

# MONTÍCULOS BINARIOS



U N I V E R S I D A D  
**COMPLUTENSE**  
M A D R I D

**ALBERTO VERDEJO**

# Implementaciones de colas de prioridad

implementación	push	top	pop
vector desordenado	1	$N$	$N$
vector ordenado	$N$	1	1
montículo binario	$\log N$	1	$\log N$
montículo k-ario	$\log_k N$	1	$k \log_k N$
Fibonacci	1	1	$\log N$
imposible	1	1	1

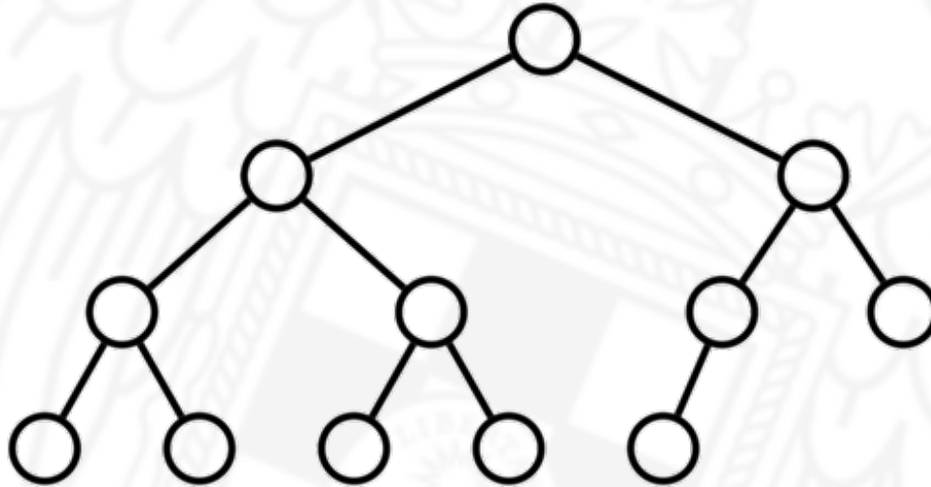
# Árboles binarios completos

- Un árbol binario de altura  $h$  es **completo** cuando todos sus nodos internos tienen dos hijos no vacíos, y todas sus hojas están en el nivel  $h$ .



# Árboles binarios semicompletos

- Un árbol binario de altura  $h$  es **semicompleto** si o bien es completo o bien tiene vacantes una serie de posiciones consecutivas del nivel  $h$  empezando por la derecha.

