

Colas de Prioridad y heaps



PILA FIFO
COLA LIFO

Colas de prioridad

- Numerosas aplicaciones
 - Sistemas operativos, algoritmos de scheduling, gestión de colas en cualquier ambiente, etc.
- La prioridad en general la expresamos con un entero, pero puede ser cualquier otro tipo α con un orden $<_\alpha$ asociado.
- Correspondencia entre máxima prioridad y un valor máximo o mínimo del valor del tipo α

La prioridad bajo os: max es la otra

La importante , depende de implementación.

¿Recordamos la definición del TAD?

Tiene que tener un
orden.

- **TAD Cola de prioridad**($\alpha, <_{\alpha}$)
- **Observadores básicos**
 - vacía?: $\text{colaPrior}(\alpha, <_{\alpha}) \rightarrow \text{bool}$
 - próximo: $\text{colaPrior}(\alpha, <_{\alpha}) c \rightarrow \alpha$ ($\sim \text{vacía?}(c)$)
 - desencolar: $\text{colaPrior}(\alpha, <_{\alpha}) c \rightarrow \text{colaPrior}(\alpha, <_{\alpha})$
($\sim \text{vacía?}(c)$)
- **Generadores**
 - vacía: $\rightarrow \text{colaPrior}(\alpha, <_{\alpha})$
 - encolar: $\alpha \times \text{colaPrior}(\alpha, <_{\alpha}) \rightarrow \text{colaPrior}(\alpha, <_{\alpha})$
- **Otras Operaciones**
 - $\cdot = \text{colaPrior} \cdot : \text{colaPrior}(\alpha, <_{\alpha}) \times \text{colaPrior}(\alpha, <_{\alpha}) \rightarrow \text{bool}$

Implementación de la $=$ dos

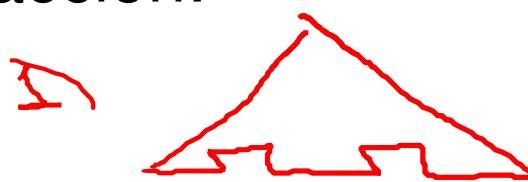
Representación de Colas de Prioridad

- La implementación más eficiente es a través de heaps → es mas sencillo que un AVL
- Heap significa, literalmente, “montón”

