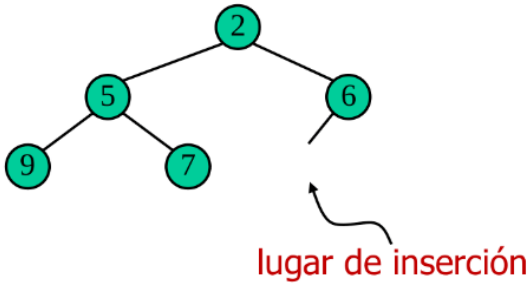
Apuntes/Heap binaria

Es una implementación de colas de prioridad que no usa punteros y permite implementar ambas operaciones con O(log N) operaciones en el peor caso. Consiste en almacenar un árbol binario completo en un arreglo tal que: su raíz esté almacenada en la posición 1; para un elemento que está en la posición i su hijo izquierdo está en **2*i**, el derecho en **2*h+1**, y el padre en **i/2**

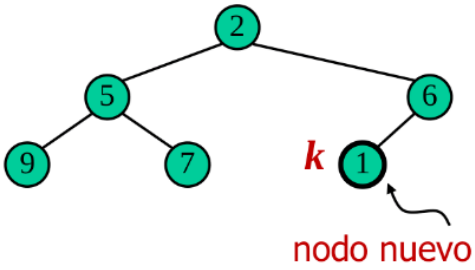
- **Propiedad estructural** → *una heap es un árbol binario completo*
- **Propiedad de orden**
  - El elemento mínimo está almacenado en la raíz
  - El dato almacenado en cada nodo es menor o igual al de sus hijos
  - En caso contrario (elementos máximos) se usa la propiedad inversa
- **Ventajas**
  - No necesita usar punteros
  - Fácil implementación de las operaciones Insert y DeleteMin

Operación: Insert

- El dato se inserta como último ítem en la heap
  - La propiedad de la heap puede ser violada
- Se debe hacer un filtrado hacia arriba para restaurar la propiedad de orden



Inserto el 1



Percolate up

Si ingreso un nuevo nodo, y el valor de ese nodo es menor que el de su padre (i/2), entonces intercambio los lugares, y hago lo mismo hasta llegar a la raíz, con la finalidad de insertar correctamente el nodo.

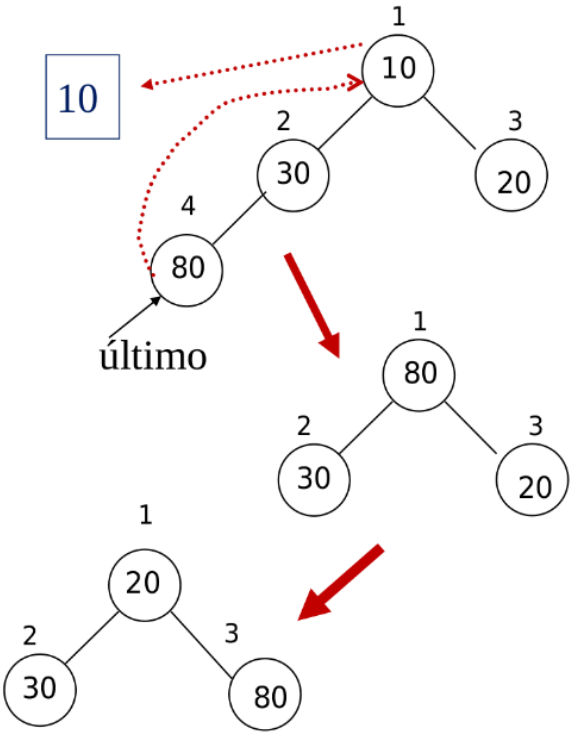
- El algoritmo tiene O(log N) intercambios

```
insert (Heap h, Comparable x) {
    h.tamaño = h.tamaño + 1;
    h.dato[h.tamaño] = x;
    percolate_up(h, h.tamaño)
} //End del insert

//Método del filtrado hacia arriba
percolate_up (Heap, h, Integer i) {
    temp = h.dato[i];
    //Mientras el nodo padre no sea la raíz y el nodo padre s
    while (i/2 > 0 & h.dato[i/2] > temp) {
        h.dato[i] = h.dato[i/2];
        i = i/2;
        }
    h.dato[i] = temp; //Ubicación correcta del elemento a fil
}
```

Operación: DeleteMin

- Guardo el dato de la raíz
- Elimino el último elemento y lo almaceno en la raíz
- Se debe hacer un filtrado hacia abajo para restaurar la propiedad de orden



Percolate down

Su lógica es similar al filtrado hacia arriba. Restaura la propiedad de orden intercambiando el dato de la raíz hacia abajo a lo largo del camino que contiene los hijos mínimos. El filtrado termina cuando se encuentra el lugar correcto dónde insertarlo. Ya que el algoritmo recorre la altura de la heap, tiene O(log N) operaciones de intercambio.

```
delete_min (Heap h, Comparable e) {
    if (h.tamaño > 0) {
        e = h.dato[1]; //Recuperar nodo raíz
        h.dato[1] = h.dato[h.tamaño];
        h.tamaño = h.tamaño - 1;
        percolate_down(h, 1);
    }
}

percolate_down (Heap h, int p) {
    candidato = h.dato[p];
    stop_perc = false;
```

```
    while (2*p <= h.tamaño) and (not stop_perc) {
    h_min = 2 * p; // buscar el hijo con clave menor
    if (h_min != h.tamaño) {
        if (h.dato[h_min+1] < h.dato[h_min])
            h_min++;
        //Percolate down
        if (candidato > h.dato[min]) {
            h.dato[p] = h.dato[h_min];
            p = h_min;
        }
    } else {
        stop perc = true;
    }
h.dato[p] = candidato;
}
```

Construcción de una heap

*Hay dos formas de construir una heap a partir de una lista de n elementos, la primera realiza (n log n) operaciones (n * log n ya que la altura del árbol es proporcional al logaritmo en base 2 de la cantidad de elementos que contiene):*

- Se pueden insertar los elementos de a uno
- Se puede usar un algoritmo de orden lineal → **BuildHeap**

BuildHeap

Esencialmente consiste en: insertar los elementos desordenados en un árbol binario completo y filtrar hacia abajo cada uno de sus elementos.

- Se elige el menor de los hijos
- Se compara el menor de los hijos con el padre
- Se empieza filtrando desde el elemento que está en la posición (*tamaño/2*)

En el filtrado de cada nodo recorremos su altura (en el peor caso). Para acotar la cantidad de operaciones del algoritmo, debemos calcular la suma de las alturas de todos los nodos. Orden lineal

Ordenación de vectores usando Heap

Dado un conjunto de n elementos y se los quiere ordenar en forma creciente, existen dos alternativas:

- Algoritmo que usa una heap y requiere una cantidad aproximada de (n log n) operaciones
    - Requiere el doble de espacio
- Utilizar el algoritmo **HeapSort**

HeapSort

Recibe una MaxHeap con los elementos que se desean ordenar, y va intercambiando el último elemento con el primero, decrementando el tamaño de la heap y realizando filtrados hacia abajo

Otras operaciones

Decrease/IncreaseKey(x, j, H)

Decrementa/incrementa la clave que está en la posición x de la heap H, en una cantidad de j. Si el valor de la clave es menor o mayor que la de sus hijos, y la heap es una MaxHeap o MinHeap, entonces se va a tener que realizar un filtrado hacia abajo/arriba