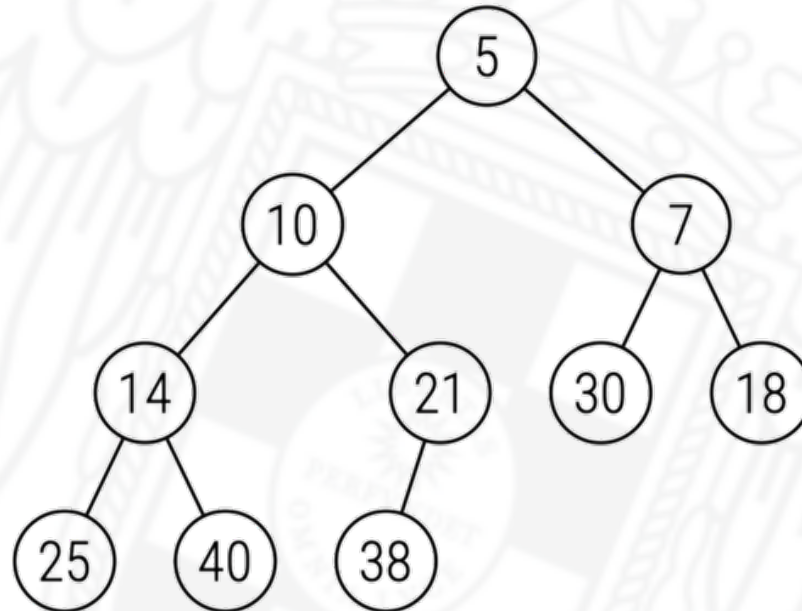


# Propiedades

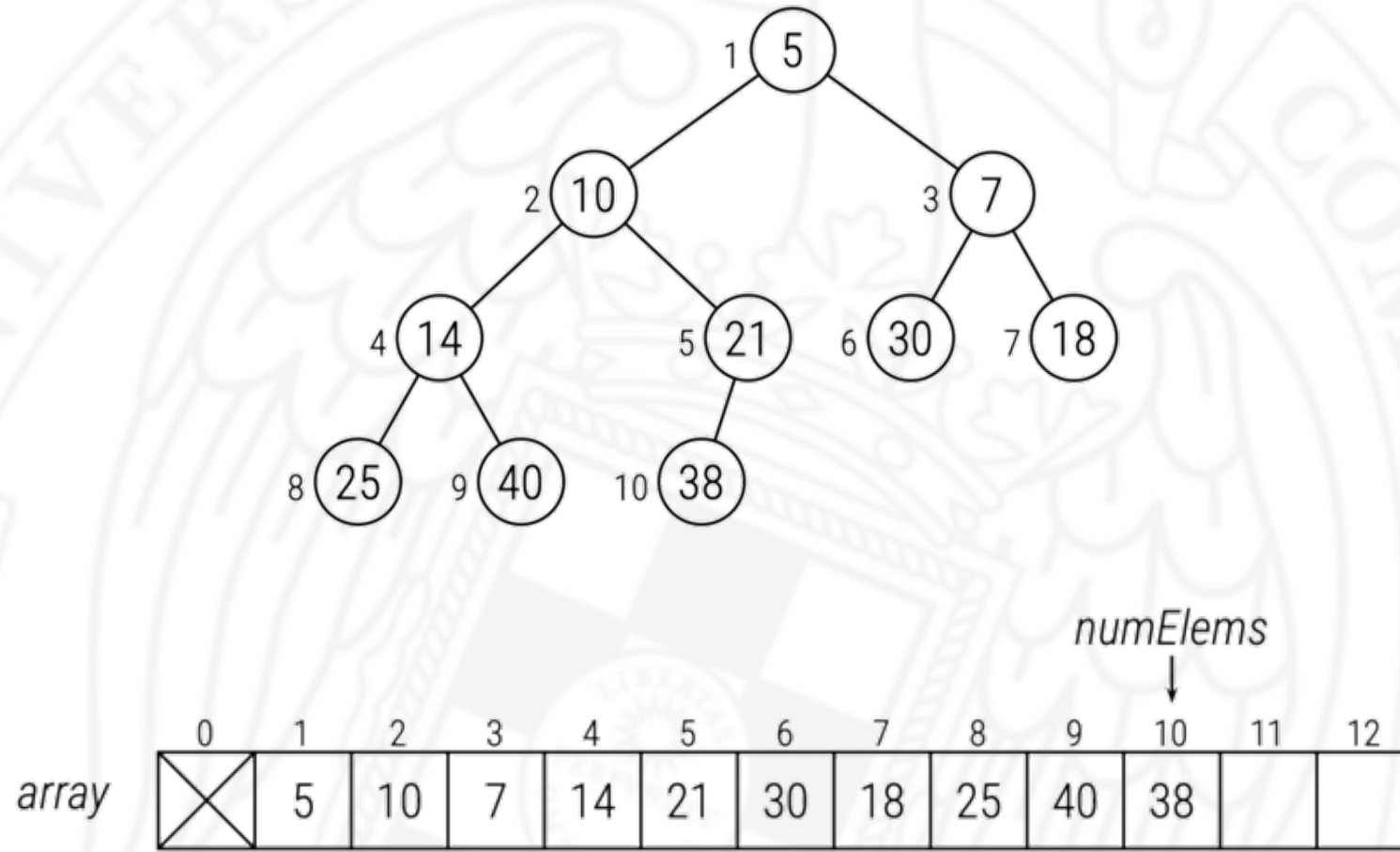
- ▶ Un árbol binario completo de altura  $h \geq 1$  tiene  $2^i - 1$  nodos en el nivel  $i$ , para todo  $i$  entre 1 y  $h$ .
- ▶ Un árbol binario completo de altura  $h \geq 1$  tiene  $2^h - 1$  hojas.
- ▶ Un árbol binario completo de altura  $h \geq 0$  tiene  $2^h - 1$  nodos.
- ▶ La altura de un árbol binario semicompleto formado por  $n$  nodos es  $\lfloor \log n \rfloor + 1$ .

# Montículos binarios

- Un **montículo (binario) de mínimos** es un árbol binario semicompleto donde el elemento en la raíz es menor (o igual) que todos los elementos en el hijo izquierdo y en el derecho, y ambos hijos son a su vez montículos de mínimos.



