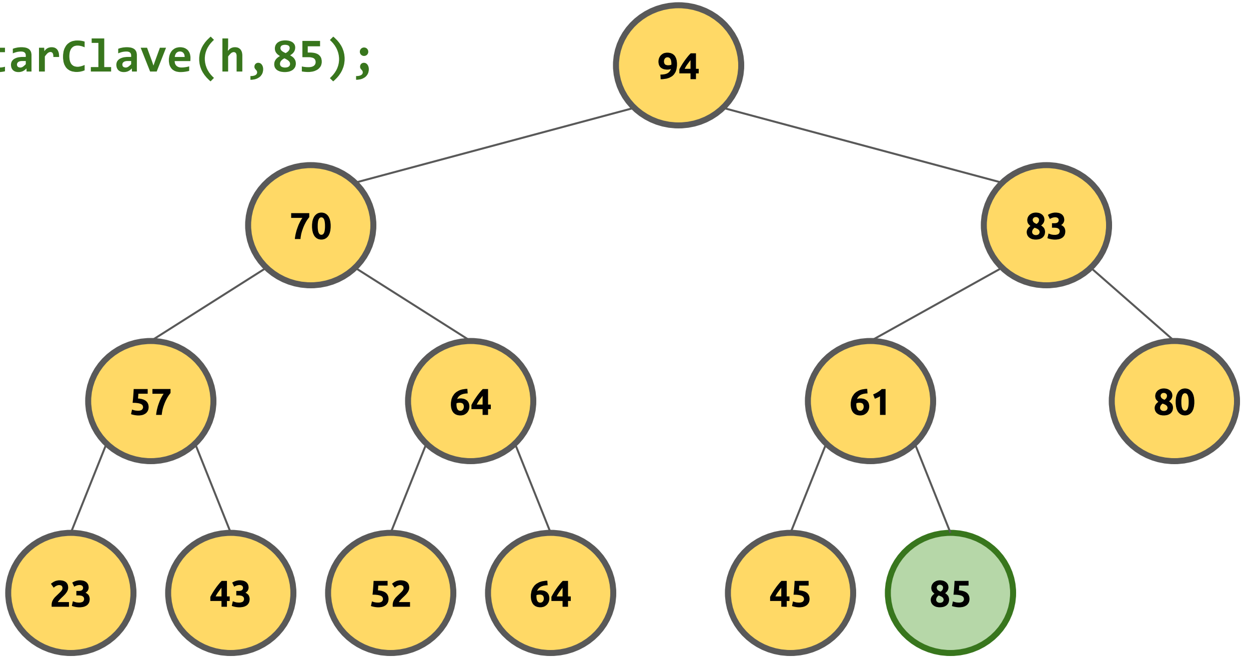
operaciones: insertar

```
insertarClave(h,85);
```