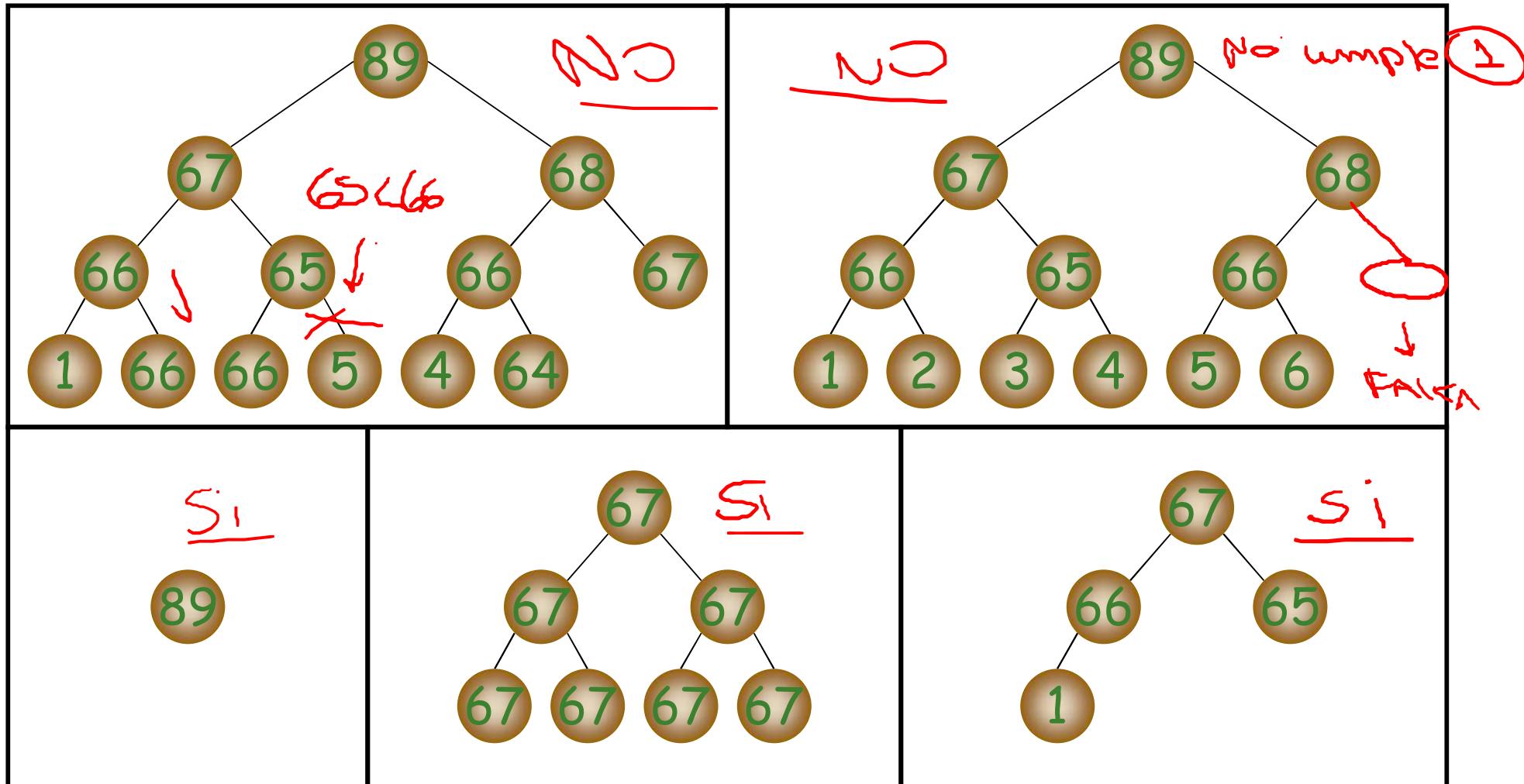


Representación de Colas de Prioridad

- Cola de prioridad($\alpha, <_\alpha$) se representa con heap
- Invariante de representación (condición de heap)
 1. Árbol binario perfectamente balanceado → Meso  
 2. La clave (prioridad) de cada nodo es mayor o igual que la de sus hijos, si los tiene
 3. Todo subárbol es un heap
 4. (no obligatorio): es “izquierdista”, o sea, el último nivel está lleno desde la izquierda.
- (Ojo: ¡no es un ABB, ni una estructura totalmente ordenada!)
- Función de abstracción:
 - Ejercicio (fácil)



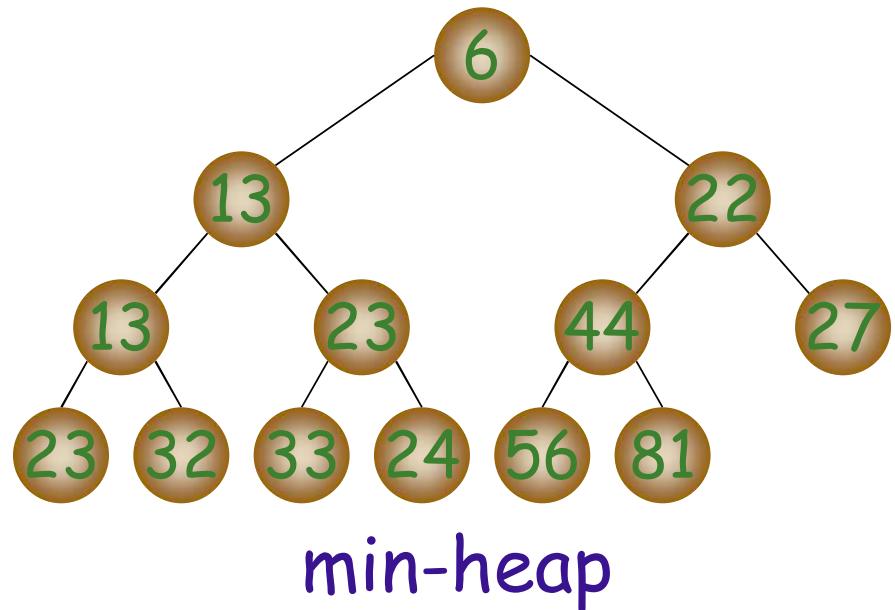
¿Son heaps?



max- y min-heap

- La estructura que estamos usando se llama *max-heap*
- Variante: *min-heap*
 - Cambiar “mayor” por “menor”

el proximo de la cde
se obtiene O(1)



Operaciones sobre un (max-)heap

- Básicamente, las mismas que tenemos definidas en el TAD Cola de Prioridad:
 - Vacía: crea un heap vacío $\textcolor{red}{O(1)}$
 - Próximo: devuelve el elemento de máxima prioridad, sin modificar el heap. $\textcolor{red}{O(1)}$
 - Encolar: agrega un nuevo elemento, hay que restablecer el invariante
 - Desencolar: elimina el elemento de máxima prioridad, hay que restablecer el invariante