# Representación de un montículo





# Implementación de las colas de prioridad



PriorityQueue.h

```
// Comparator dice cuándo un valor de tipo T es más prioritario que otro
template <typename T, typename Comparator = std::less<T>>
class PriorityQueue {

    // vector que contiene los datos
    std::vector<T> array; // primer elemento en la posición 1

    // Objeto función que sabe comparar elementos.
    // antes(a,b) es cierto si a es más prioritario que b
    // (a debe salir antes que b)
    Comparator antes;

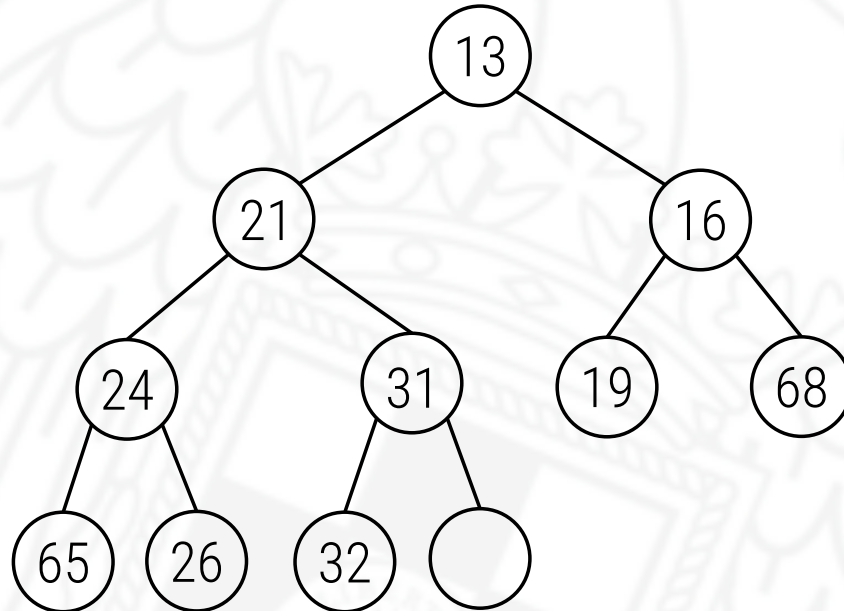
public:

    PriorityQueue(Comparator c = Comparator()) : array(1), antes(c) {}
```