operaciones: insertar

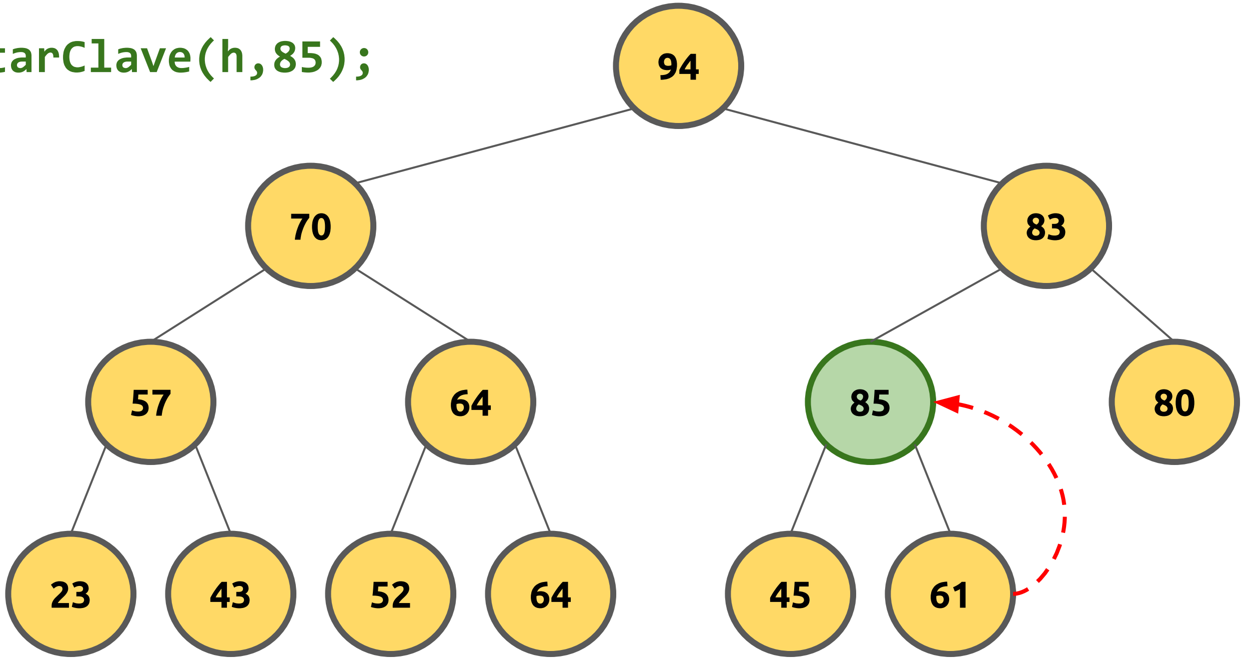
```
insertarClave(h,85);
```



1. Insertamos el elemento en el primer espacio libre a la izquierda

operaciones: insertar

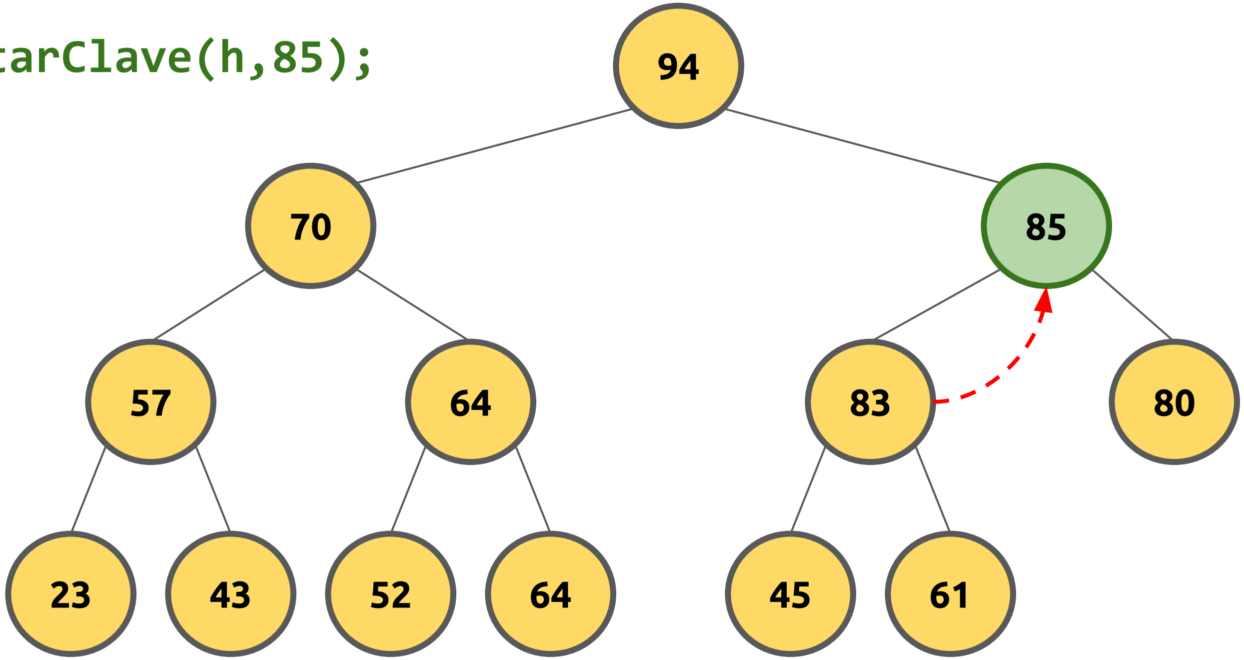
```
insertarClave(h,85);
```



1. Insertamos el elemento en el primer espacio libre a la izquierda
2. Lo hacemos "flotar" hasta que encuentra su lugar

operaciones: insertar

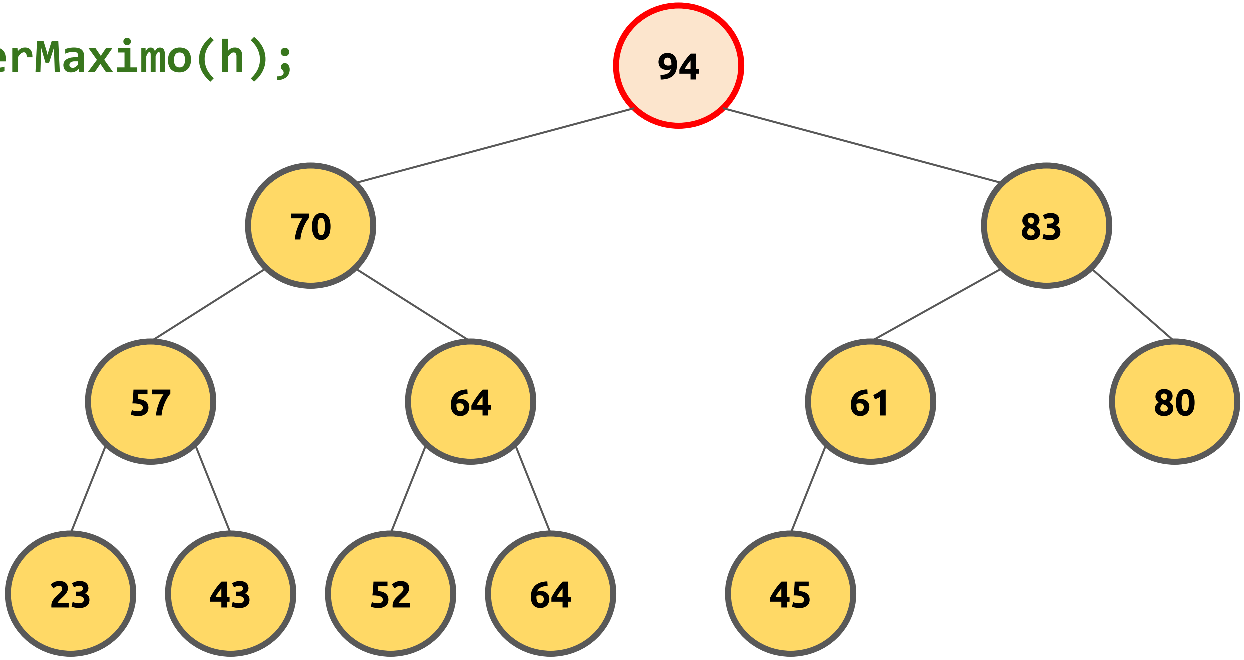
```
insertarClave(h,85);
```



1. Insertamos el elemento en el primer espacio libre a la izquierda
2. Lo hacemos "flotar" hasta que encuentra su lugar

operaciones: extraer el mayor

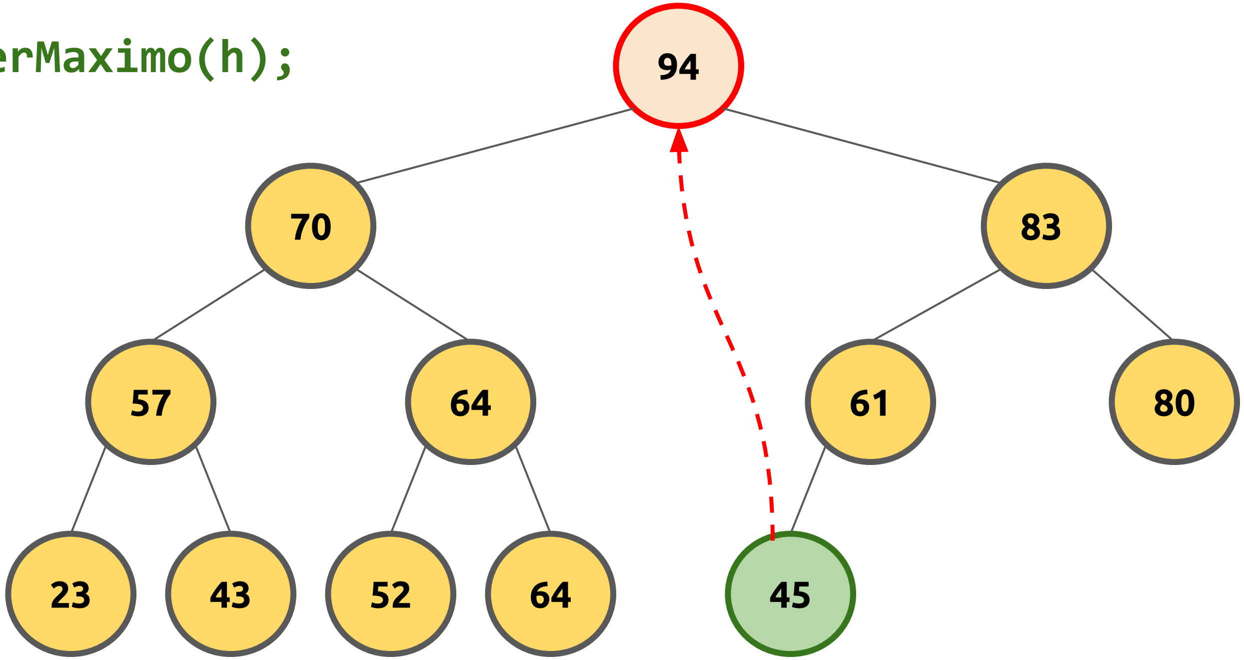
```
extraerMaximo(h);
```



1. Remplazamos el mayor por el "último" nodo (el que está más abajo y a la derecha)
2. Lo hundimos hasta que encuentra su lugar

