HEAPSORT

UNIVERSIDAD
COMPLUTENSE
MADRID

ALBERTO VERDEJO

Heapsort abstracto

- Utiliza una cola de prioridad en vez de un montículo directamente.

```
template <typename T>
void heapsort_abstracto(std::vector<T> & v) {
    PriorityQueue<T> colap;
    for (auto const& e : v)
        colap.push(e);
    for (int i = 0; i < v.size(); ++i) {
        v[i] = colap.top();
        colap.pop();
    }
}
```

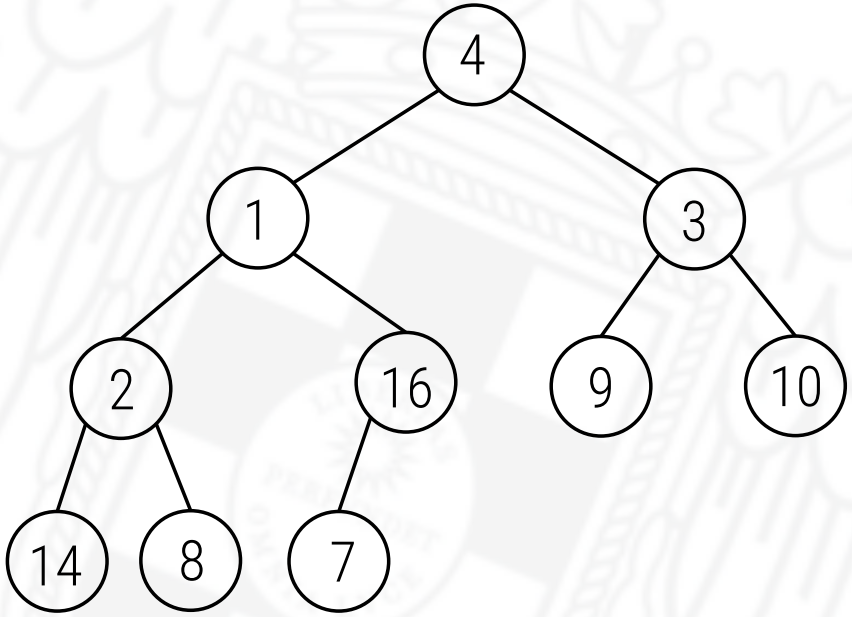
El coste en tiempo está en $\Theta(N \log N)$, y en espacio adicional en $\Theta(N)$.

Heapsort

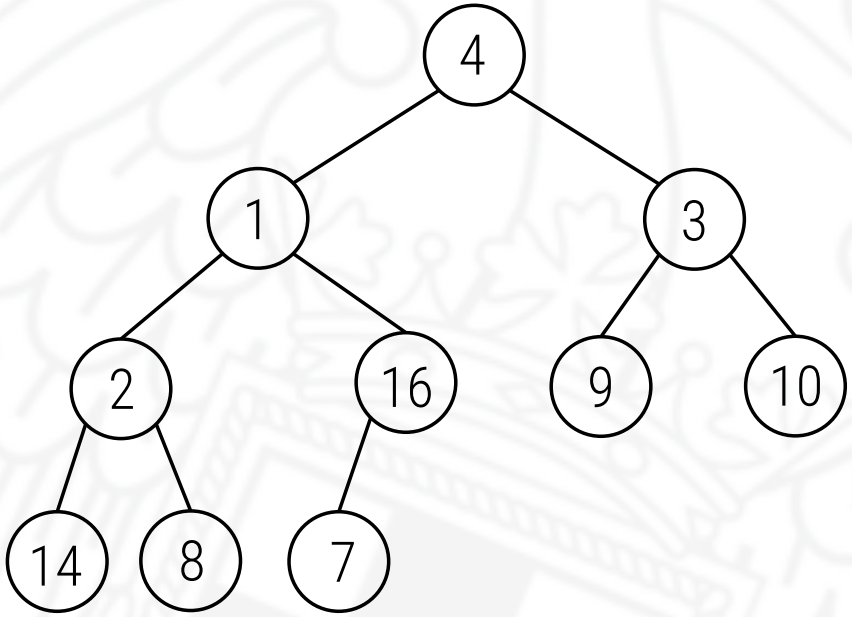
- Podemos ahorrarnos ese espacio adicional si utilizamos el mismo vector para representar el montículo auxiliar.
- Primero el vector se convierte en un montículo.
- Después se va extrayendo sucesivamente el elemento más prioritario para colocarlo al final del vector, en la parte ya no necesaria para almacenar el montículo, cada vez más pequeño.