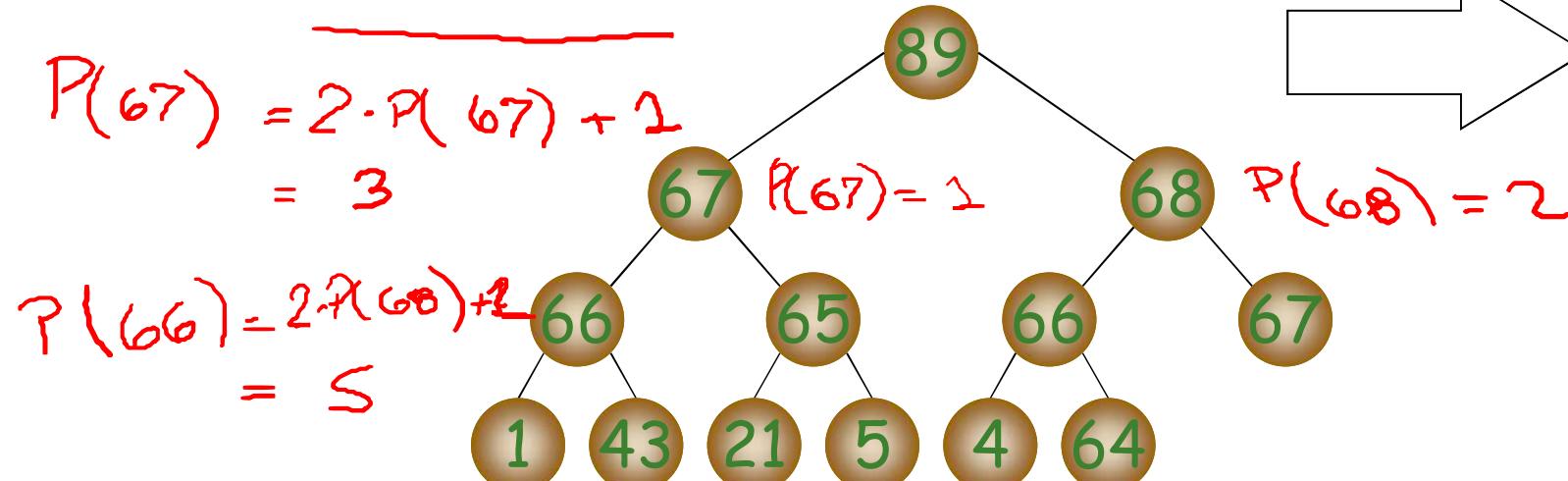
Implementación de heaps

- Todas las representaciones usadas para árboles binarios son admisibles
 - representación con punteros, eventualmente con punteros hijo-padre
 - representación con arrays $\xrightarrow{\text{Implicito}}$
 - particularmente eficiente

Representación con arrays

Por el índice no
hay elementos
null

- Cada nodo v es almacenado en la posición $p(v)$
 - si v es la raíz, entonces $p(v)=0$
 - si v es el hijo izquierdo de u entonces $p(v)=2p(u)+1$
 - si v es el hijo derecho de u entonces $p(v)=2p(u)+2$



89	0
67	1
68	2
66	3
65	4
66	5
67	6
1	7
43	8
21	9
5	10
4	11
64	12

Heaps sobre arrays

Ventajas

- Muy eficientes en términos de espacio (¡ver desventajas!)
- Facilidad de navegación

■ padre $i \rightarrow$ hijos j_{izq} y j_{der}

$$\square j_{izq} = 2i + 1, j_{der} = 2i + 2$$

■ hijo $i \rightarrow$ padre j

$$\square j = \lfloor (i-1) / 2 \rfloor$$

$$P(68) = \frac{P(66) - 1}{2}$$

~~des~~ $\xrightarrow{\text{des}}$ ~~desp~~ $\xrightarrow{\text{desp}}$

Desventaja

- Implementación estática (puede ser necesario duplicar el arreglo (o achicarlo) a medida que se agregan/eliminan elementos)

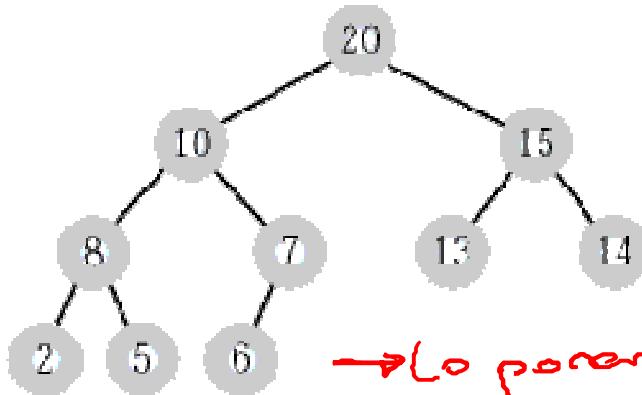
Algoritmos

■ Próximo:

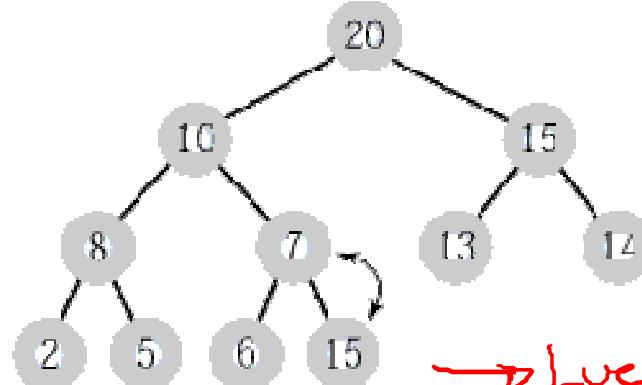
- El elemento de prioridad máxima está en la posición 0 del arreglo
- Operación de costo constante $O(1)$

Algoritmo Encolar (ejemplo)

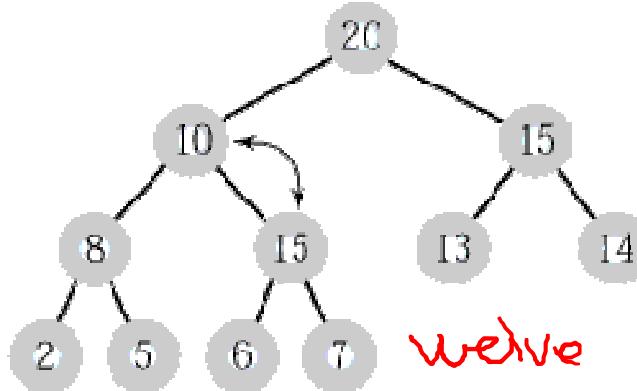
Inserción (IS)



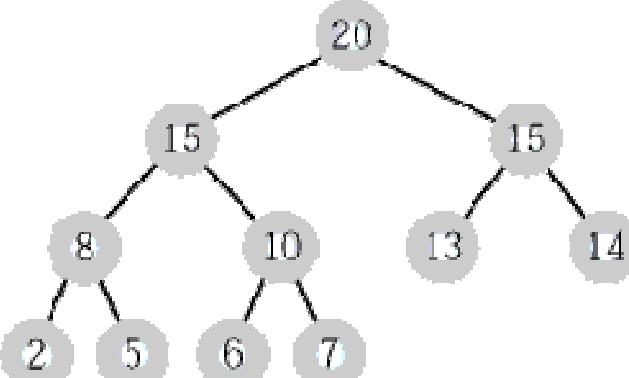
a) → Lo ponemos
en la posición
correcta



b) → Luego esto



c) → Vuelve a rotar
para cumplir
el invarionte



d)

Algoritmo Encolar

$O(\log(n))$

■ Encolar(elemento)

- Insertar elemento al final del heap
- Subir (elemento)

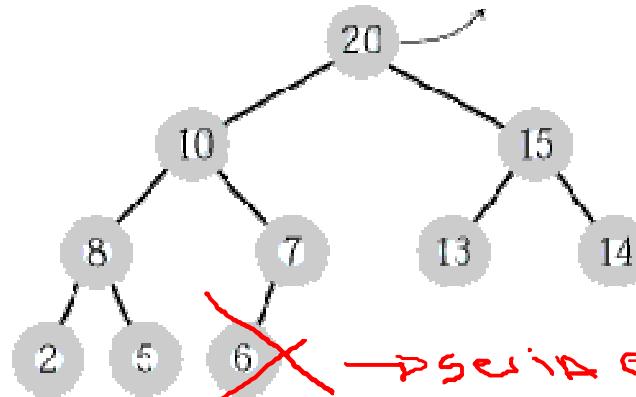
■ Subir(elemento)

- while (elemento no es raíz) y_L(prioridad(elemento) > prioridad(padre(elemento)))
 - Intercambiar elemento con padre

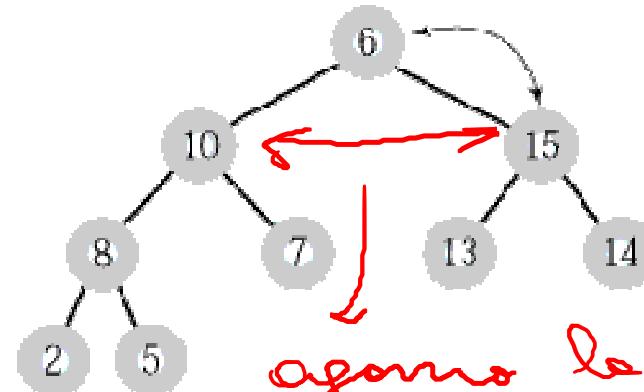
$O(2)$
tenemos un
Puntero que
apunta al
final del
heap

Algoritmo Desencolar (ejemplo)