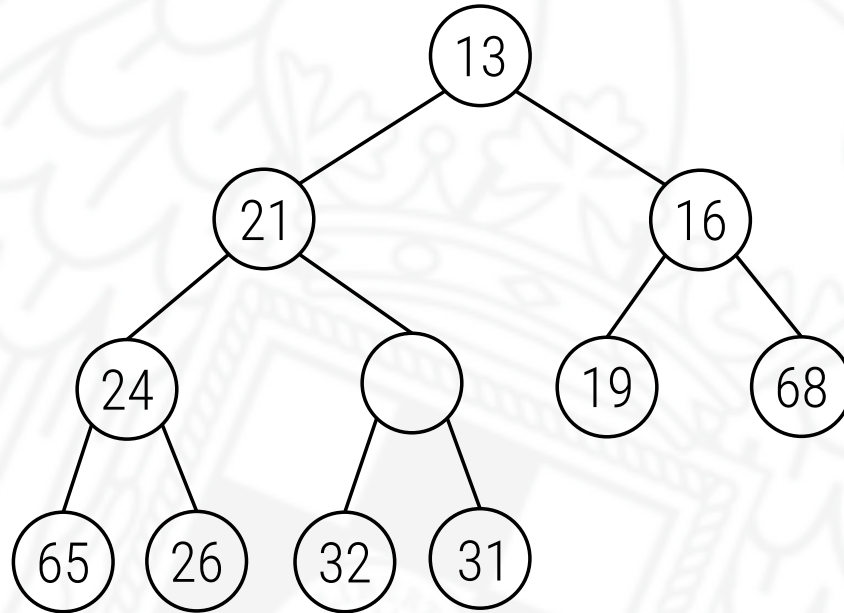
# Inserción

- Insertar el 14



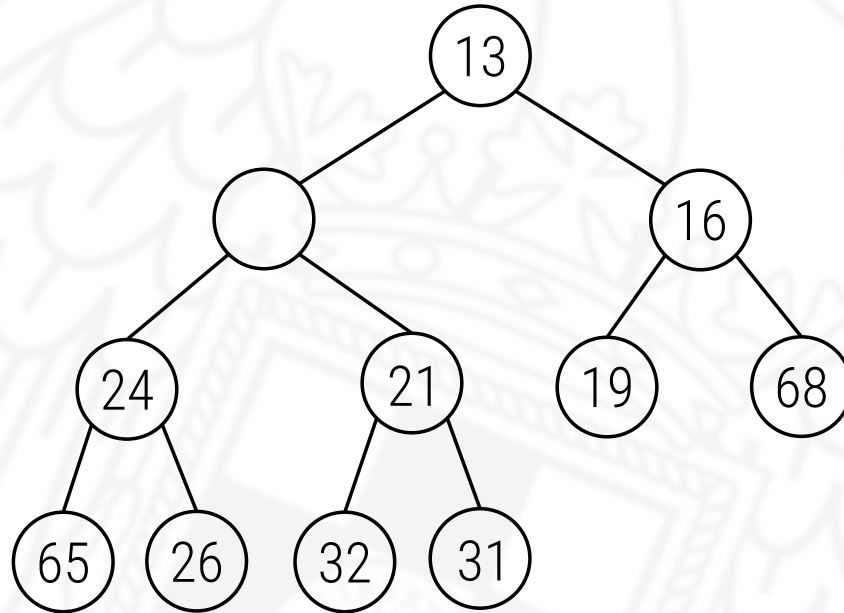
# Inserción

- Insertar el 14

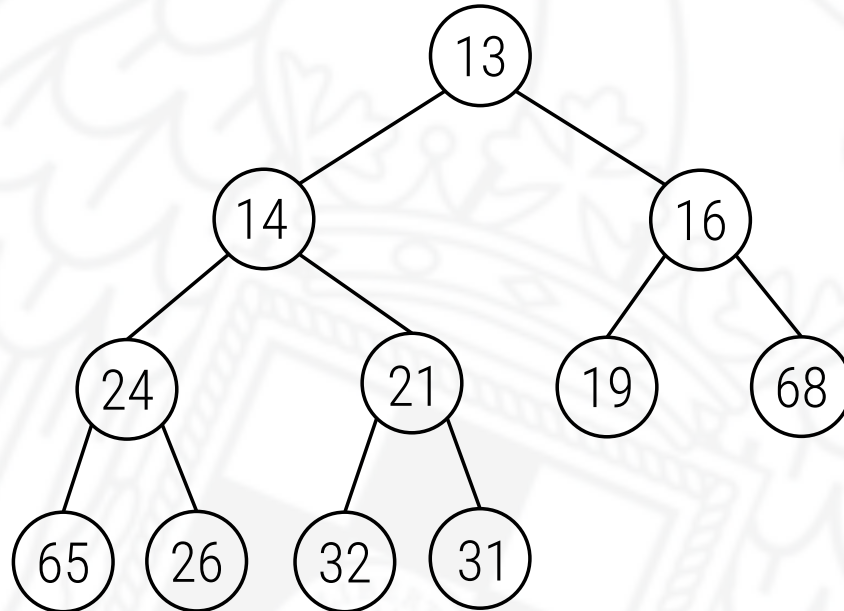


# Inserción

- Insertar el 14



# Inserción



# Implementación de las colas de prioridad



PriorityQueue.h

```
public:
    void push(T const& x) {
        array.push_back(x);
        flotar(array.size() - 1);
    }

private:
    void flotar(int i) {
        T elem = array[i];
        int hueco = i;
        while (hueco != 1 && antes(elem, array[hueco / 2])) {
            array[hueco] = array[hueco / 2];
            hueco /= 2;
        }
        array[hueco] = elem;
    }
}
```