operaciones: extraer el mayor

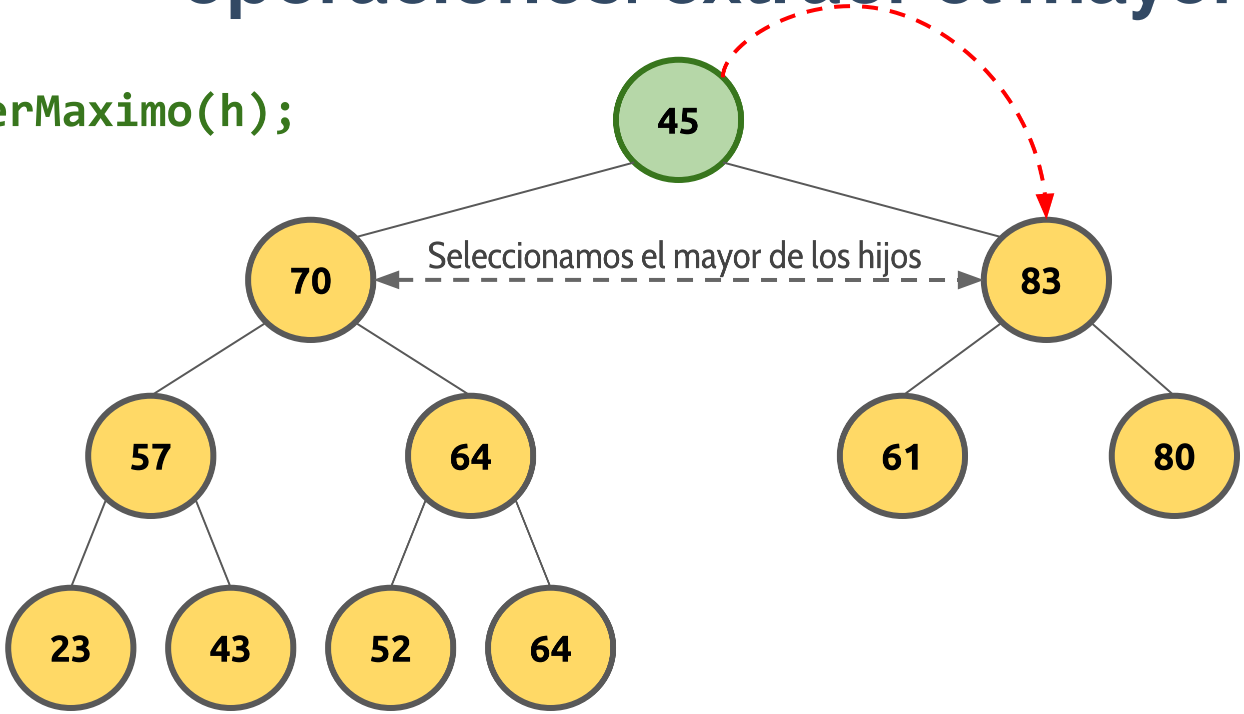
```
extraerMaximo(h);
```



1. Remplazamos el mayor por el "último" nodo (el que está más abajo y a la derecha)
2. Lo hundimos hasta que encuentra su lugar

operaciones: extraer el mayor

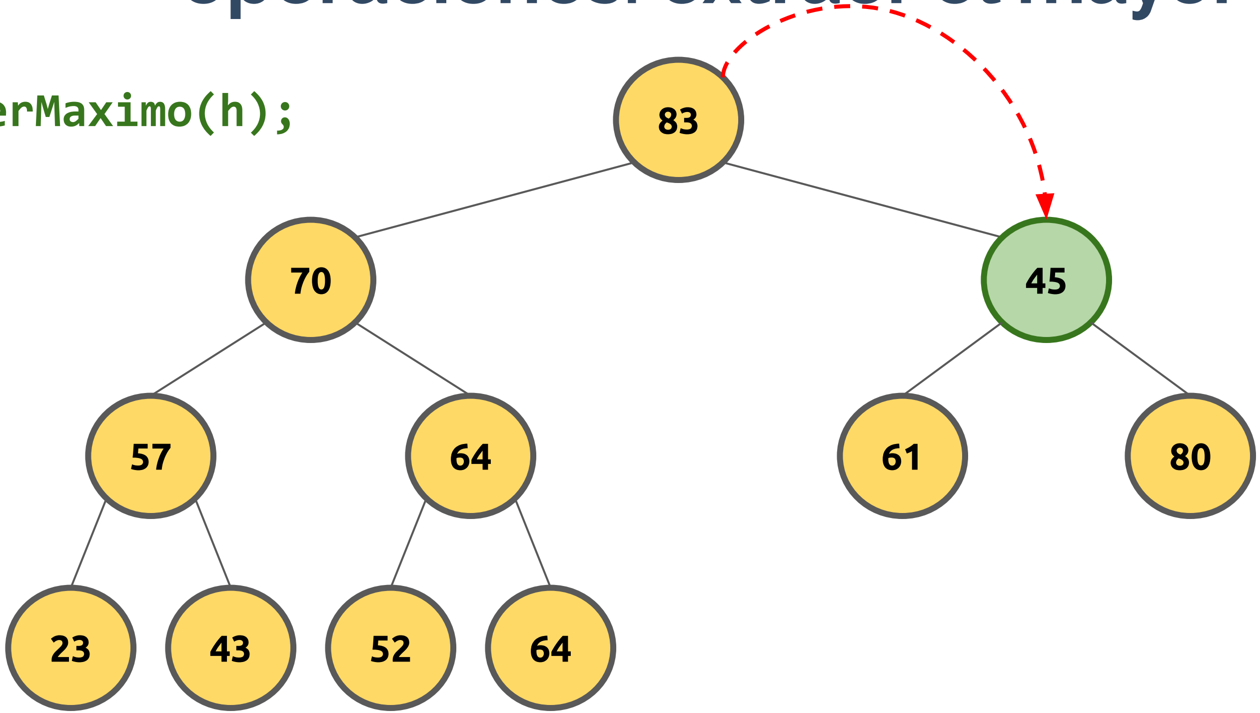
```
extraerMaximo(h);
```



1. Remplazamos el mayor por el "último" nodo (el que está más abajo y a la derecha)
2. Lo hundimos hasta que encuentra su lugar

operaciones: extraer el mayor

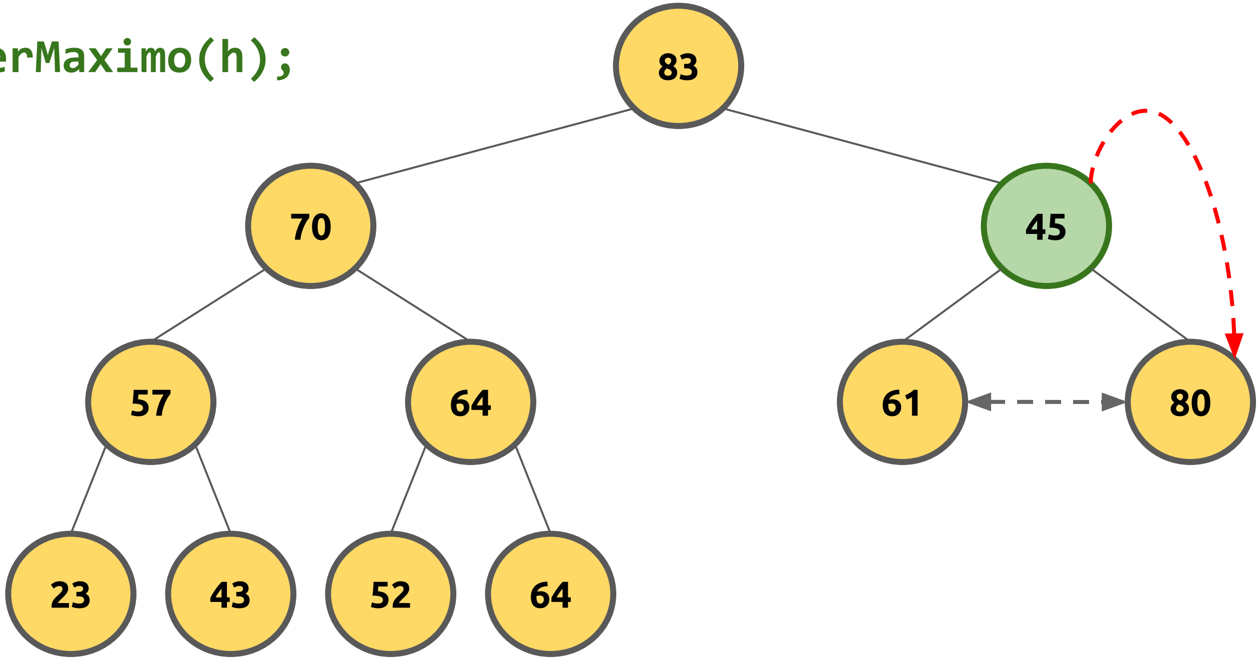
```
extraerMaximo(h);
```



1. Remplazamos el mayor por el "último" nodo (el que está más abajo y a la derecha)
2. Lo hundimos hasta que encuentra su lugar