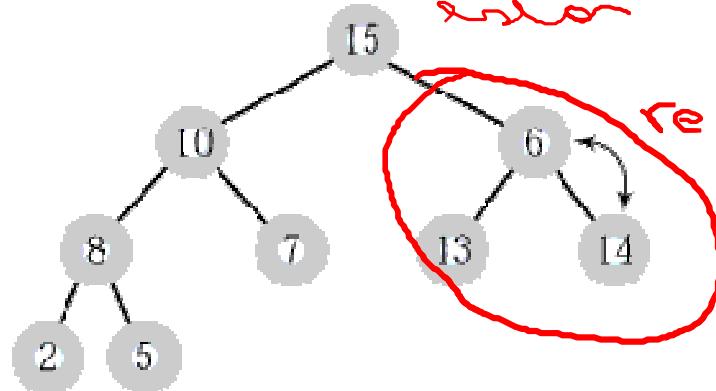
→ eliminar la raíz en max-heap



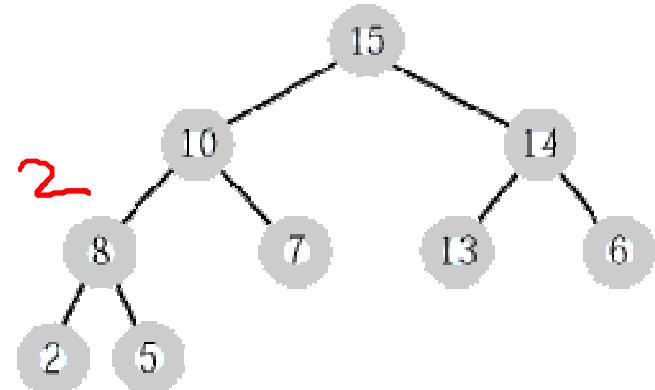
a) Proximo que
no tiene
esta



agorro la raíz
b) Del mayor.



repito
Pass 2



Algoritmo Desencolar

■ Desencolar

- Reemplazar el primer elemento con la última hoja y eliminar la última hoja
- Bajar(raíz)

■ Bajar(p)

- while (p no es hoja) y L ($\text{prioridad}(p) < \text{prioridad}(\text{algun hijo de } p)$)
 - Intercambiar p con el hijo de mayor prioridad

Costos

- Tanto para encolar como para desencolar, proporcionales a la altura del heap, que es.....

Logarítmica

$$O(\lg n)$$

Cada vez salimos menos de una sola de las siguientes proximas

Array2Heap

- Dado un array arr, lo transforma en un array que representa un heap a través de una permutación de sus elementos
- Algoritmo simple

para i desde 1 hasta tam(arr)

encolar(arr[i]); → Func de orden $\mathcal{O}(n)$.

- Costo (utilizando la aproximación de Stirling del factorial):

$$\sum_{i=1}^n \lg i = \lg n! = \frac{\ln n!}{\ln 2} \approx \frac{1}{\ln 2} (n \ln n - n) = \Theta(n \lg n)$$

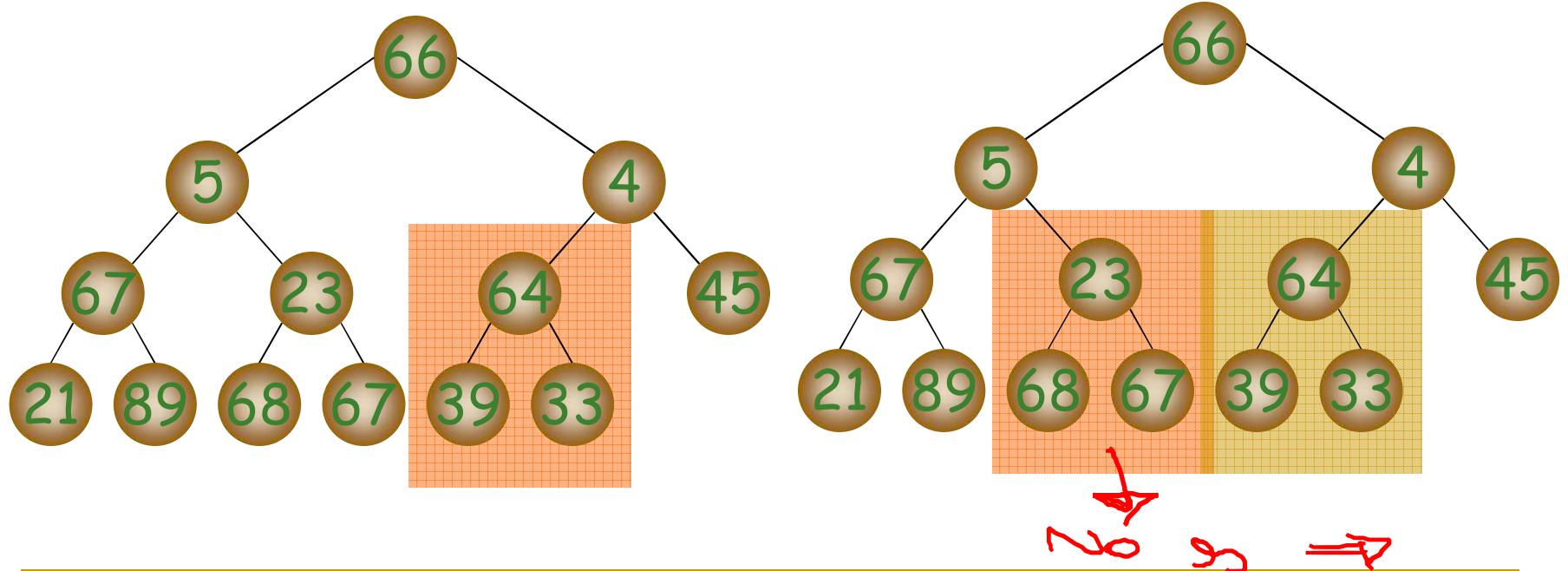
Tenemos n elementos y hacemos por $\lg(n)$

