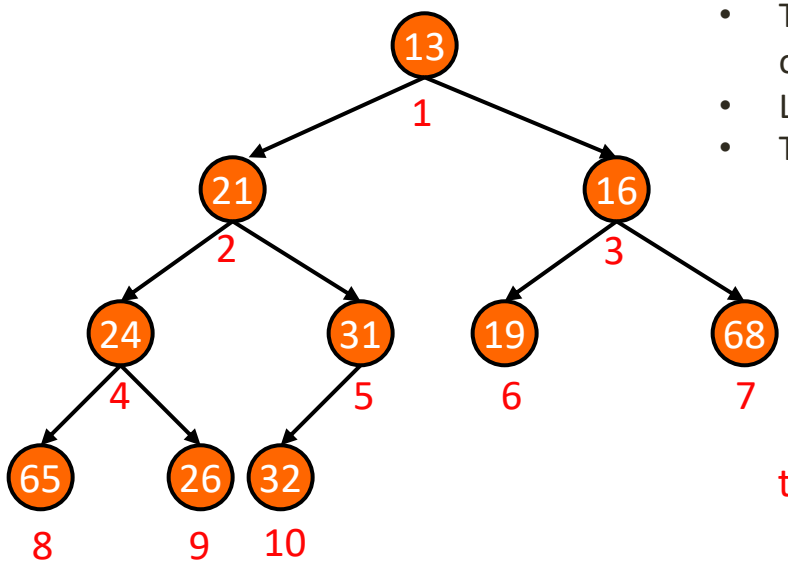
2. Montículo (*Heap*) Binario-*Representación minHeap (cont.)*

Representación

- elArray[1] representa a su Nodo Raíz
- si elArray[i] representa a su i-ésimo Nodo (Por Niveles)
  - Su Hijo Izquierdo es elArray[2i], si $2i \leq$ talla
  - Su Hijo Derecho es elArray[2i+1], si $2i + 1 \leq$ talla
  - Su Padre es elArray[i/2], excepto para i = 1

Propiedad de orden en un minHeap: elArray[padre(i)] <= elArray[i], 2<=i<=talla



Propiedades que se deducen:

- Todo camino desde la raíz a una hoja es una secuencia ordenada (13, 21, 31, 32; 13, 16, 68)
- La raíz es el nodo de valor mínimo
- Todo subárbol de un minHeap es también un minHeap

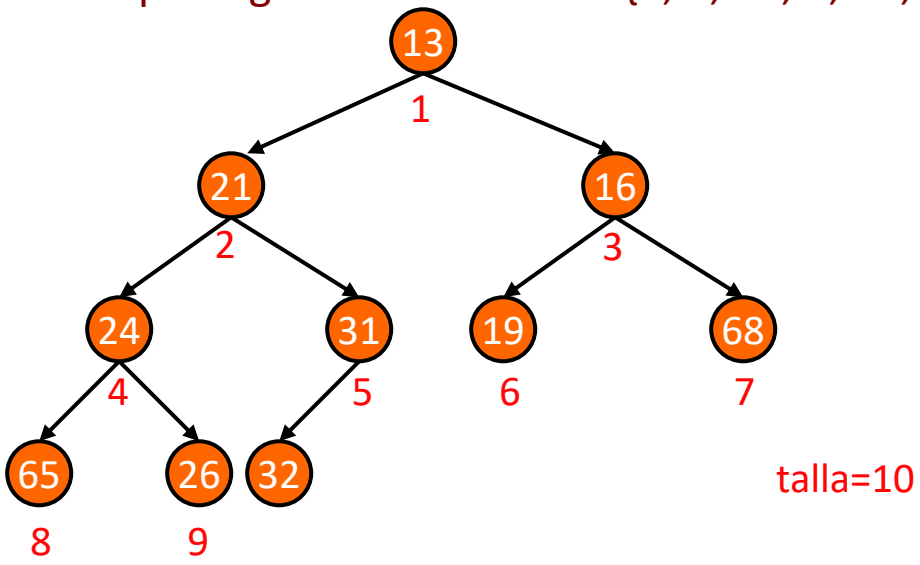
talla=10

1	2	3	4	5	6	7	8	9	10	
	13	21	16	24	31	19	68	65	26	32	....

* E[] elArray
* int talla

2. Montículo (*Heap*) Binario-*Ejercicios*

1. Suponiendo que no hay elementos repetidos y que estamos hablando de un minHeap:
   a) ¿Dónde estará el mínimo?
   b) ¿Dónde estará el máximo?
   c) ¿Cualquier elemento de una hoja será mayor que los elementos de los nodos internos?
   d) ¿Un minHeap es un vector ordenado de forma creciente?
   e) ¿Es un minHeap la siguiente secuencia: {1, 5, 12, 7, 17, 14, 13, 28, 6, 18}



1	2	3	4	5	6	7	8	9	10	
	13	21	16	24	31	19	68	65	26	32	....

* E[] elArray
* int talla

Ejercicio 3: Escribir un método en la clase *MonticuloBinario*, que representa un montículo binario minimal, que obtenga su elemento máximo realizando el mínimo número de comparaciones.

Solución: Este problema es sencillo. El máximo seguro que está en una de las hojas, por lo que es suficiente recorrer las hojas y quedarse con el máximo. La primera hoja estará en la posición talla/2+1 y la última hoja en la posición talla.

```
public E maximo() {
  if (talla == 0) return null;
  int primeraHoja = talla/2 + 1;
  E max = elArray[primeraHoja];
  for (int i = primeraHoja + 1; i <= talla; i++)
    if (max.compareTo(elArray[i]) < 0) max = elArray[i];
  return max;
}
```