operaciones: extraer el mayor

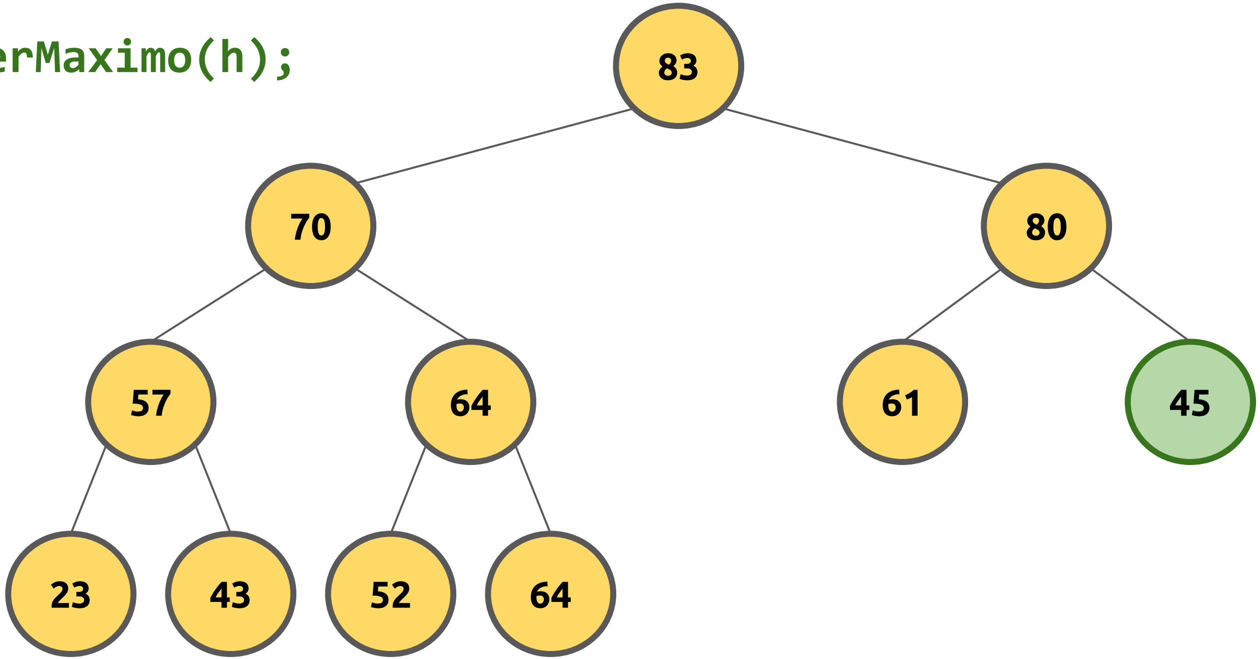
```
extraerMaximo(h);
```



1. Remplazamos el mayor por el "último" nodo (el que está más abajo y a la derecha)
2. Lo hundimos hasta que encuentra su lugar

operaciones: extraer el mayor

```
extraerMaximo(h);
```



1. Remplazamos el mayor por el "último" nodo (el que está más abajo y a la derecha)
2. Lo hundimos hasta que encuentra su lugar

montículo binario: complejidad

En el peor caso:

Encontrar el **máximo** (mínimo): $O(1)$

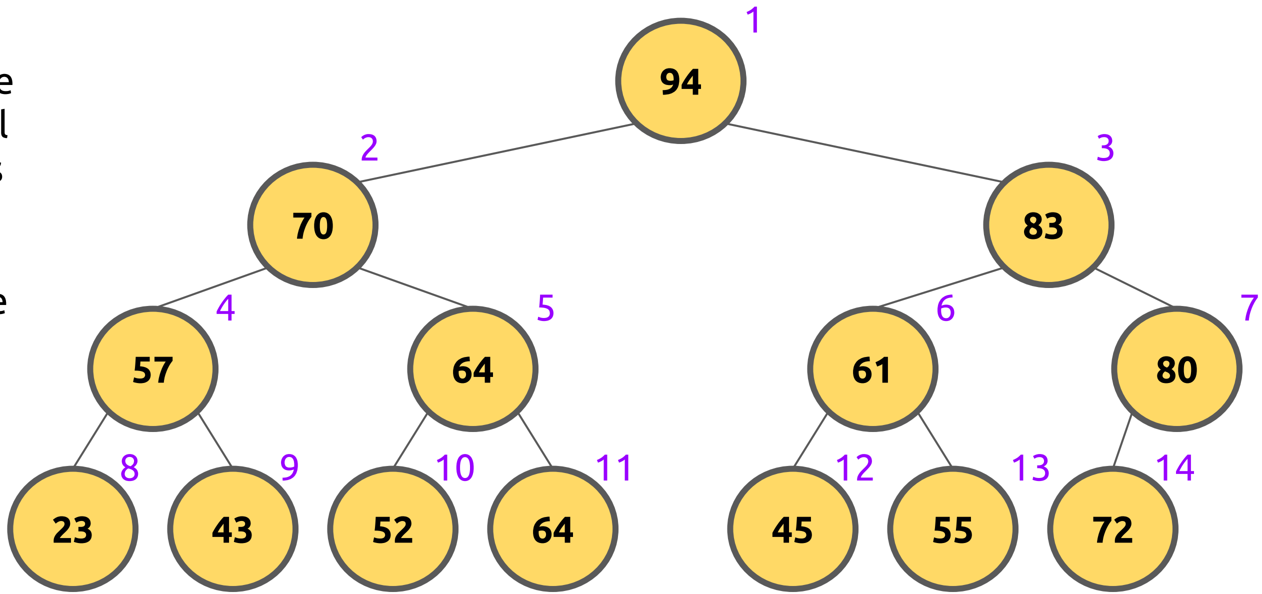
Insertar un elemento (si hay espacio en el array): $O(\log n)$

Eliminar un elemento: $O(\log n)$

implementación en un array

Los hijos del elemento que se encuentra en la posición **i** del arreglo, se encuentran en las posiciones **2i y 2i + 1**.

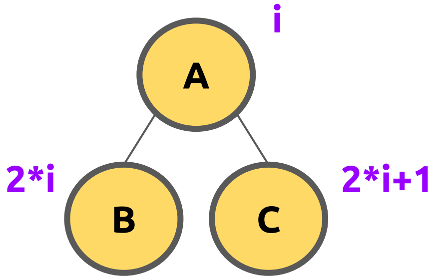
El padre del elemento que se encuentra en la posición i**,** se encuentra en la posición **i / 2**.



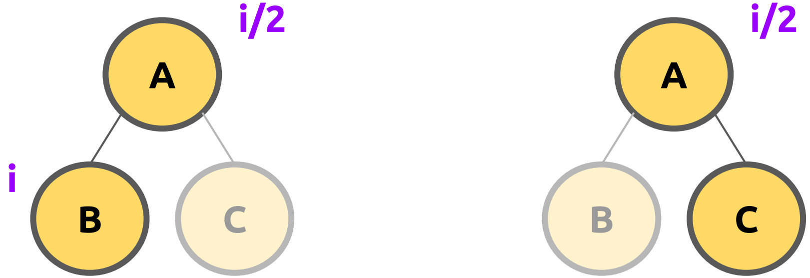
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
94	70	83	57	64	61	80	23	43	52	64	45	55	72	

implementación en un array

Los hijos del elemento que se encuentra en la posición i del arreglo, se encuentran en las posiciones **2i y 2i + 1.**

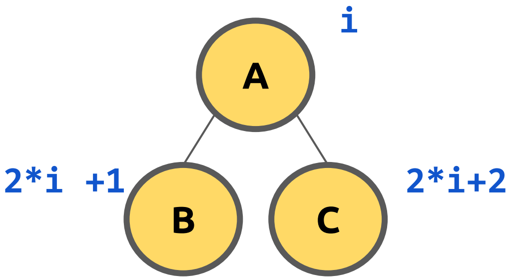


El padre del elemento que se encuentra en la posición i, se encuentra en la posición **i / 2**

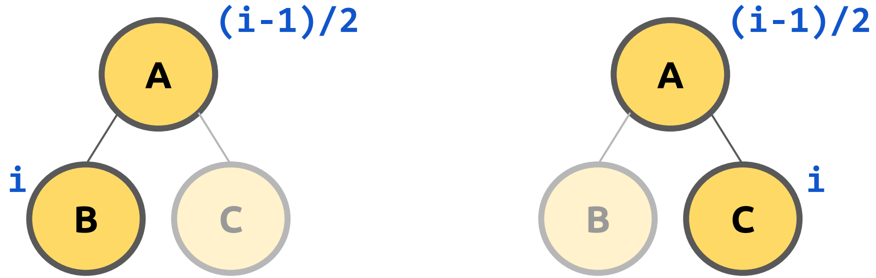


implementación en un array (basado en 0)

Los hijos del elemento que se encuentra en la posición i del arreglo, se encuentran en las posiciones **`2i+1 y 2i+2`**



El padre del elemento que se encuentra en la posición i, se encuentra en la posición **`(i-1)/2`**



implementación en un array

Como consecuencia de estar representado por un árbol completo, se llena de manera regular nivel por nivel

Se vacía en el orden inverso

Por eso en cada momento los i-1 primeros niveles están llenos

El nivel i se llena de izquierda a derecha

Esto facilita la implementación del montículo binario en un arreglo

¿en que posición comienzan las hojas?