Convertir el vector en un montículo

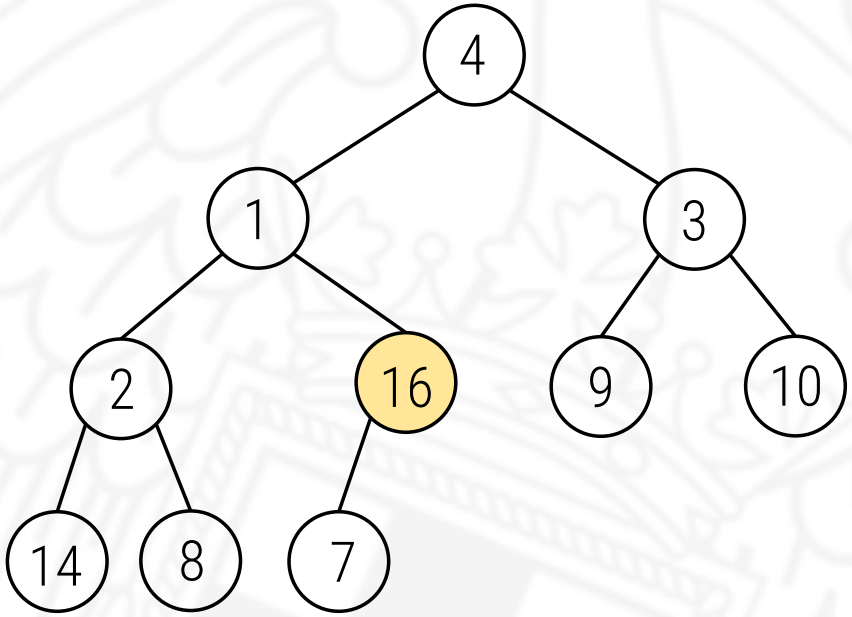
4	1	3	2	16	9	10	14	8	7



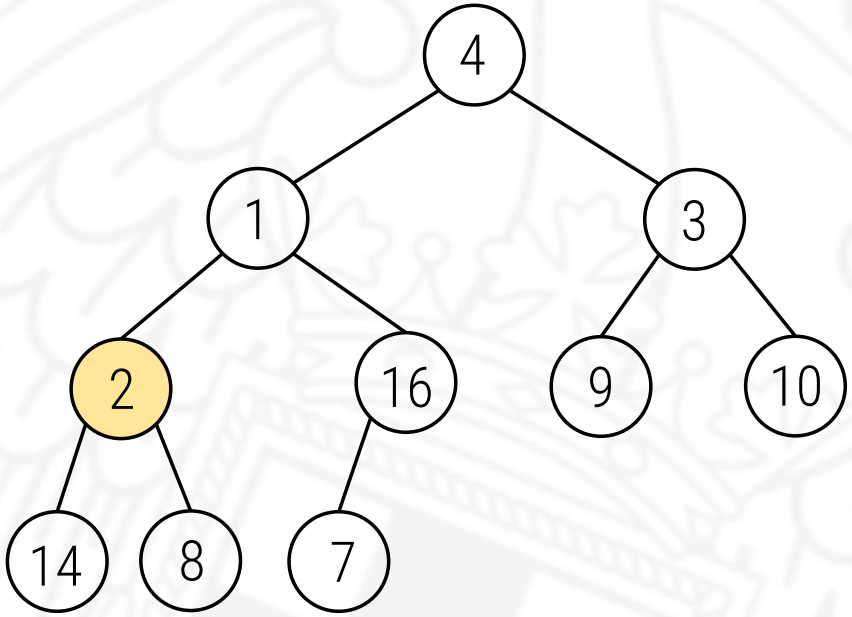
Convertir el vector en un montículo



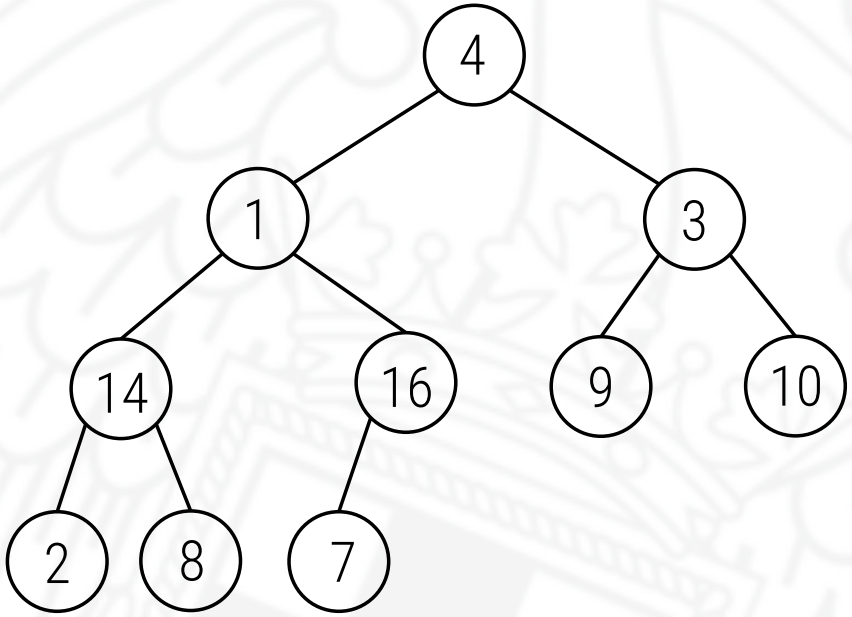
Convertir el vector en un montículo



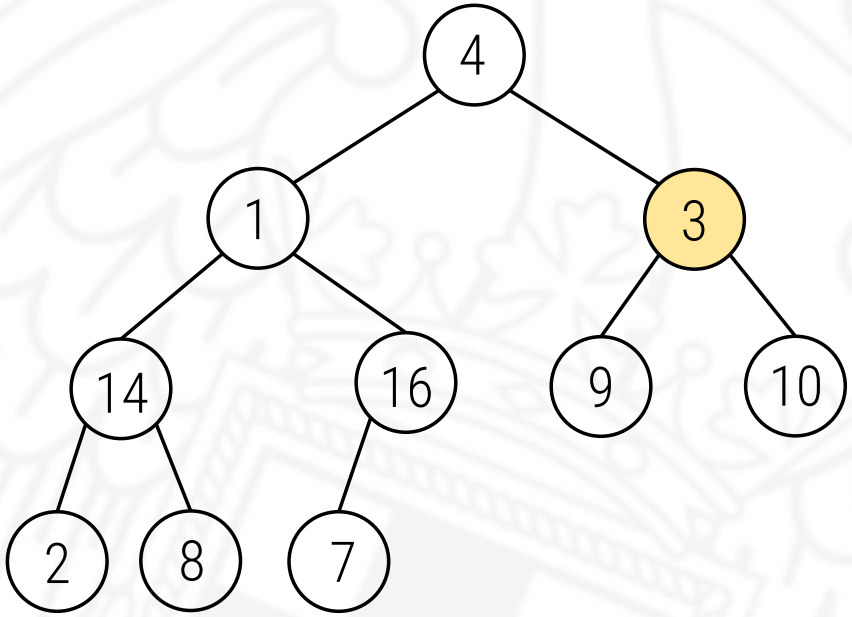
Convertir el vector en un montículo



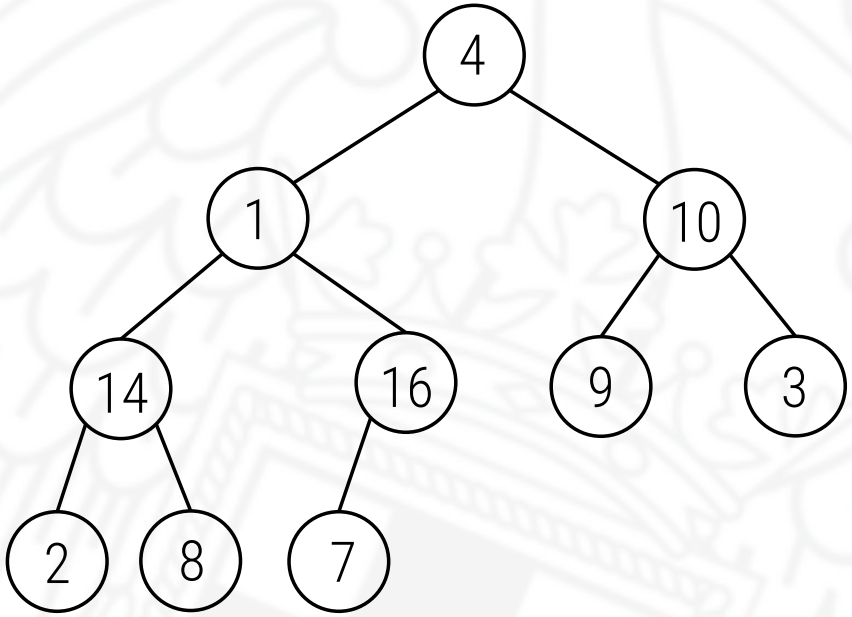
Convertir el vector en un montículo



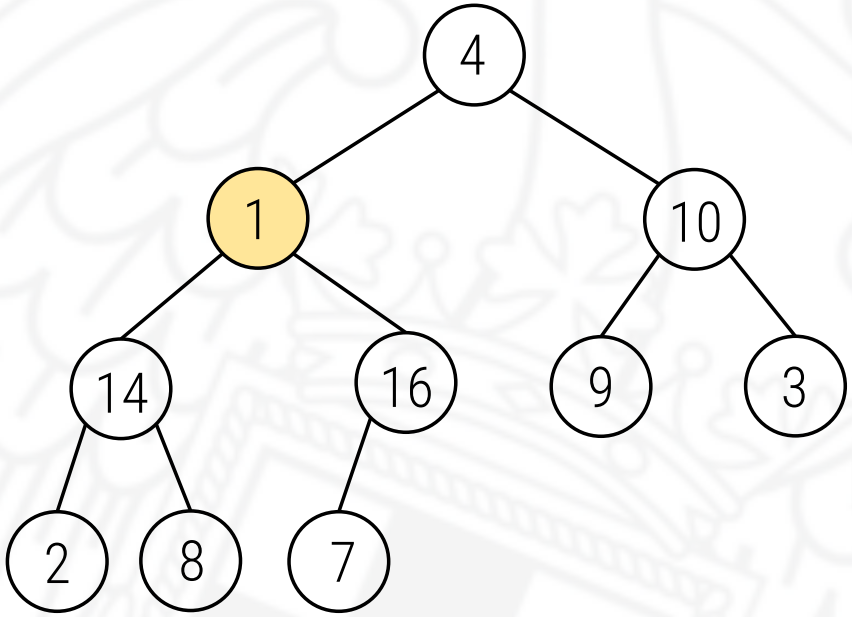
Convertir el vector en un montículo



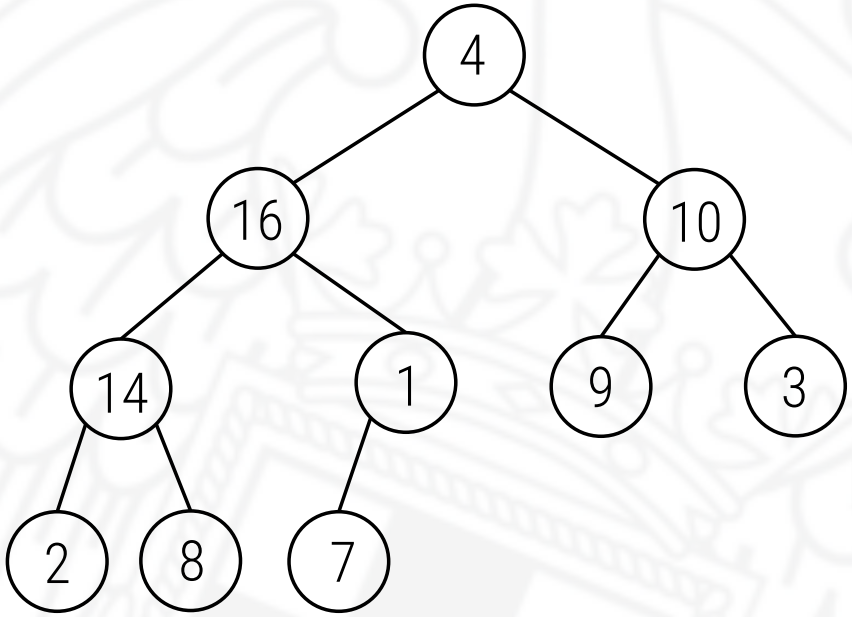
Convertir el vector en un montículo



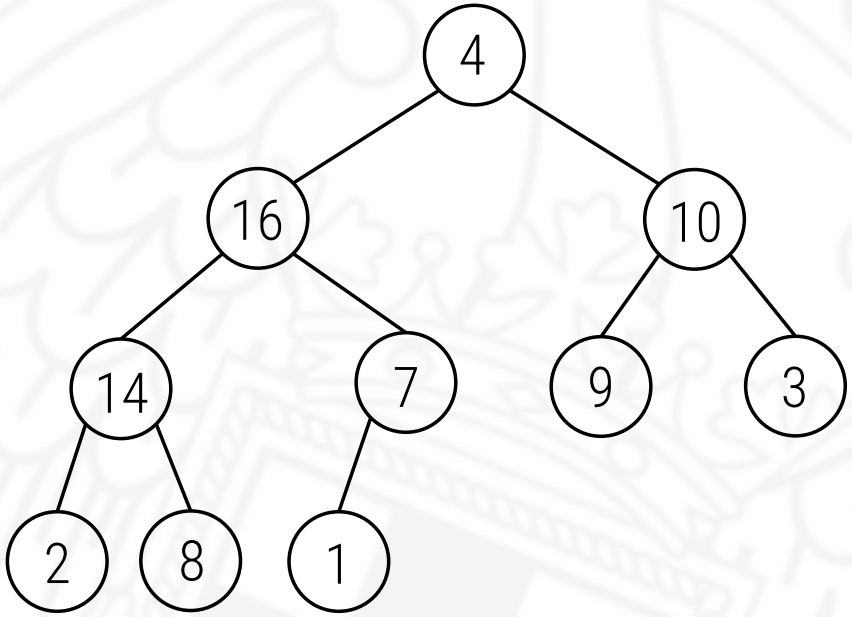
Convertir el vector en un montículo



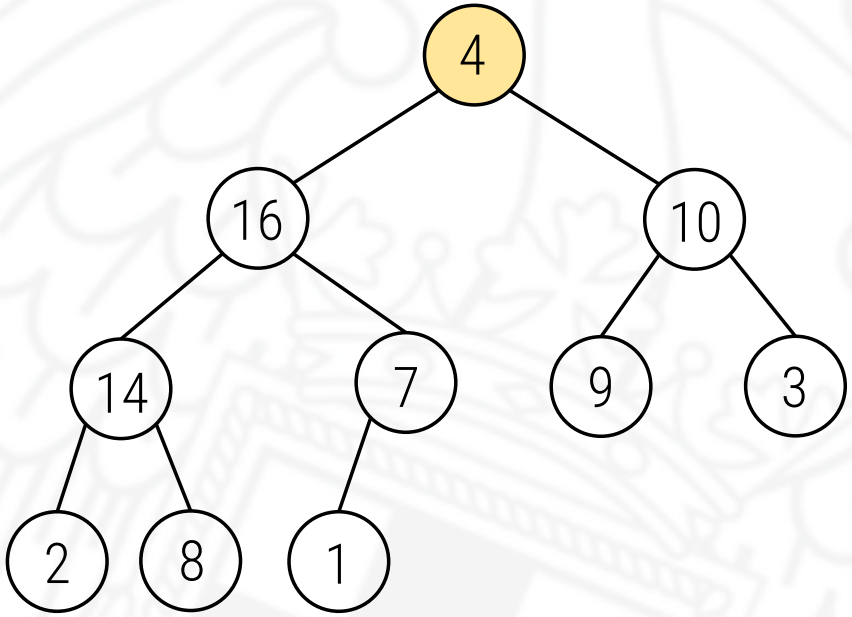
Convertir el vector en un montículo



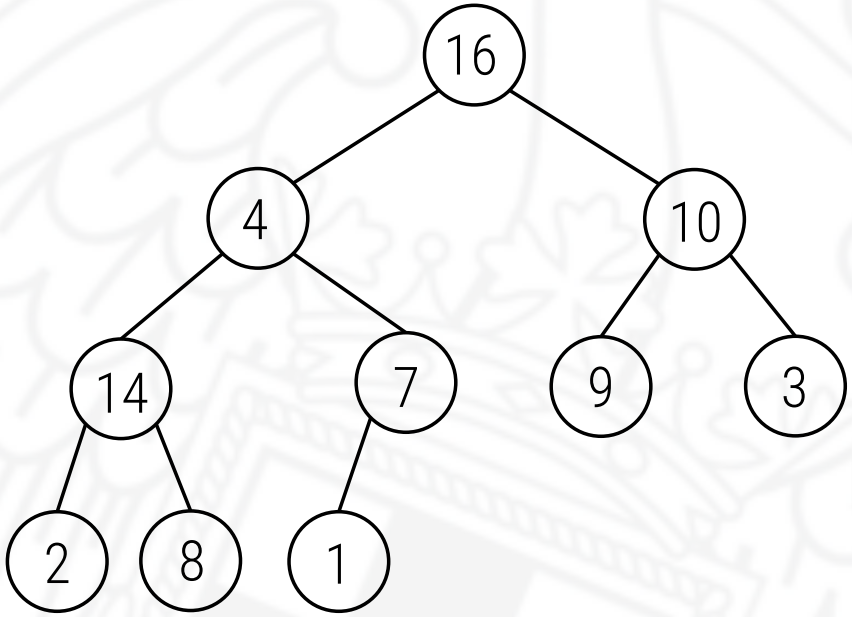