# Implementación de las colas de prioridad



PriorityQueue.h

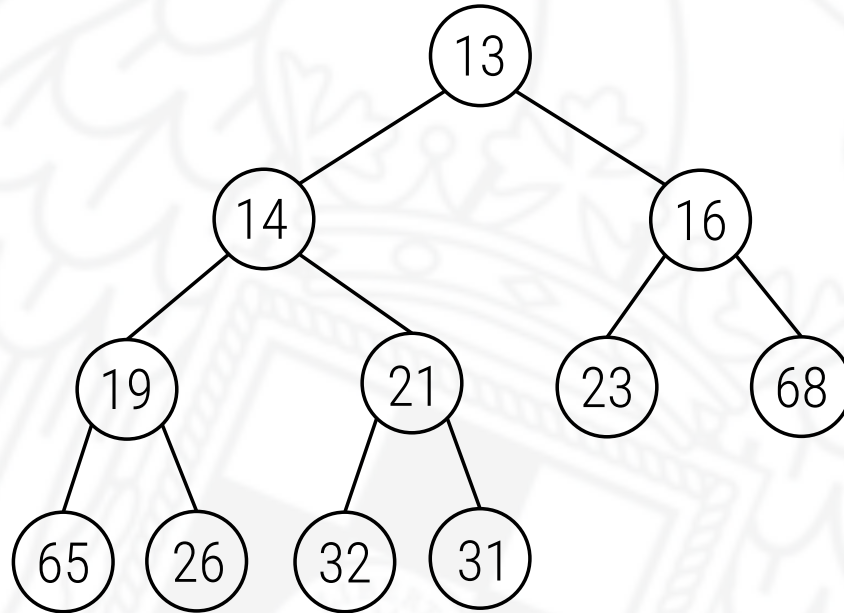
```
public:
    int size() const { return array.size() - 1; }

    bool empty() const { return size() == 0; }

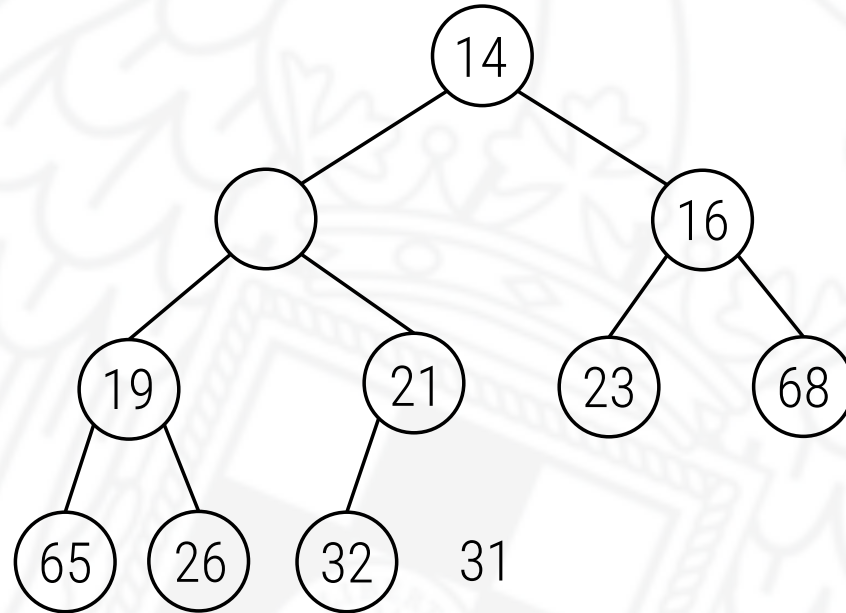
    T const& top() const {
        if (empty())
            throw std::domain_error("La cola vacia no tiene top");
        else
            return array[1];
    }
```