Array2Heap/2

- El algoritmo de Floyd, en cambio, está basado en la idea de aplicar la operación bajar a árboles binarios tales que los hijos de la raíz son raíces de heaps.
- Progresivamente se “heapifican” (“heapify”) los subárboles con raíz en el penúltimo nivel, luego los del antepenúltimo, etc.
 - Estrategia bottom-up

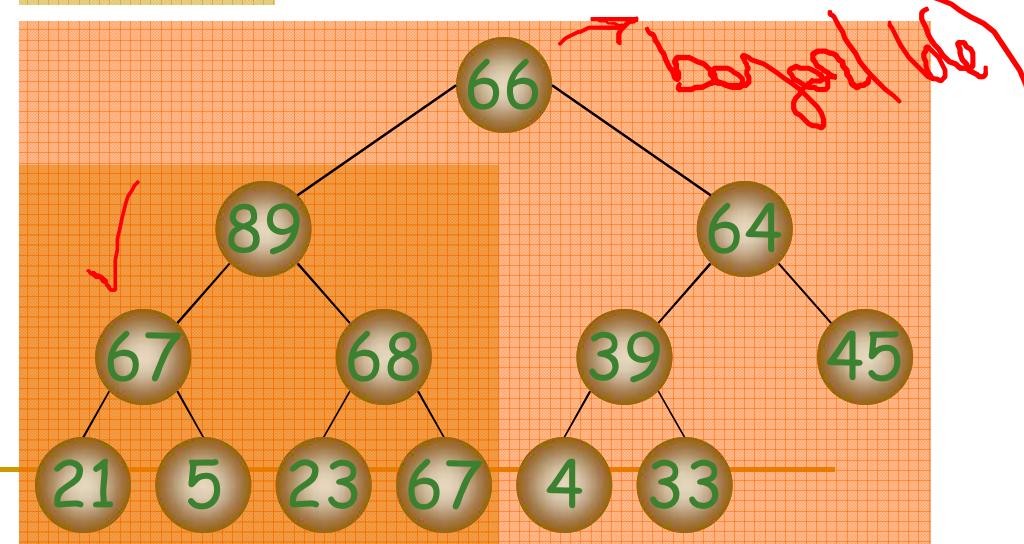
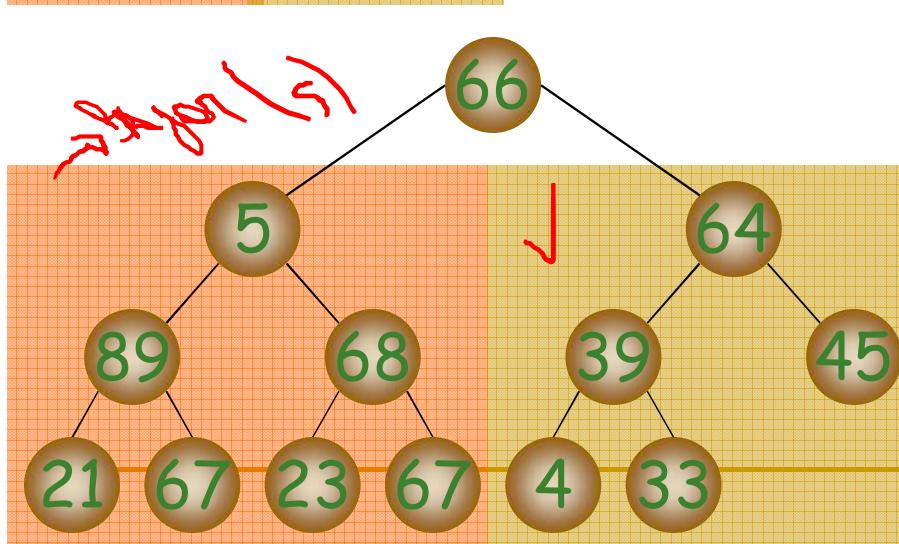
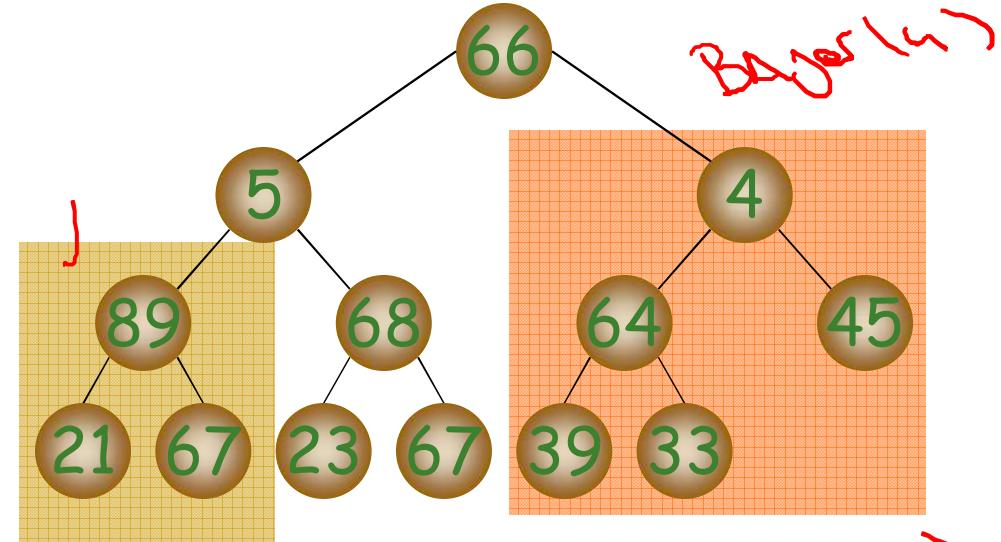
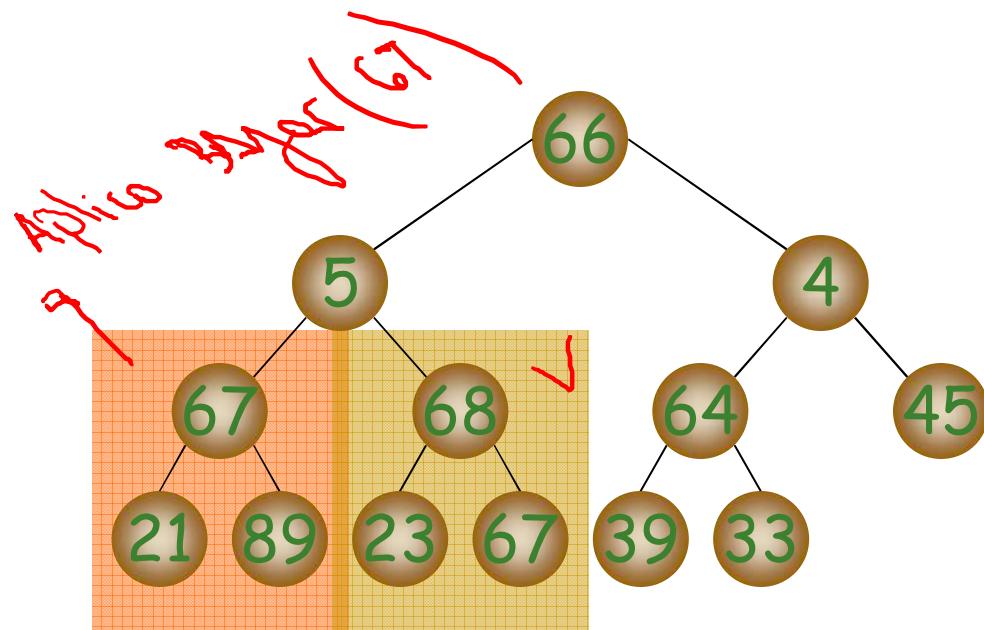
algoritmo de Floyd