Convertir el vector en un montículo



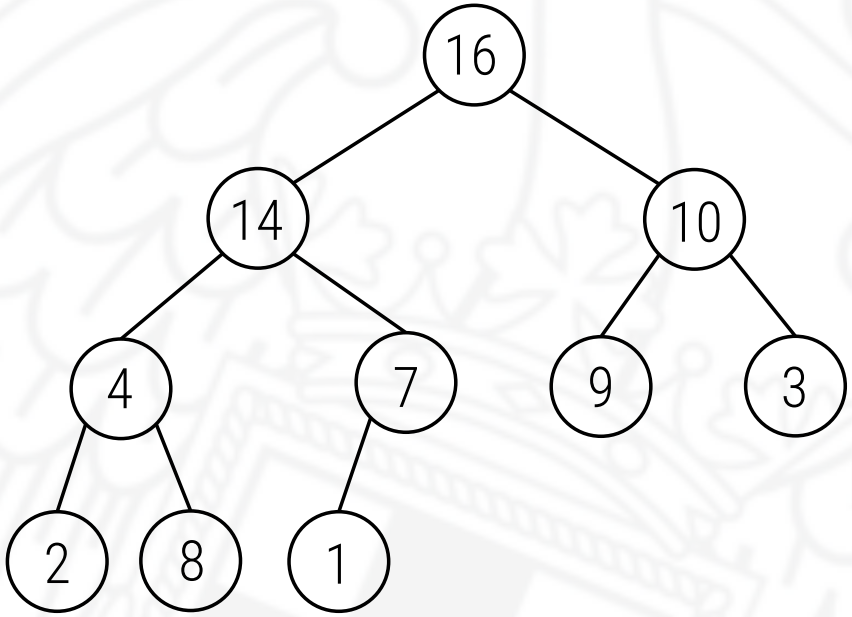
Convertir el vector en un montículo



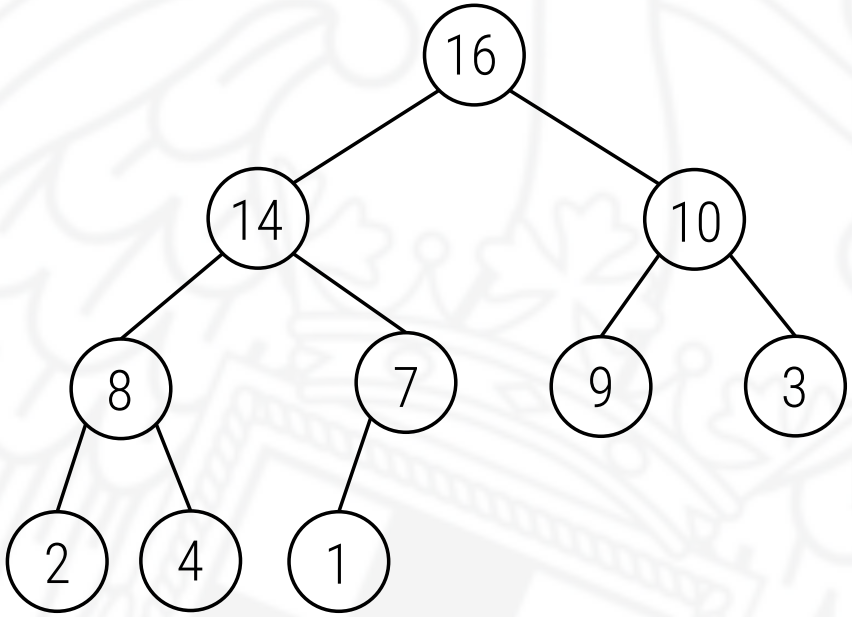
Convertir el vector en un montículo



Convertir el vector en un montículo



Convertir el vector en un montículo



16	14	10	8	7	9	3	2	4	1

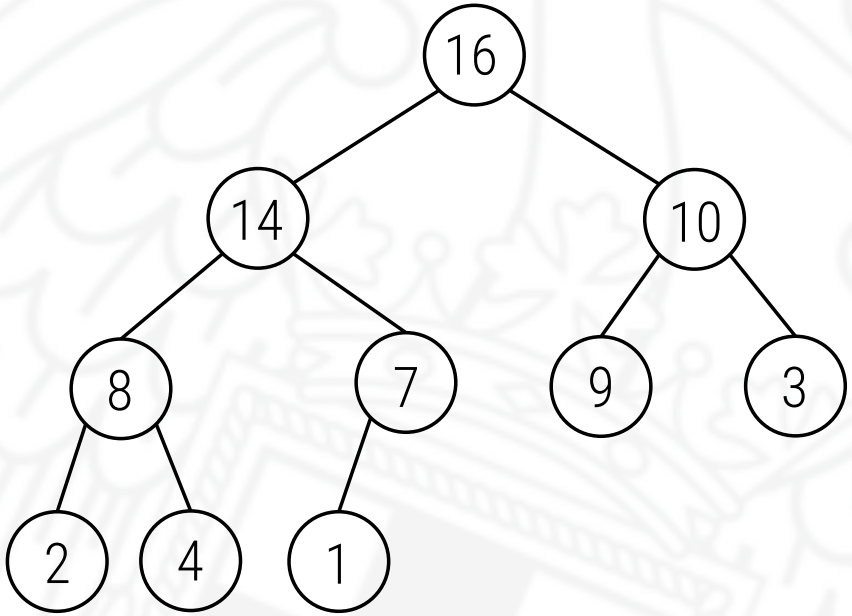
Coste (amortizado)

nivel	nodos	hunden
h	2^{h-1}	nada
$h-1$	2^{h-2}	cada uno 1
$h-2$	2^{h-3}	cada uno 2
	$\vdots$	
i	2^{i-1}	cada uno $h-i$
	$\vdots$	
1	1	$h-1$

$$\sum_{i=1}^{h-1}(h-i)2^{i-1} = \sum_{j=2}^{h}(j-1)2^{h-j} < \sum_{j=1}^{h} j2^{h-j} = 2^h \sum_{j=1}^{h}\frac{j}{2^j}$$

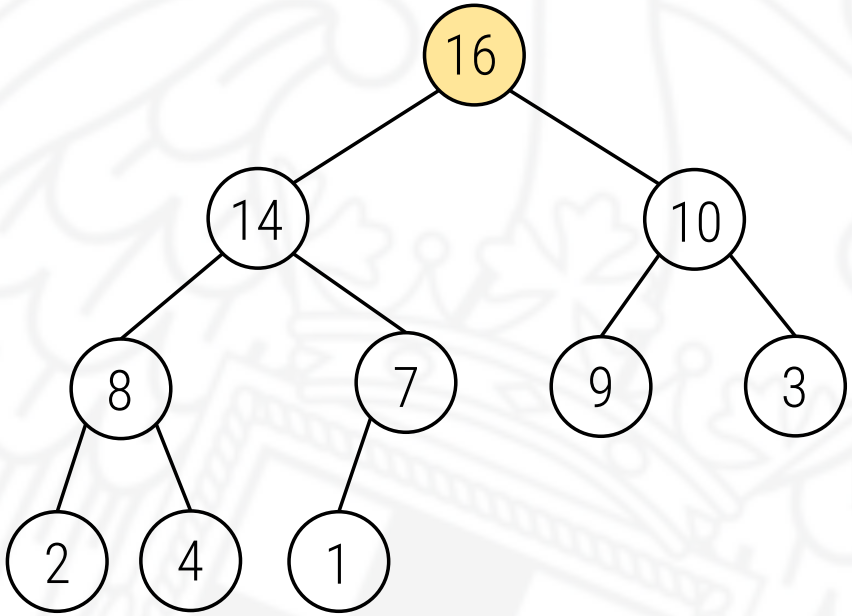
$$= 2^h\left(2-\frac{h+2}{2^h}\right) \leq 2^{h+1} = 2^{\lfloor \log N \rfloor + 2} \in O(N)$$