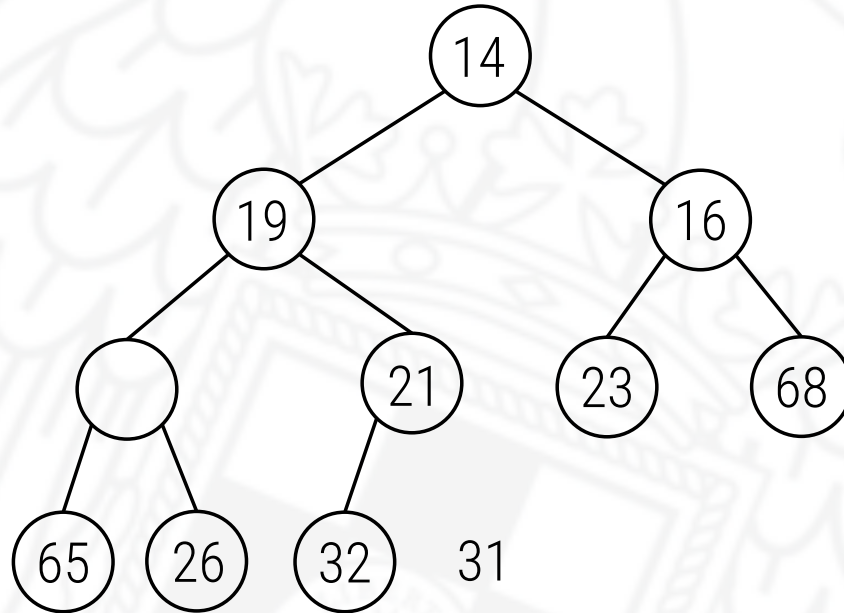
# Eliminación del más prioritario



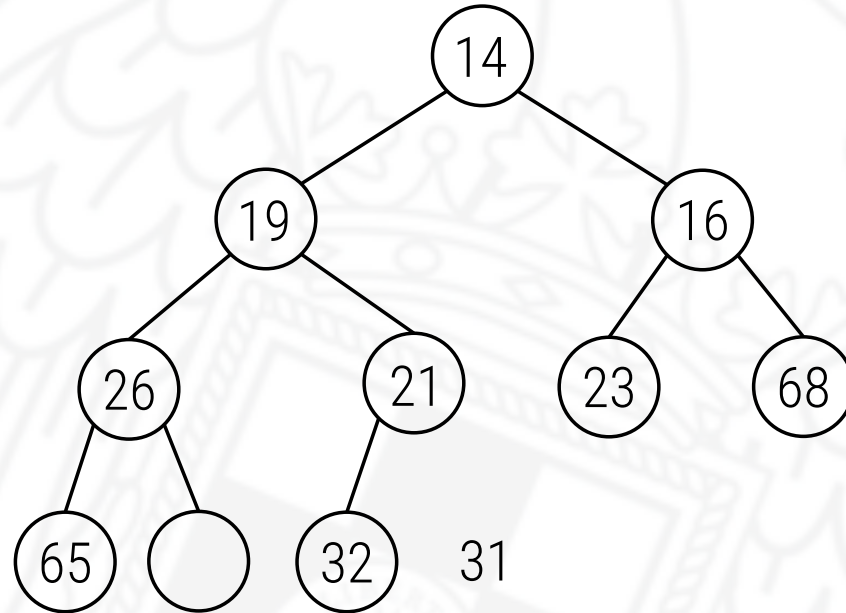
# Eliminación del más prioritario



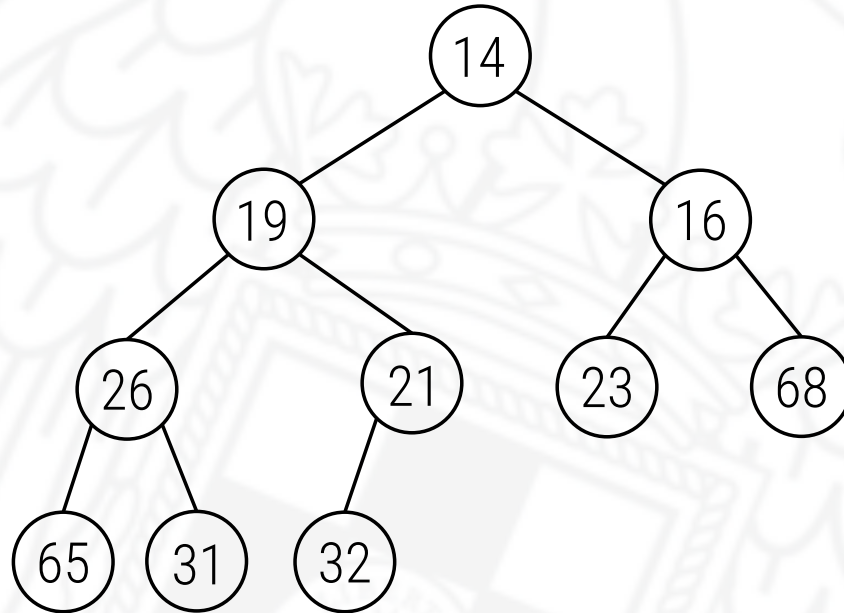
# Eliminación del más prioritario



# Eliminación del más prioritario



# Eliminación del más prioritario



# Implementación de las colas de prioridad

```
public:
    void pop() {
        if (empty())
            throw std::domain_error(
                "Imposible eliminar el primero de una cola vacia");
        else {
            array[1] = array.back();
            array.pop_back();
            if (!empty()) hundir(1);
        }
    }
}
```



# Implementación de las colas de prioridad



PriorityQueue.h

```
private:
```

```
void hundir(int i) {
    T elem = array[i];
    int hueco = i;
    int hijo = 2 * hueco; // hijo izquierdo, si existe
    while (hijo <= size()) {
        // cambiar al hijo derecho si existe y va antes que el izquierdo
        if (hijo < size() && antes(array[hijo + 1], array[hijo]))
            ++hijo;
        // flotar el hijo si va antes que el elemento hundiéndose
        if (antes(array[hijo], elem)) {
            array[hueco] = array[hijo];
            hueco = hijo; hijo = 2 * hueco;
        } else break;
    }
    array[hueco] = elem;
}
```