0	1	2	3	4	5	6	7	8	9	10	11	12
66	5	4	67	23	64	45	21	89	68	67	39	33

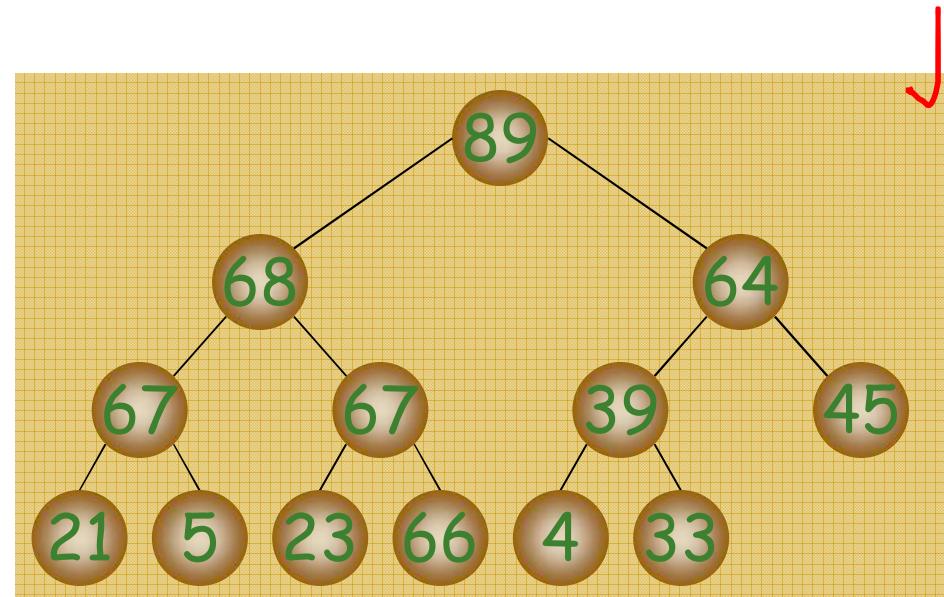


Datos bajar (23)

algoritmo de Floyd/2



algoritmo de Floyd/3

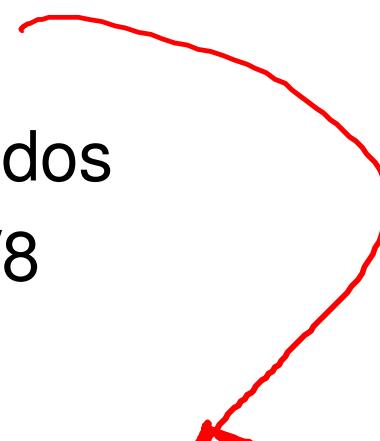


0	1	2	3	4	5	6	7	8	9	10	11	12
89	68	64	67	67	39	45	21	5	23	66	4	33

↳ Permutación del Arreglo Original

Análisis del Algoritmo de Floyd

- Caso peor: cada llamada a bajar hace el máximo número de intercambios
- Suponemos un heap con $n = 2^k - 1$ nodos
(árbol binario completo de altura k)
 - En el último nivel hay $(n + 1)/2$ hojas
 - En el penúltimo nivel hay $(n + 1)/4$ nodos
 - En el antepenúltimo nivel hay $(n + 1)/8$
 - Y así sucesivamente....

$$n_h = n_i + 1$$


Análisis del algoritmo de Floyd/2

- Una llamada de bajar sobre un nodo de nivel i , provoca como máximo $k - i$ intercambios (op. dominante)
 - 1 intercambio si i es penúltimo nivel, 2 si i es el antepenúltimo, ..., $k-1$ intercambios si $i=1$
- # max de intercambios =
 $(\# \text{ nodos en el penúltimo nivel}) \cdot 1 +$
 $(\# \text{ nodos en el antepenúltimo nivel}) \cdot 2 + \dots +$
 $(\# \text{ nodos en el nivel } 2) \cdot (k-2) +$
 $(\# \text{ nodos en el nivel } 1) \cdot (k-1)$, con $k = \lg(n+1)$

Análisis del algoritmo de Floyd/2

■ # max de intercambios =

$$\begin{aligned} & ((n + 1) / 4) \cdot 1 + ((n + 1) / 8) \cdot 2 + \dots + \\ & 2 \cdot (\lg(n + 1) - 2) + 1 \cdot (\lg(n + 1) - 1) \end{aligned}$$

■ # max de intercambios =

$$\begin{aligned} \sum_{i=2}^{\log(n+1)} \frac{n+1}{2^i} (i-1) &= (n+1) \sum_{i=2}^{\log(n+1)} \frac{i-1}{2^i} \\ &= (n+1) \left(\sum_{i=2}^{\log(n+1)} \frac{i}{2^i} - \sum_{i=2}^{\log(n+1)} \frac{1}{2^i} \right) \end{aligned}$$

Sumo el costo
de cada nivel

Análisis del algoritmo de Floyd/3

considerando que

$$\text{Asintótico} \quad \sum_{i=2}^{\infty} \frac{i}{2^i} = \frac{3}{2} \quad \sum_{i=2}^{\log(n+1)} \frac{1}{2^i} > 0$$

Deducimos que

$$(n+1) \sum_{i=2}^{\log(n+1)} \frac{i-1}{2^i} < (n+1) \left(\frac{3}{2} - \sum_{i=2}^{\log(n+1)} \frac{1}{2^i} \right) < \frac{3}{2}(n+1)$$

$\Rightarrow \# \max \text{ de intercambios} = O(n)$

Costo

Floyd

Aplicaciones del algoritmo de Floyd

■ Implementación de operaciones no standard

- Eliminación de una clave cualquiera
- Pueden ser requeridas en algún contexto
- Ejemplo: kill de un proceso dado su PID → *matar un proceso*
- O(n) para encontrar la clave, O(1) para eliminarla, O(n) para reconstruir el invariante de representación con el algoritmo de Floyd

■ Ordenamiento de un array

- Ya lo veremos.....