Ordenar

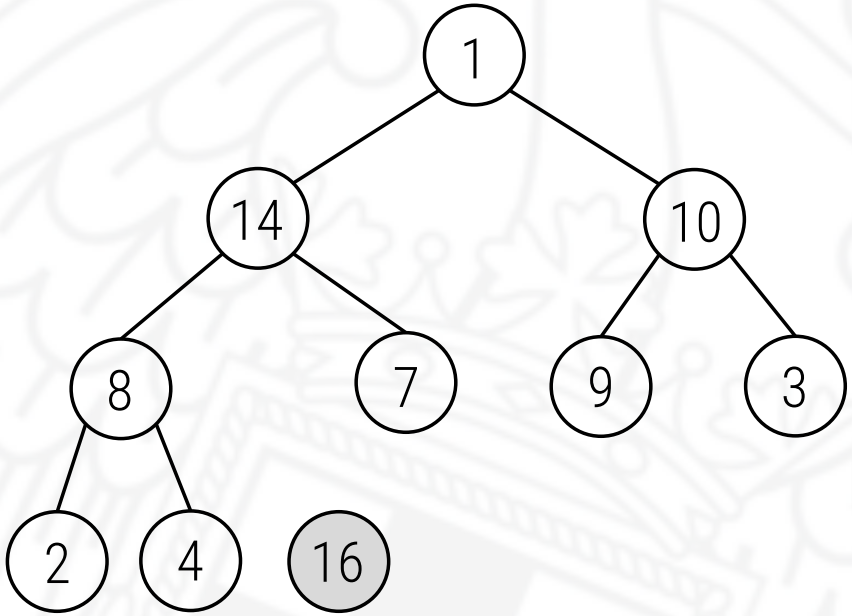


16	14	10	8	7	9	3	2	4	1

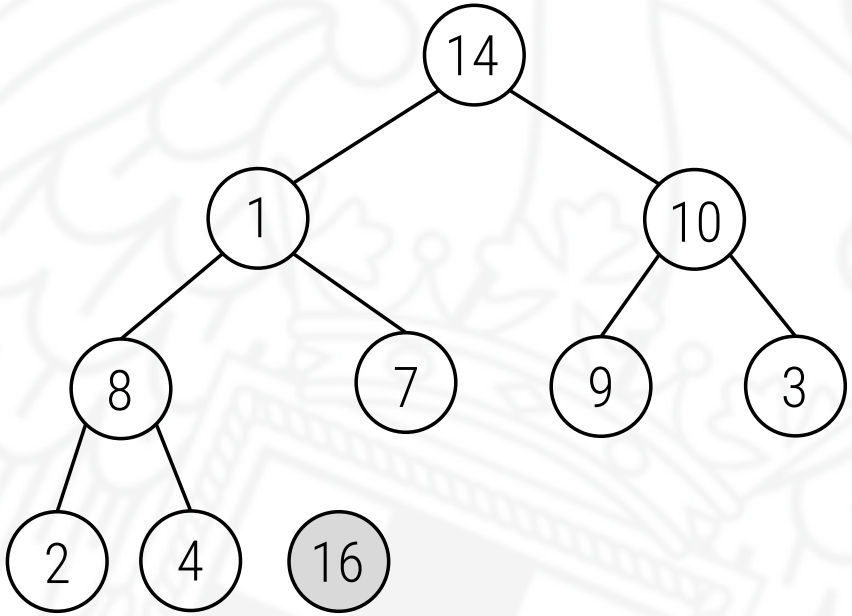
Ordenar



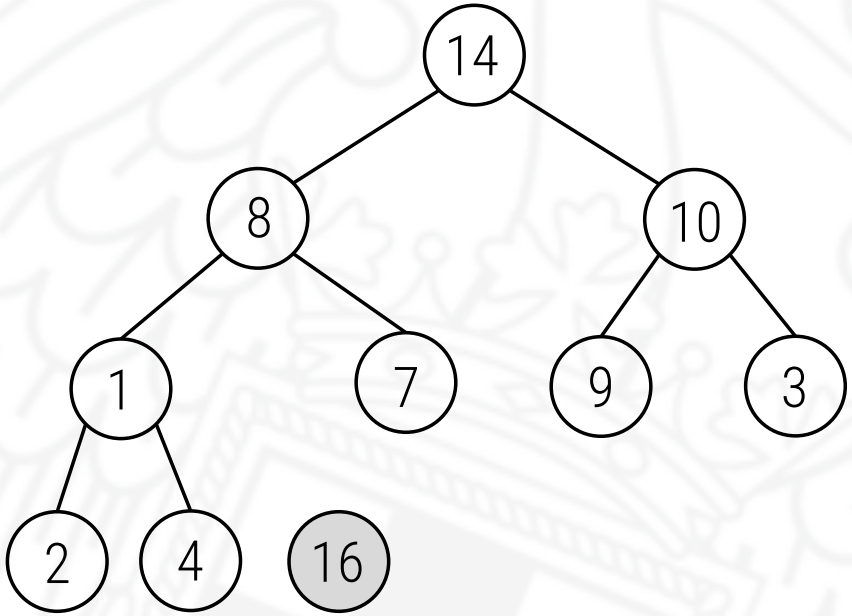
Ordenar



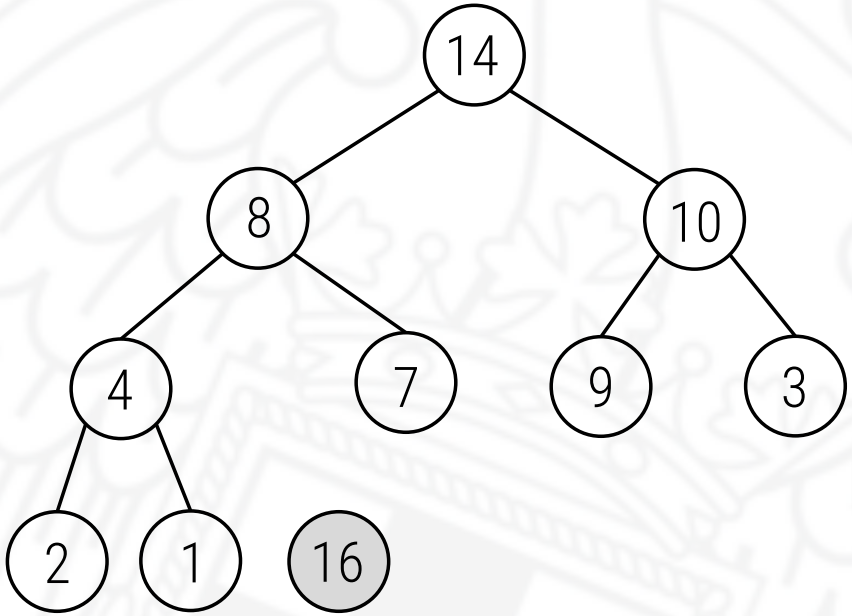
Ordenar



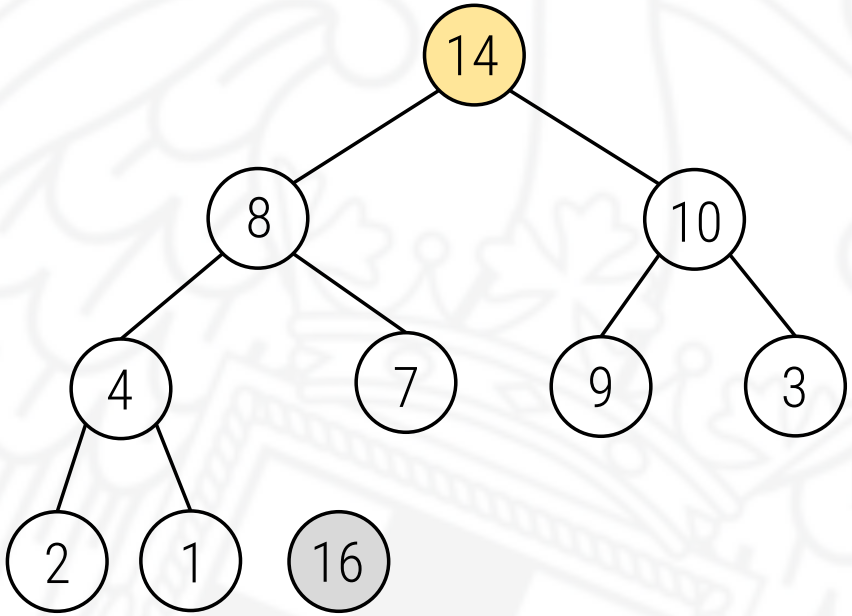
Ordenar



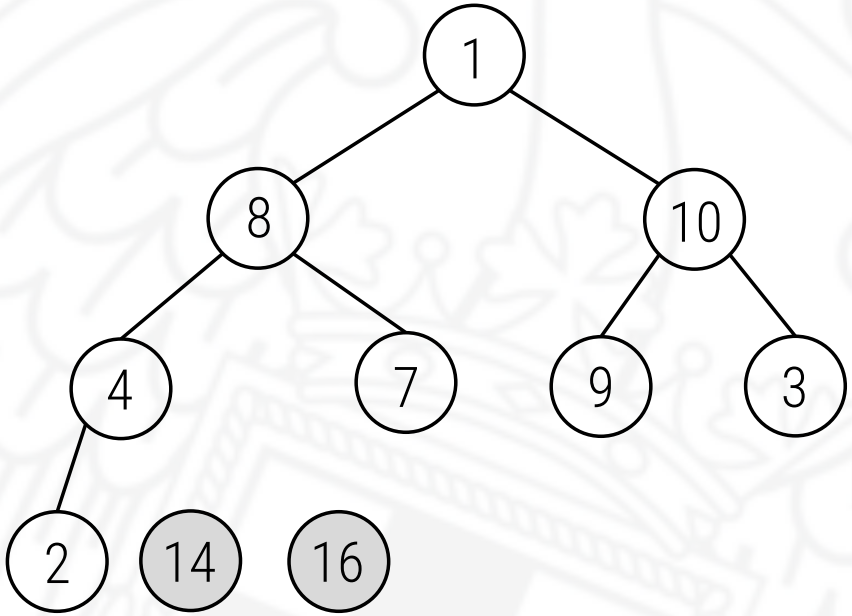
Ordenar



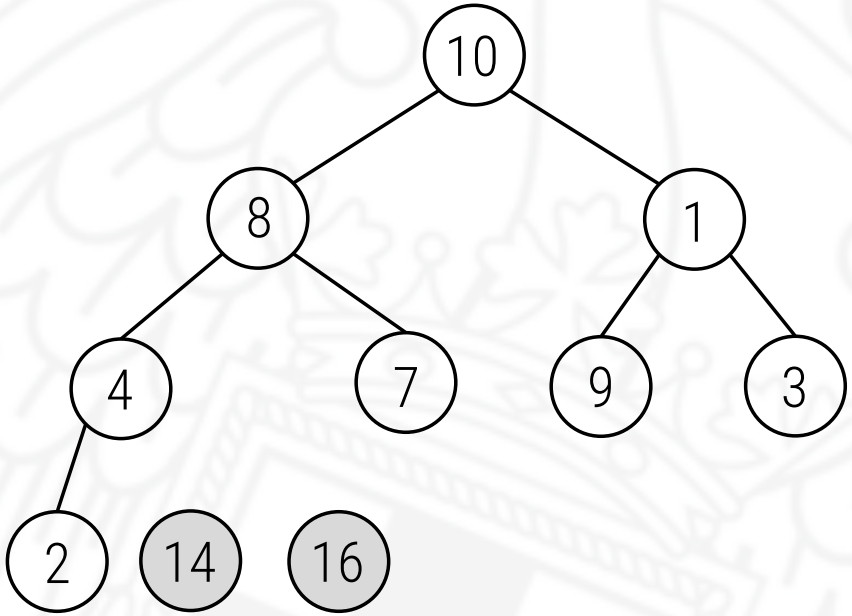
Ordenar



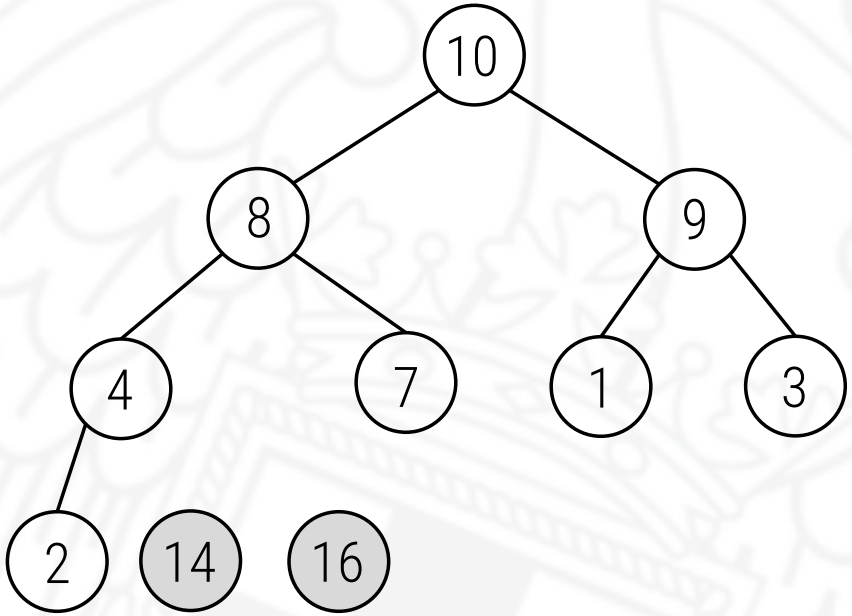
Ordenar



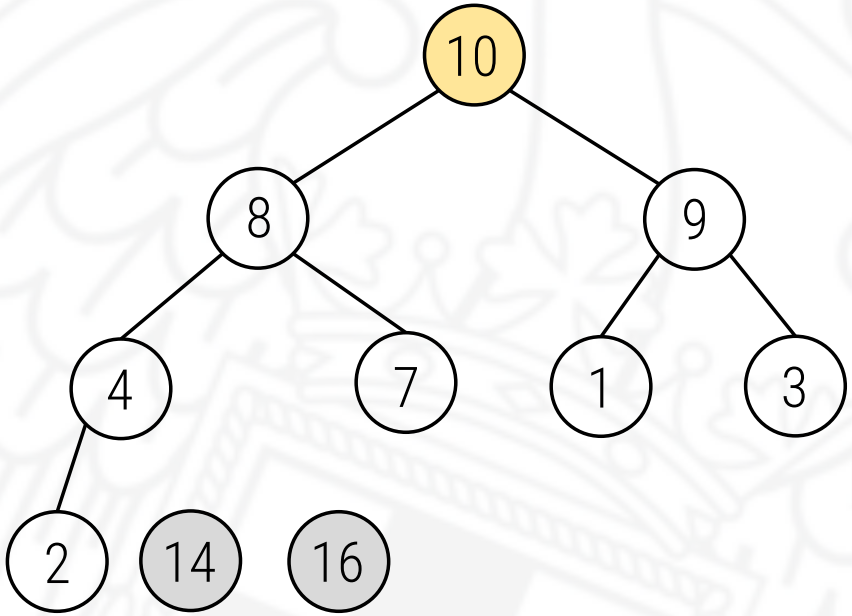
Ordenar



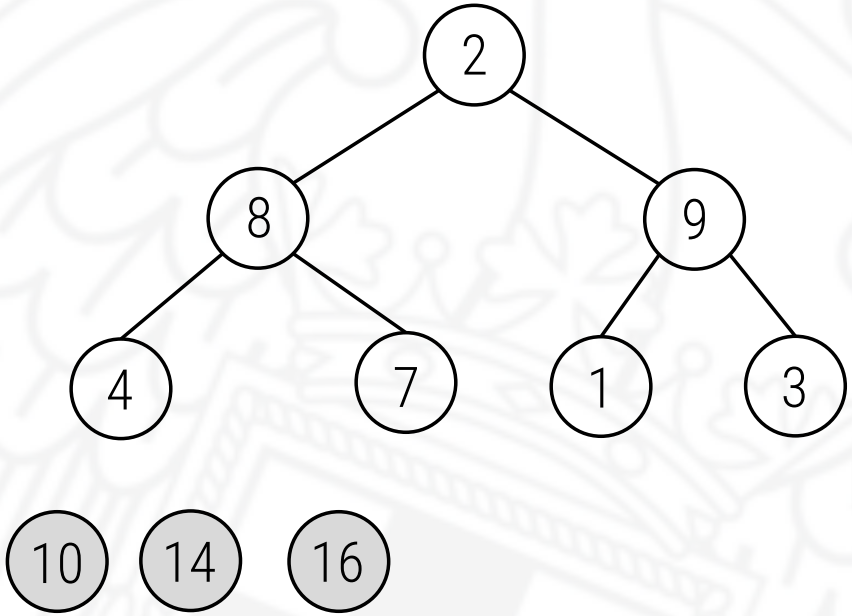
Ordenar



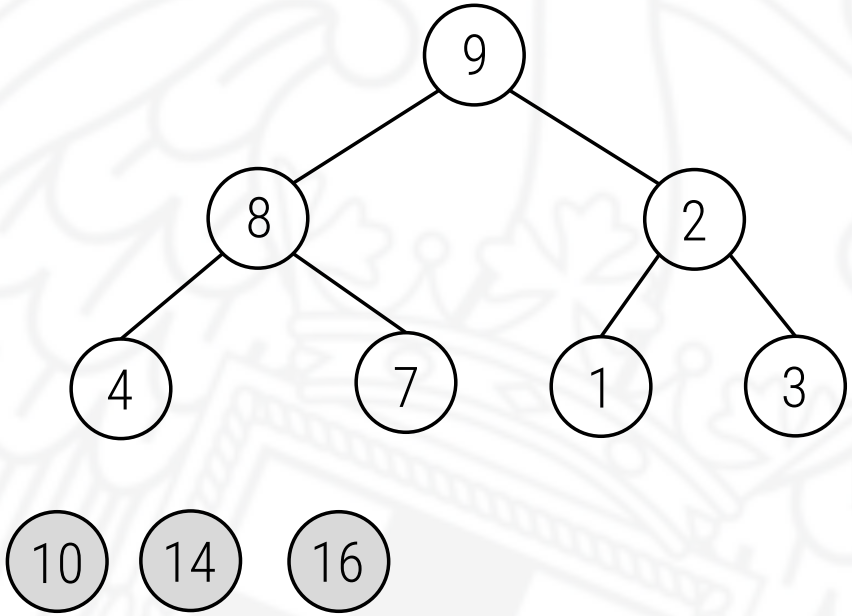
Ordenar



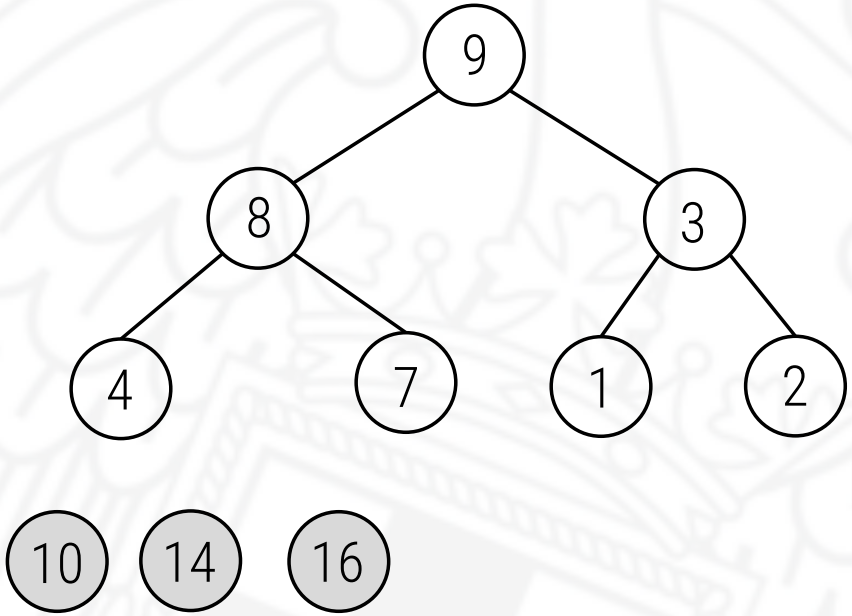
Ordenar



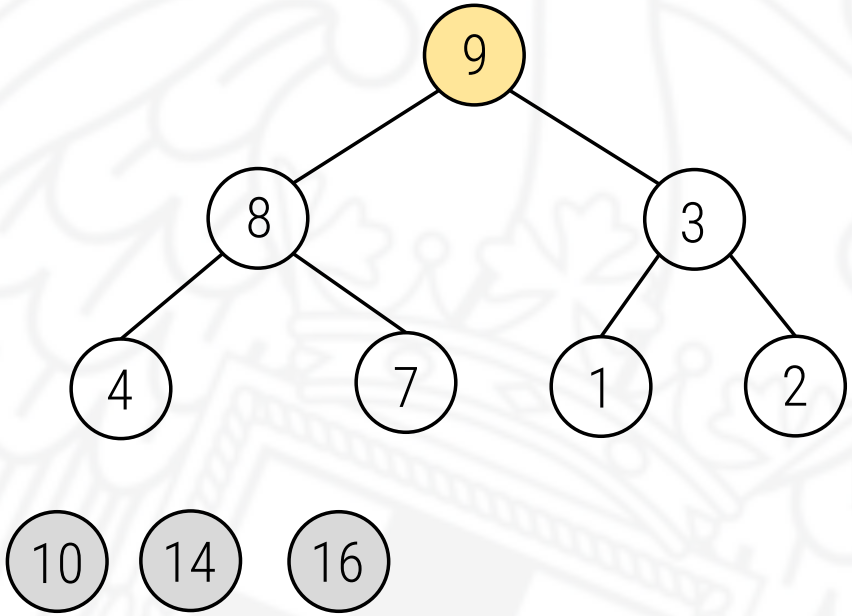
Ordenar



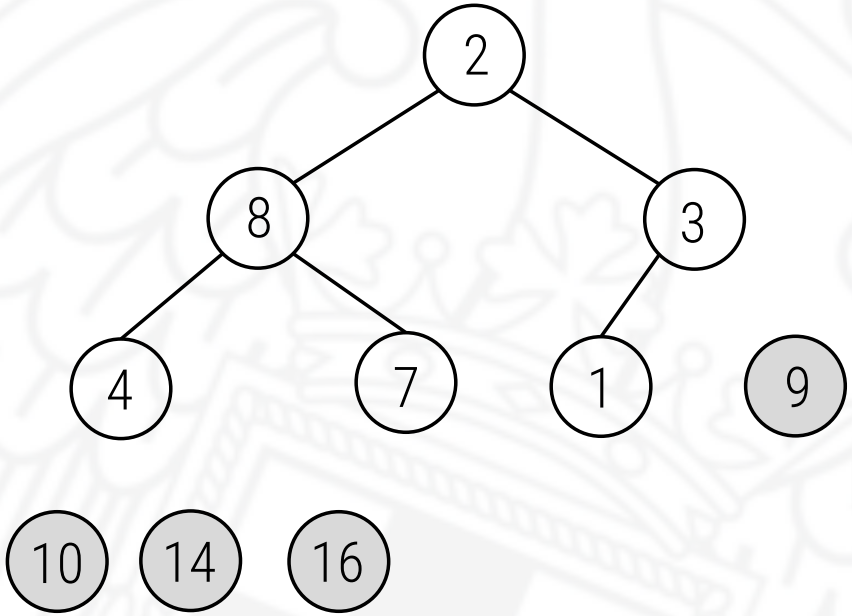
Ordenar