Si consideramos que la primera posición del array es 1 como en el ejemplo, las hojas van desde **n/2 + 1** hasta **n**

Si consideramos que la posición inicial es 0, entonces las hojas ocuparán las posiciones **n/2** hasta **n-1**

Recordemos que la altura del árbol también está en función de n y se calcular como **h = ceil(log2(n+1)) - 1**

Ejercicio 3.01.

Implementá una estructura de montículo (min heap)

Debe tener las siguientes operaciones públicas

```
InsertarClave(int)
ObtenerMinimo(): int
ExtraerMinimo(): int
```

Montículo Binario

```
class MinHeap {
  private:
      int *h;
      int capacidad;
      int tamano;
      int padre(int i) { return (i-1)/2; }
      int hijoIzquierdo(int i){return (2*i + 1);}
      int hijoDerecho(int i) {return (2*i + 2);}
  public:
      MinHeap(int);
      void Heapify(int);
      int extraerMinimo();
      int obtenerMinimo() {return h[0];}
      void insertarClave(int k);
};
```

Ejercicio 3.02.

Dada una estructura de MinHeap, implementá la función

```
EliminarClave(int)
```

Ayuda: esta funcionalidad se puede implementar mediante una función que reduzca el valor de un nodo, asignando mediante esta el valor mínimo posible y luego extraer el mínimo

Ejercicio 3.03.

Escribí un programa que determine si un array de valores es un MaxHeap mediante una función recursiva

Ejercicio 3.04.

Crear un programa que calcule la mediana permanente de un stream de números.

Ejercicio 3.04.

Algoritmo:

1. Crear 2 montículos binarios: un MaxHeap y un MinHeap
2. El valor inicial de la mediana será 0
3. Para cada valor que se ingresa se procederá según las siguientes condiciones:

Ejercicio 3.04.

a. Si el tamaño del maxheap es mayor que el tamaño del minheap
   i. si el nuevo elemento es menor que la mediana anterior, entonces
      extraer del maxheap
      insertar este elemento en el minheap
      insertar el nuevo elemento en el maxheap
   ii. sino
      insertar el nuevo elemento en el minheap
   iii. calcular la mediana como la suma de los dos valores en la raiz de ambos heaps sobre 2
b. Si el tamaño del maxheap es menor que el tamaño del minheap
   i. si el nuevo elemento es mayor que la mediana anterior, entonces
      extraer del minheap
      insertar este elemento en el maxheap
      insertar el nuevo elemento en el minheap
   ii. sino
      insertar el nuevo elemento en el maxheap
   iii. calcular la mediana como la suma de los dos valores en la raiz de ambos

Ejercicio 3.04.

c. Si el tamaño del maxheap es igual al tamaño del minheap entonces:
   i. si el nuevo elemento es menor que la mediana actual
      insertar el nuevo elemento en el maxheap
      la mediana estará dada por la raíz del maxheap
   ii. sino
      insertar el nuevo elemento en el minheap
      la mediana estará dada por la raíz del minheap

heapsort

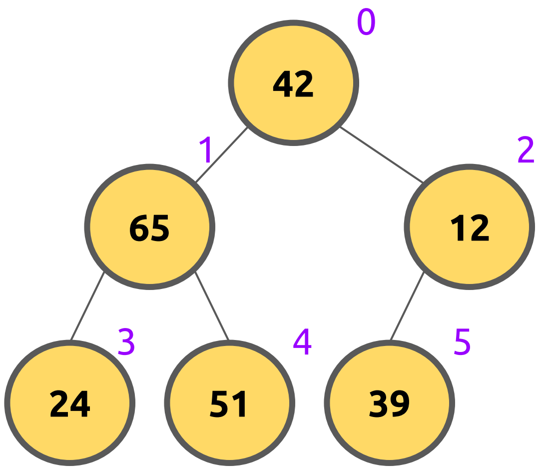
Lo primero que hacemos es hacer un heap con los elementos a ser ordenados:

- Un **max-heap** si queremos ordenarlos en orden **ascendente**
- Un **min-heap** si queremos ordenarlos de forma **descendente**

Una vez que hemos formado el heap, extraemos el mayor y lo colocamos en el último lugar del array y repetimos este procedimiento hasta que el array está ordenado