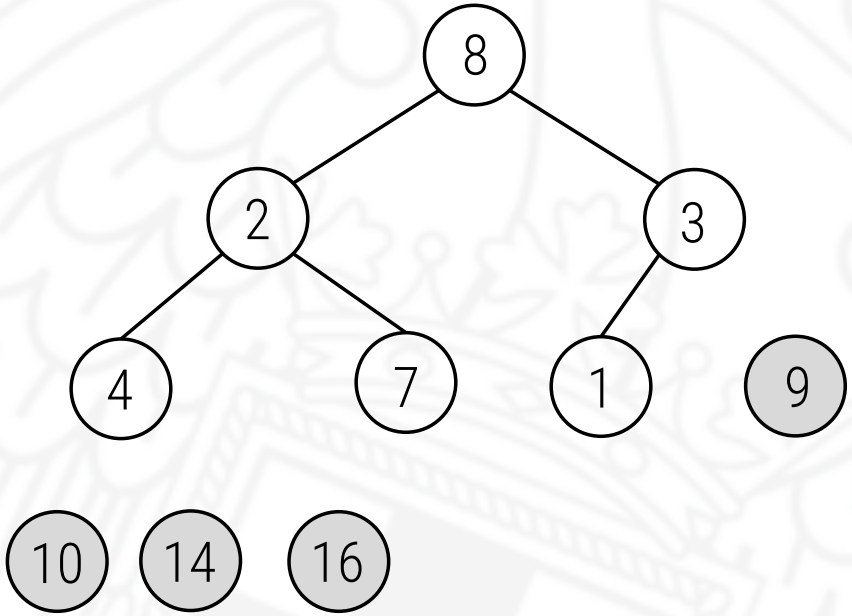
Ordenar



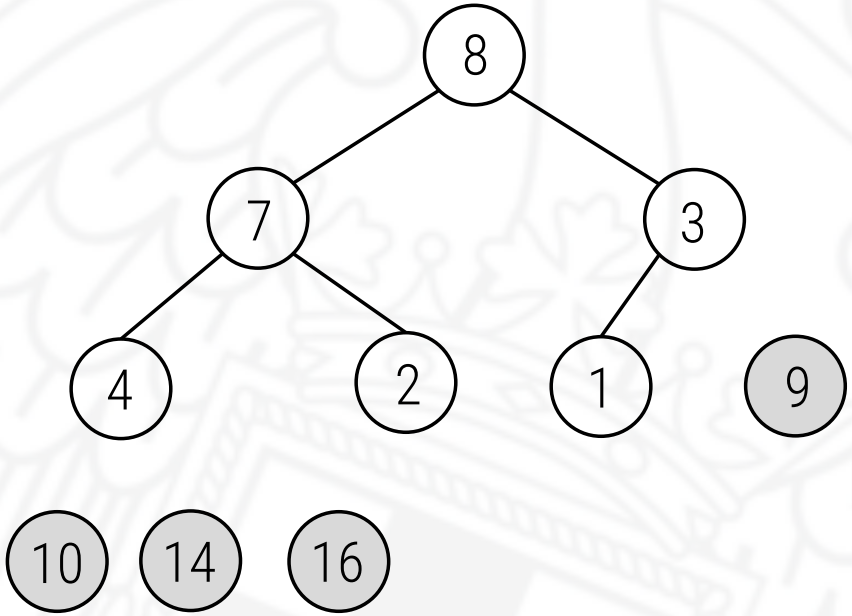
Ordenar



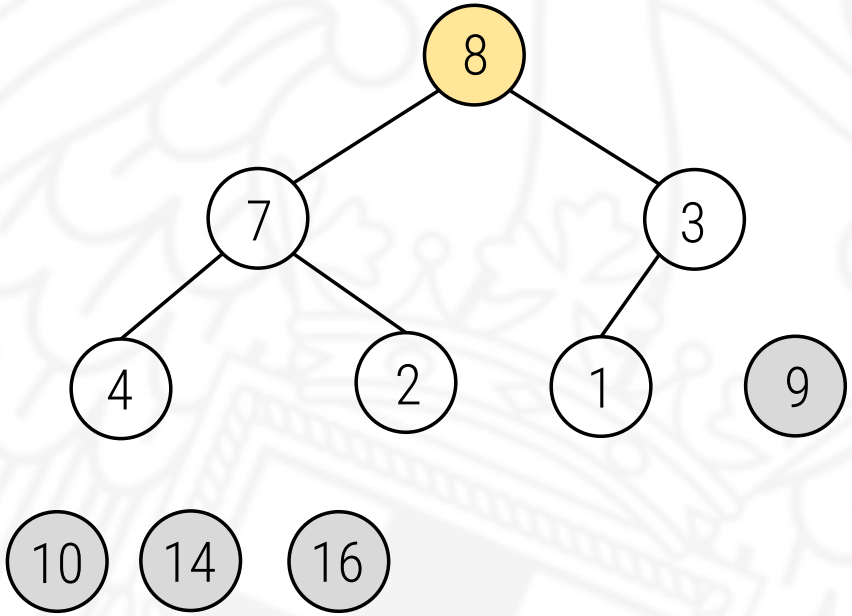
Ordenar



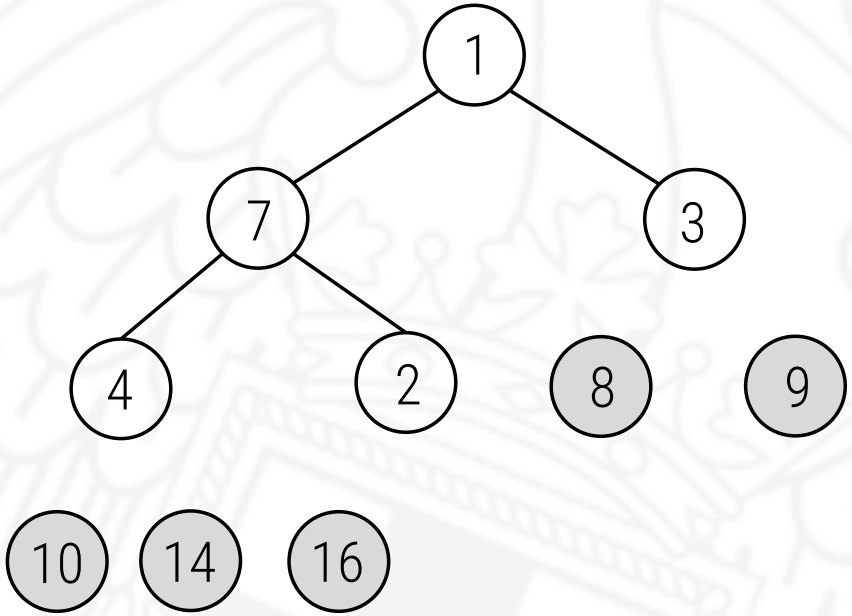
Ordenar



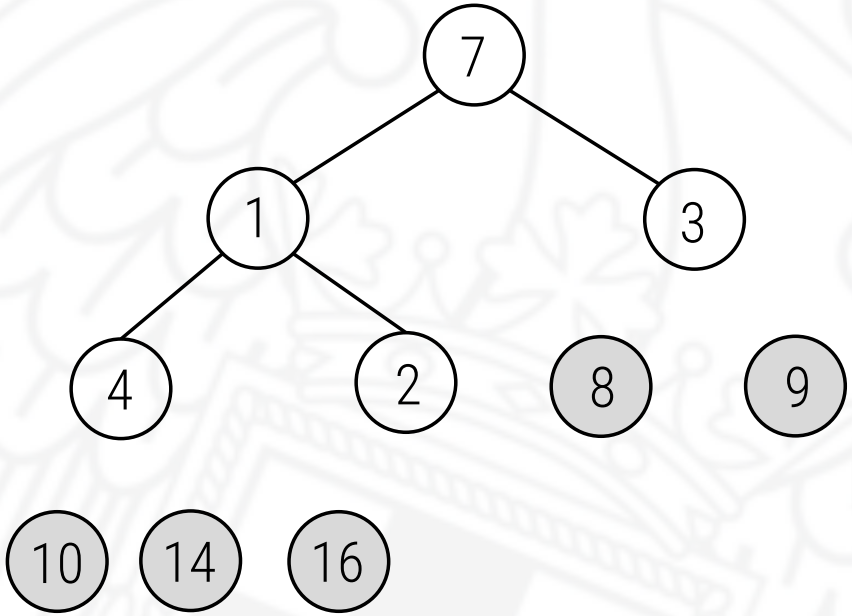
Ordenar



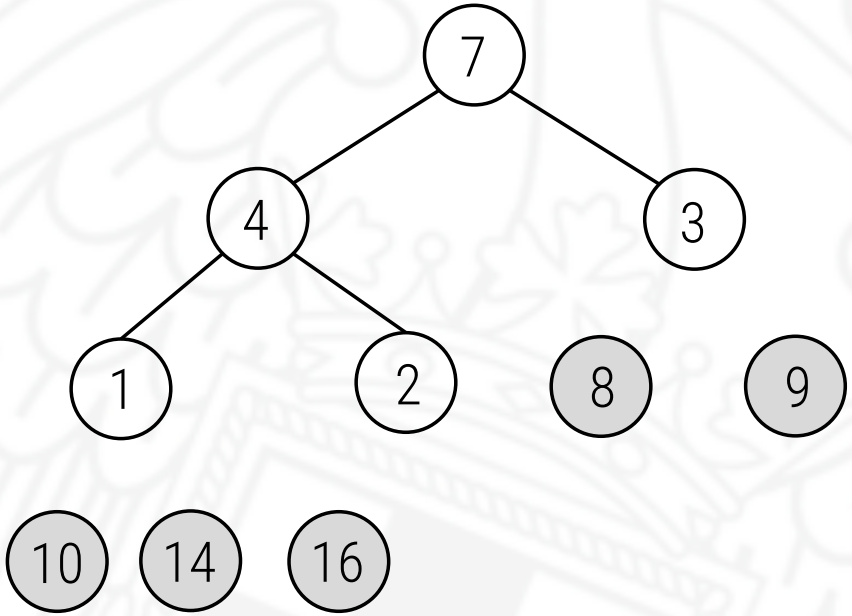
Ordenar



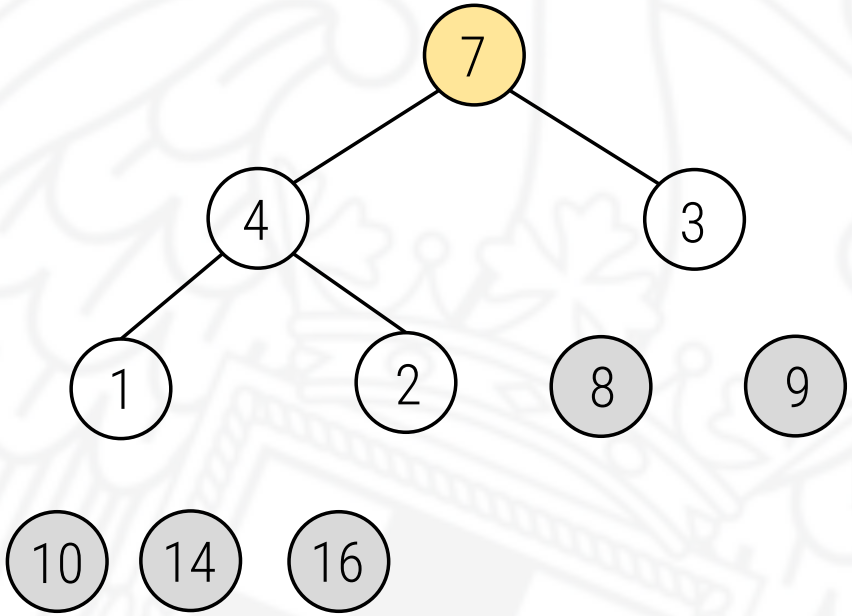
Ordenar



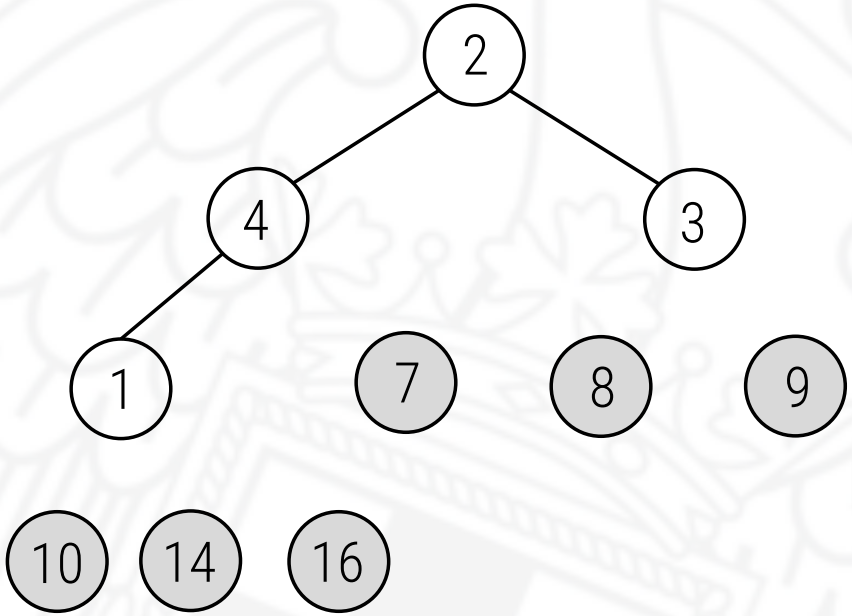
Ordenar



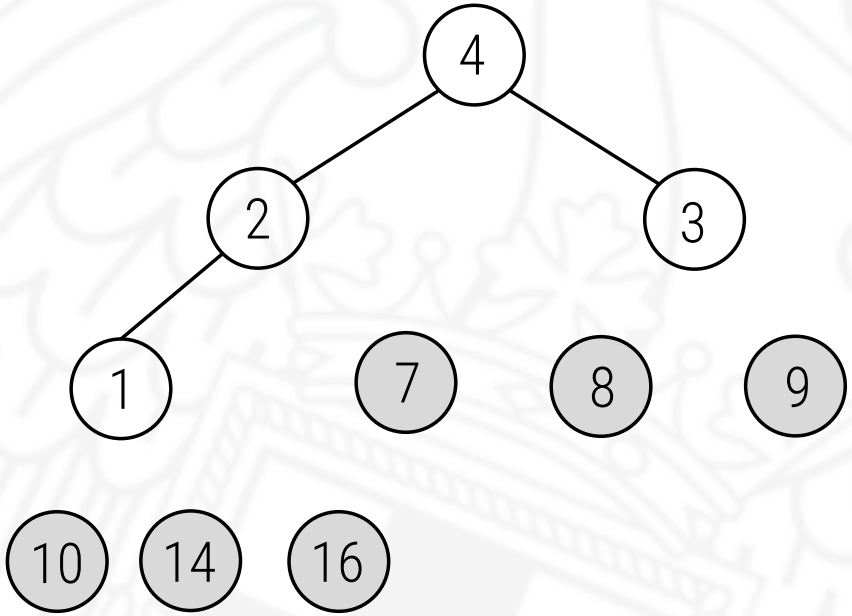
Ordenar



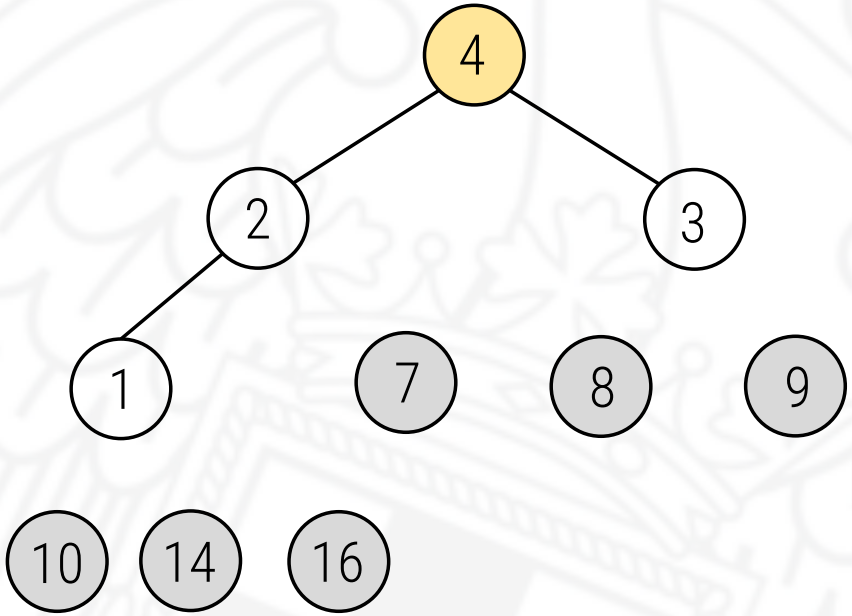
Ordenar



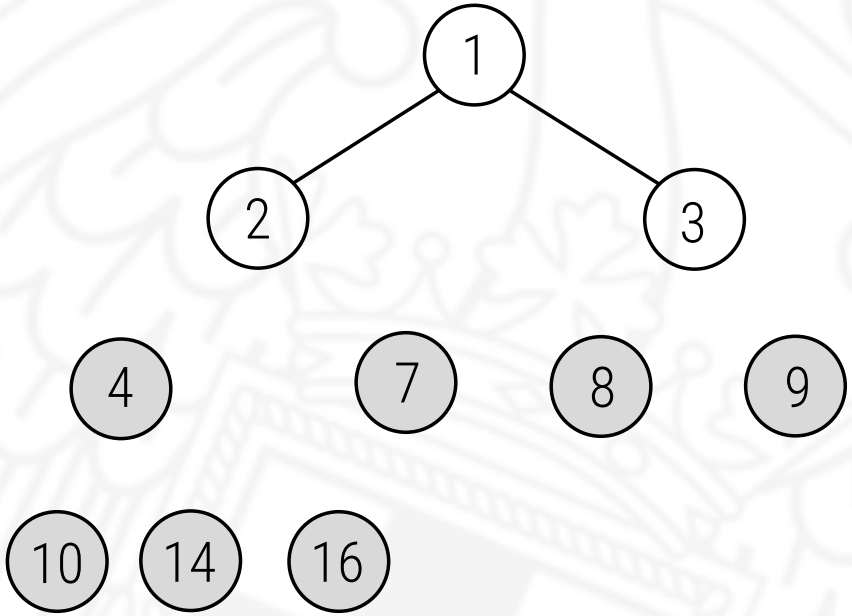
Ordenar



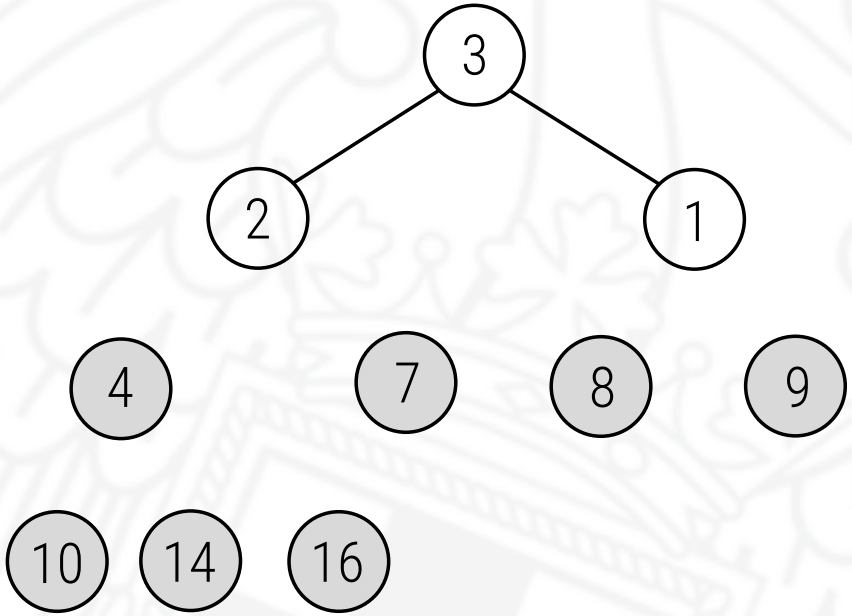
Ordenar



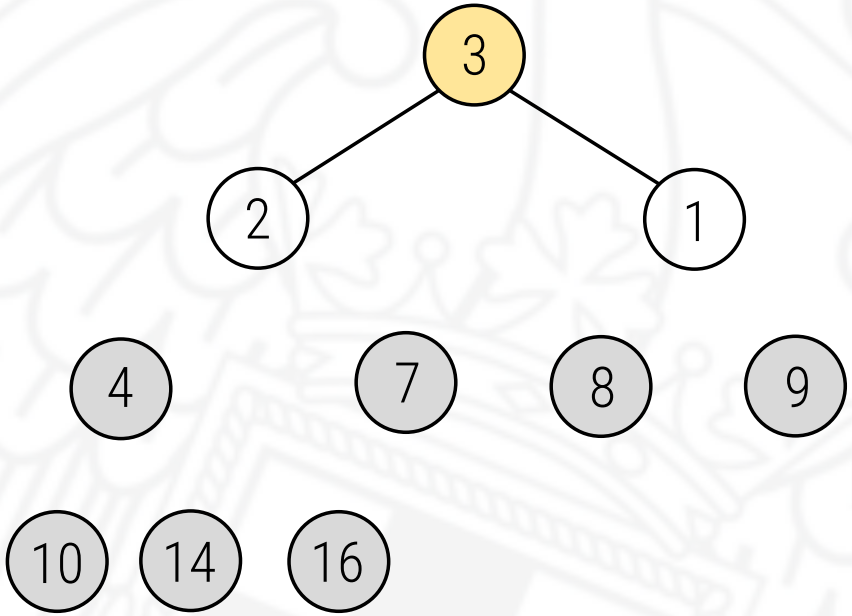
Ordenar



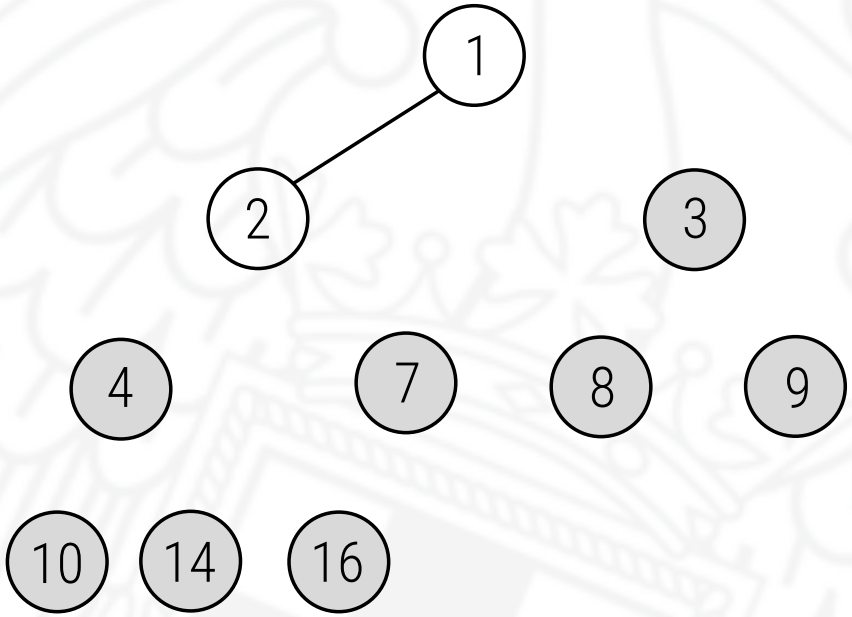
Ordenar



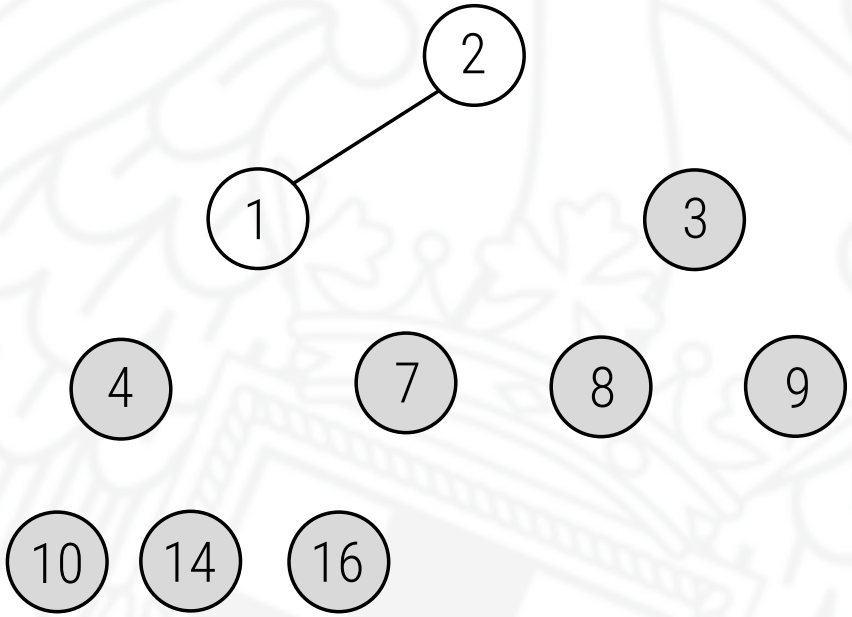
Ordenar



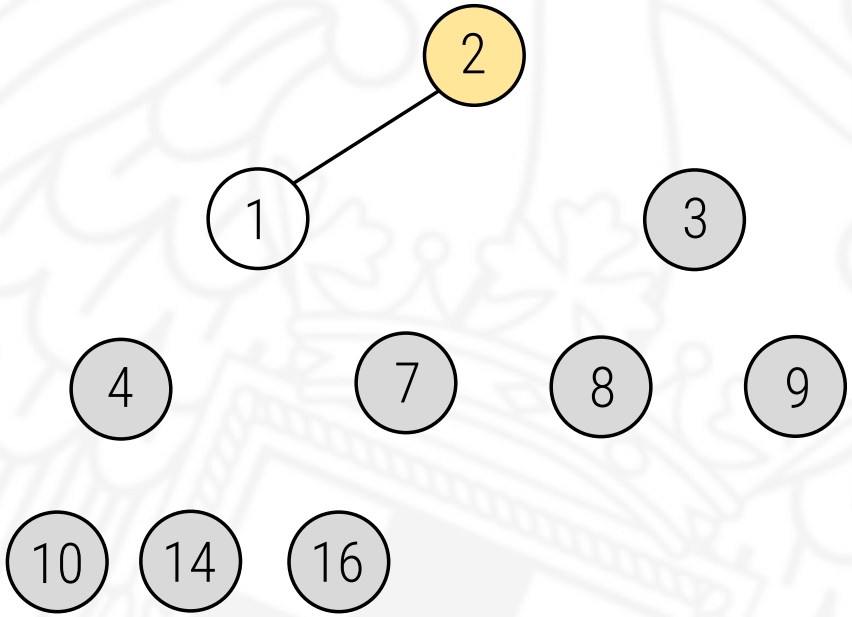
Ordenar



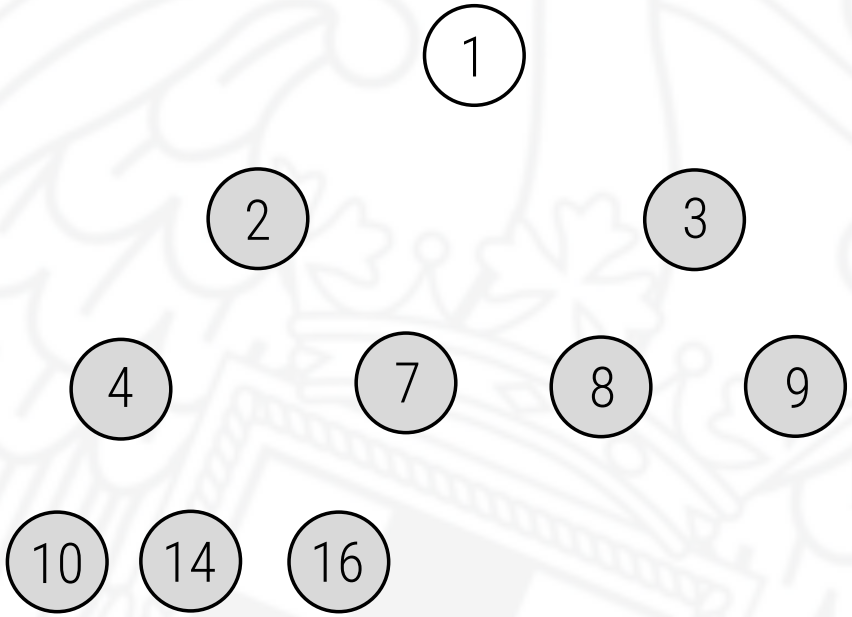
Ordenar



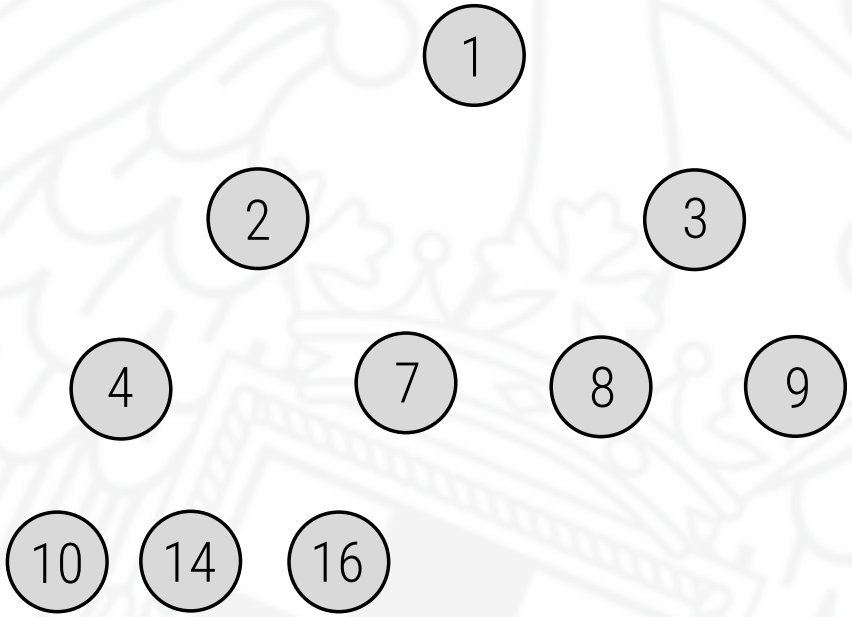
Ordenar



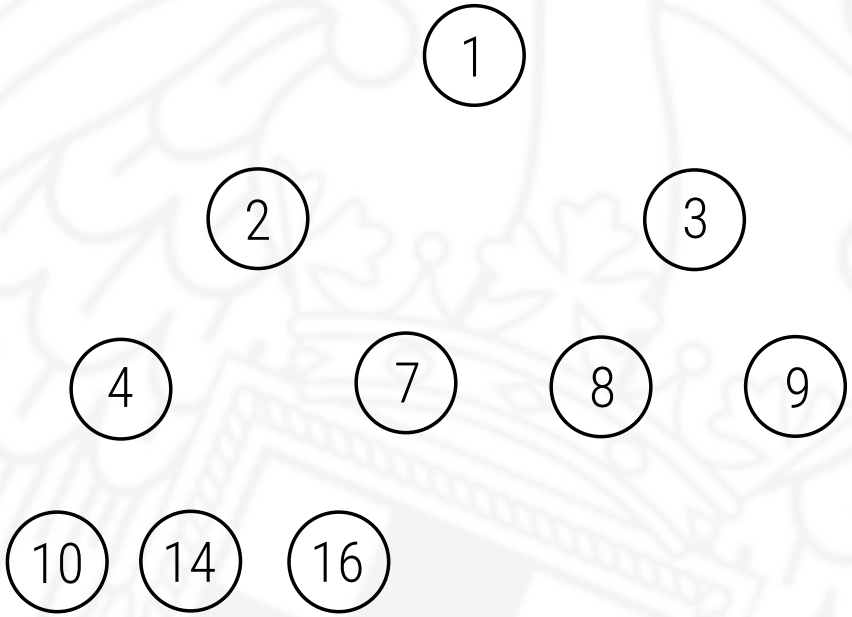
Ordenar



Ordenar



Ordenar



1	2	3	4	7	8	9	10	14	16

Implementación

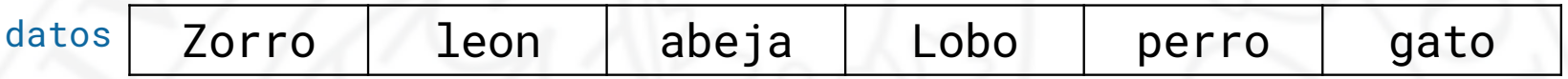
```
template <typename T, typename Comparador = std::less<T>>
void heapsort(std::vector<T> & v, Comparador cmp = Comparador()) {
    // monticulizar
    for (int i = (v.size() - 1) / 2; i >= 0; --i)
        hundir_max(v, v.size(), i, cmp);
    // ordenar
    for (int i = v.size() - 1; i > 0; --i) {
        std::swap(v[i], v[0]);
        hundir_max(v, i, 0, cmp);
    }
}
```

Implementación

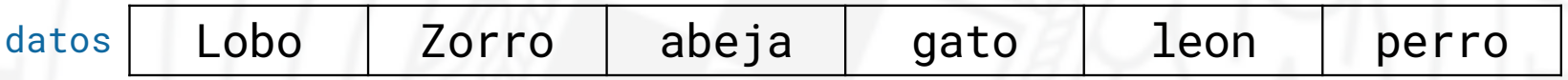