heapsort: ejemplo

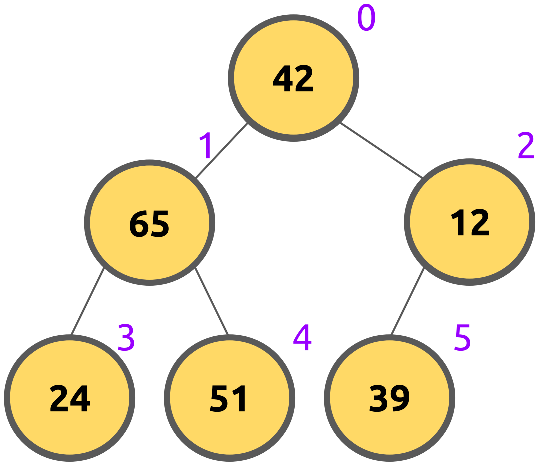
0	1	2	3	4	5
42	65	12	24	51	39



1. Crear el max heap
2. Extraer el mayor
3. Ubicarlo en la partición ordenada

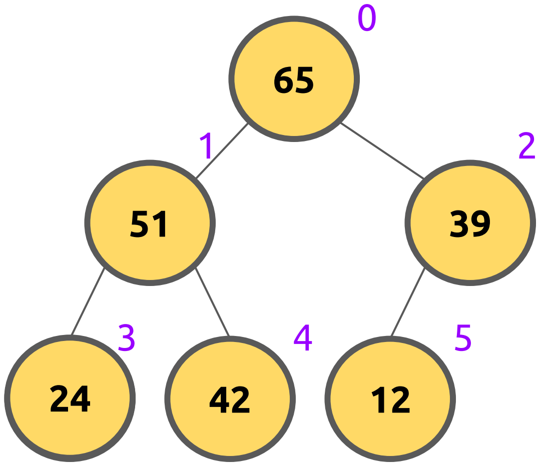
heapsort: ejemplo

0	1	2	3	4	5
42	65	12	24	51	39

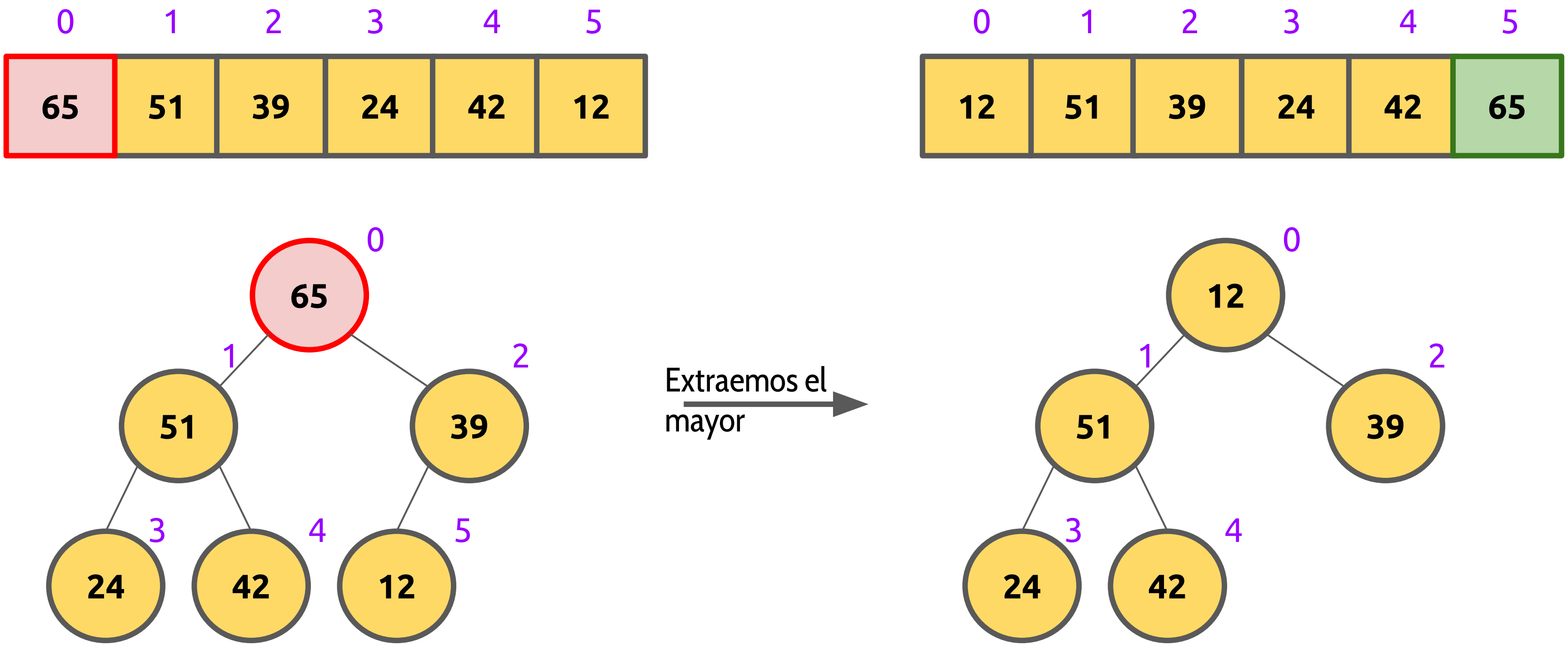


Construimos
el max heap

0	1	2	3	4	5
65	51	39	24	42	12



heapsort: ejemplo



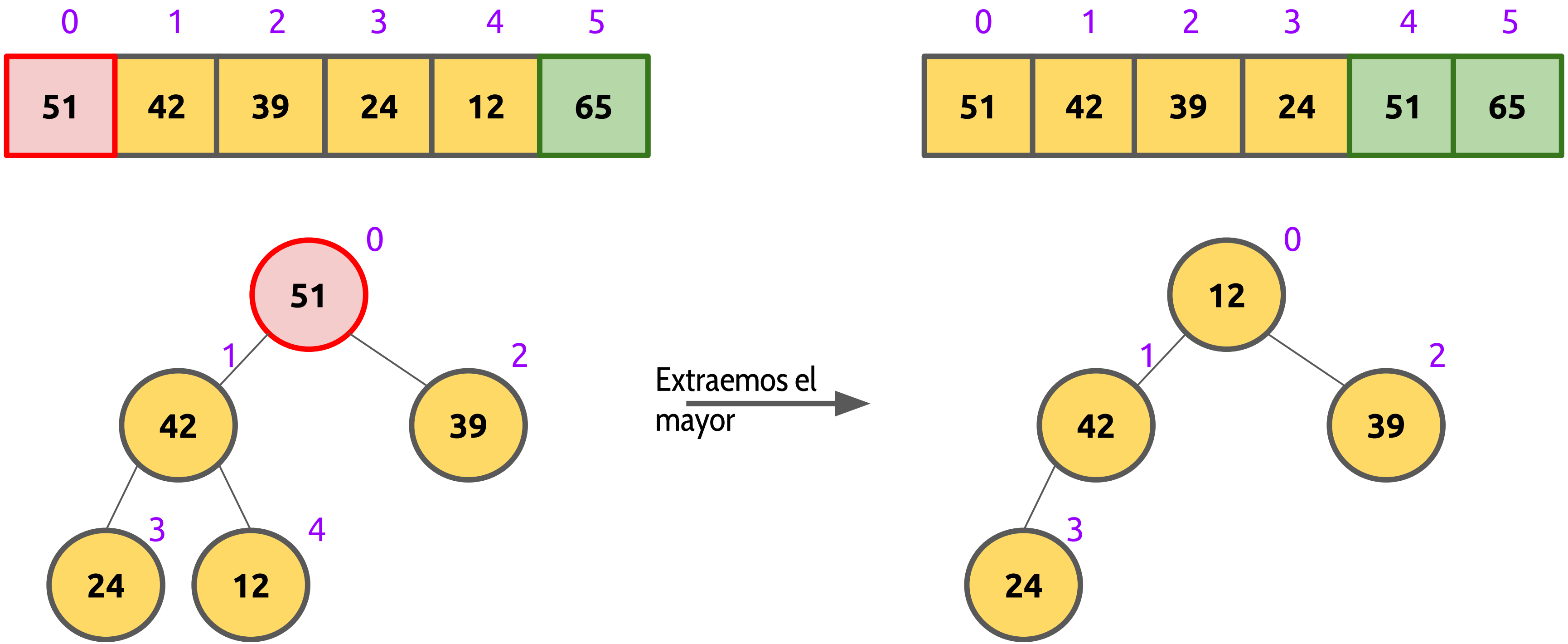
heapsort: ejemplo

0	1	2	3	4	5
12	51	39	24	42	65

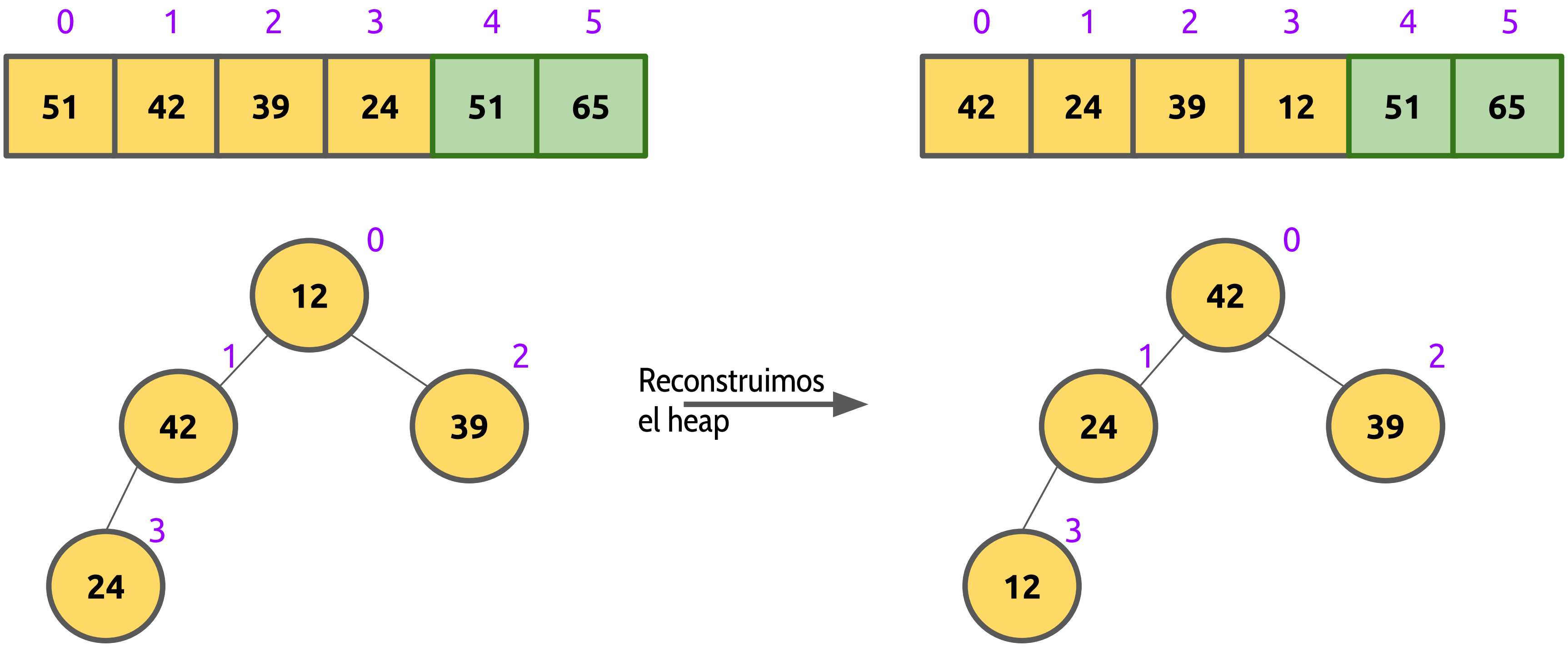
0	1	2	3	4	5
51	42	39	24	12	65

UBP UNIVERSIDAD BLAS PASCAL

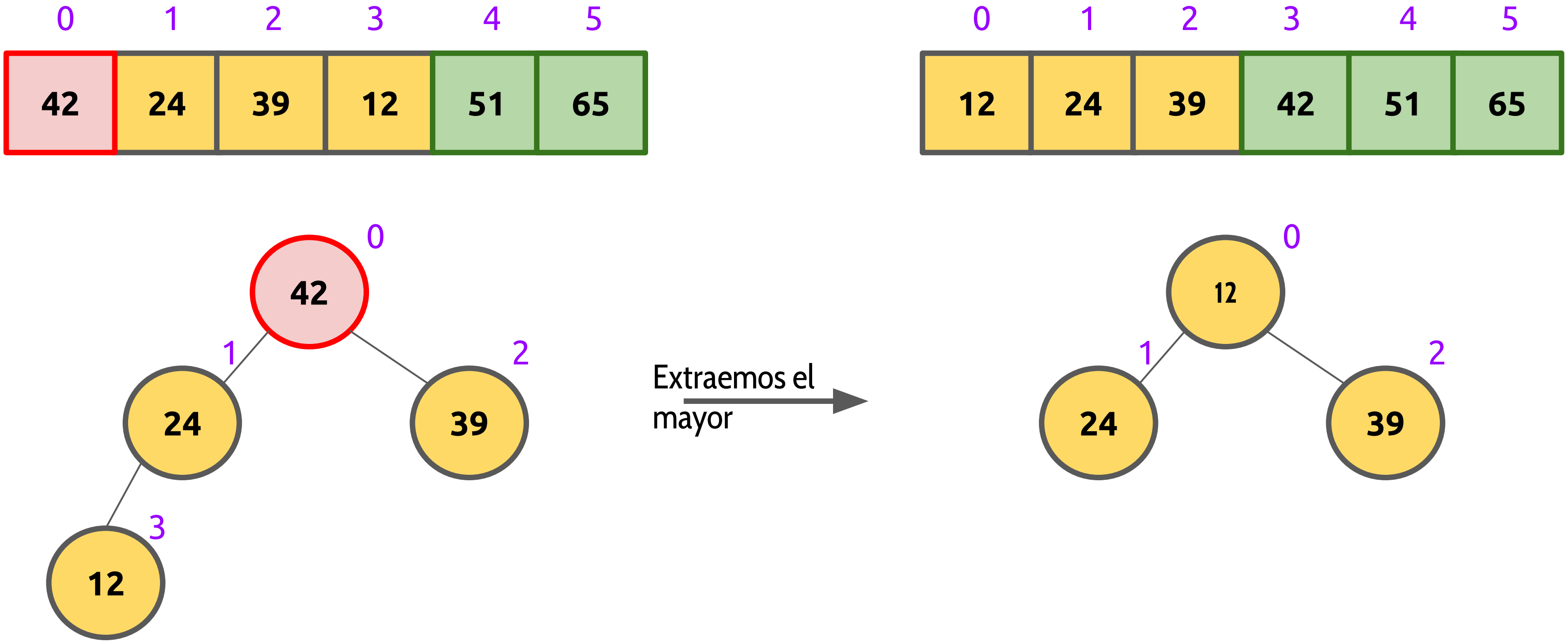
heapsort: ejemplo



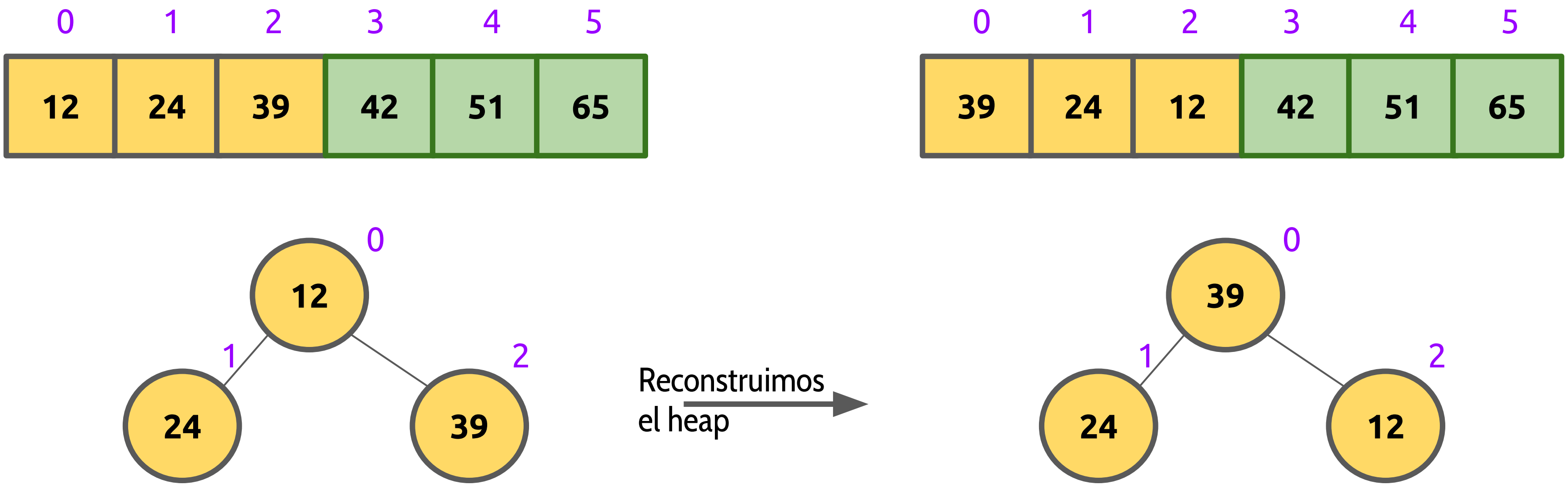
heapsort: ejemplo



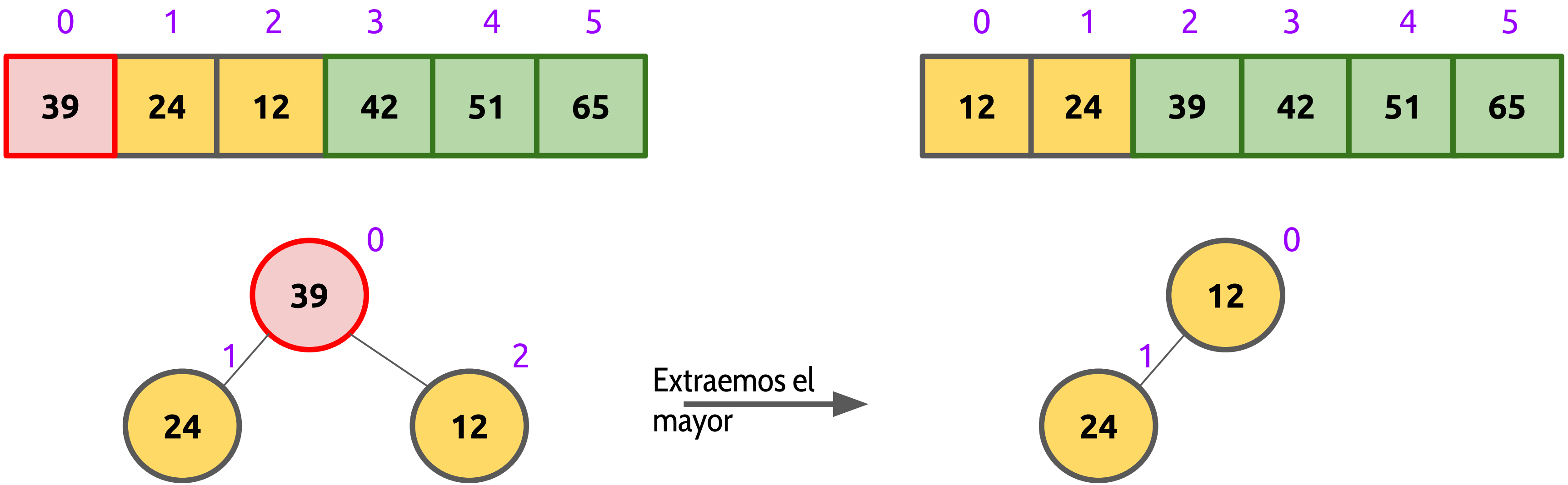
heapsort: ejemplo



heapsort: ejemplo



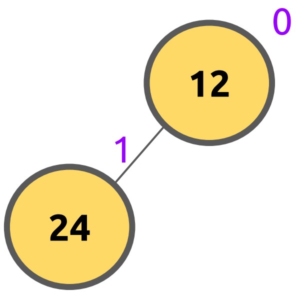
heapsort: ejemplo



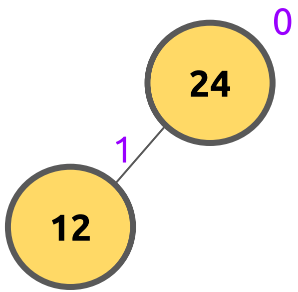
heapsort: ejemplo

0	1	2	3	4	5
12	24	39	42	51	65

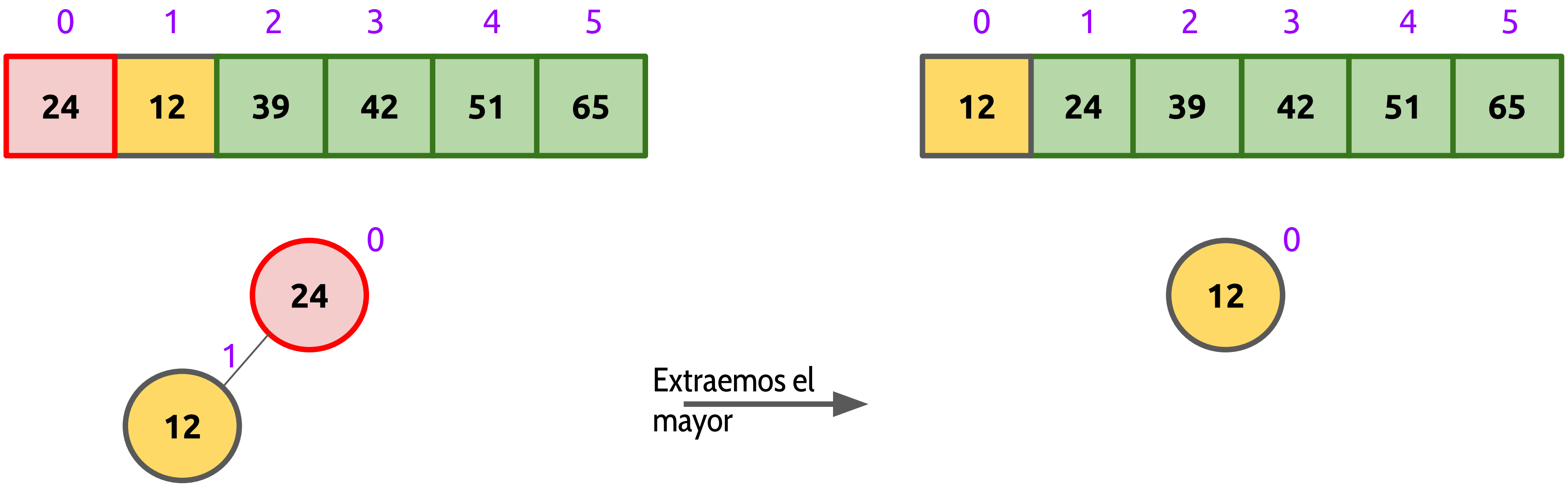
0	1	2	3	4	5
24	12	39	42	51	65



Reconstruimos el heap



heapsort: ejemplo



heapsort: ejemplo