```
template <typename T, typename Comparador>
void hundir_max(std::vector<T> & v, int N, int j, Comparador cmp) {
    // montículo en v en posiciones de 0 a N-1
    T elem = v[j];
    int hueco = j;
    int hijo = 2*hueco + 1; // hijo izquierdo, si existe
    while (hijo < N) {
        // cambiar al hijo derecho si existe y va antes que el izquierdo
        if (hijo + 1 < N && cmp(v[hijo], v[hijo + 1]))
            ++hijo;
        // flotar el hijo mayor si va antes que el elemento hundiéndose
        if (cmp(elem, v[hijo])) {
            v[hueco] = v[hijo];
            hueco = hijo; hijo = 2*hueco + 1;
        } else break;
    }
    v[hueco] = elem;
}
```

Ejemplo

```
vector<string> datos;
```




```
heapsort(datos);
```



Ejemplo

```
string a_minusculas(string s) {
    for (char & c : s) c = tolower(c);
    return s;
}

class ComparaString {
public:
    bool operator()(string a, string b) {
        return a_minusculas(a) < a_minusculas(b);
    }
};

heapsort(datos, ComparaString());
```

datos	abeja	gato	leon	Lobo	perro	Zorro

Ejemplo mejorado

```
class ComparaString { public:
   bool operator()(std::string const& a, std::string const& b) {
      int i = 0;
      while (i != a.length()) {
         if (i == b.length() || tolower(b[i]) < tolower(a[i])) return false;
         else if (tolower(a[i]) < tolower(b[i])) return true;
         ++i;
      }
      return i != b.length();
   }
};

heapsort(datos, ComparaString());
```

datos	abeja	gato	leon	Lobo	perro	Zorro