0	1	2	3	4	5
12	**24**	**39**	**42**	**51**	**65**

heapsort: algoritmo

1. Construir un heap con los elementos a ordenar.

2. El menor elemento estará almacenado en la raíz del heap. Intercambiamos este elemento por el último elemento del heap y reducimos el tamaño del heap en 1. Después aplicamos heapify al elemento raíz del heap.

3. Repetimos el proceso anterior mientras el tamaño del heap es mayor que 1.

heapsort: complejidad

O(n log n)

Construir el max heap: O(n)

Reconstruir el heap: O(log n) la llamamos (n-1) veces

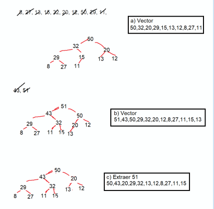
Ejercicio 3.05.

Implementá el método de ordenación Heap Sort para un array de valores enteros

Ejercicio 3.06. (en papel)

Suponé un max-heap que contiene las claves 8, 27, 13, 15, 32, 20, 12, 50, 29, 11, insertadas en este orden partiendo de un heap vacío

a) Escribí el contenido del array después de realizar las inserciones
b) Insertá las claves 43 y 51 y volvé a escribir el contenido del array
c) Ejecutá tres veces `extraerMaximo()` y escribí el estado del array en cada caso



Ejercicio 3.07.

Dados n tramos de soga de tamaño t. Escribí un programa que permita calcular el **costo** de unir tramos de soga con el menor costo. Teniendo en cuenta que el costo de unir dos tramos de soga es igual a la suma de la longitud de ambos tramos.

Por ejemplo dadas sogas de longitudes 2, 4, 3 y 6
1. Unimos la de 2 y 3 quedando una de costo 5
2. Unimos la de 4 y la de 5 resultando una de 9 el costo acumulado ahora es 9+5 = 14
3. Unimos la de la de 9 y la de 6 el costo de la union es 15 y el costo acumulado es 14+15 = 29.
Este es el costo total

Ejercicio 3.08.

Implementá la clase MinHeap del ejercicio 3.01 usando el contenedor **`vector`** en lugar de un arreglo

Ejercicio 3.09.

Creá un programa que administre la cola de atención de un hospital de urgencias. Cuando los pacientes llegan, se registran los siguientes datos: Nombre, Apellido, DNI y gravedad del caso. La gravedad es un valor entre 1 (más leve) y 100 (más grave) que se determina durante la pre admisión.

Los pacientes son atendidos por orden de gravedad, sin importar el orden de llegada del paciente.