

Árboles Heaps

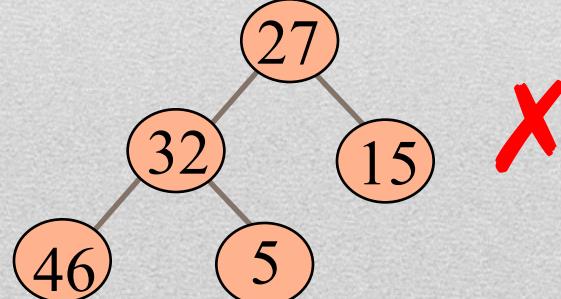
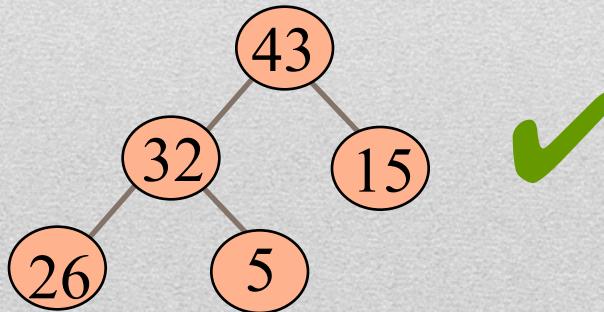
(mejora de los Árboles Binarios de Búsqueda)

Ing. Luis Humberto González

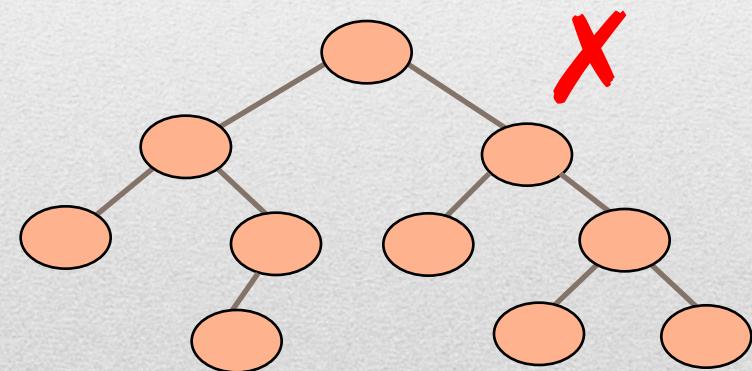
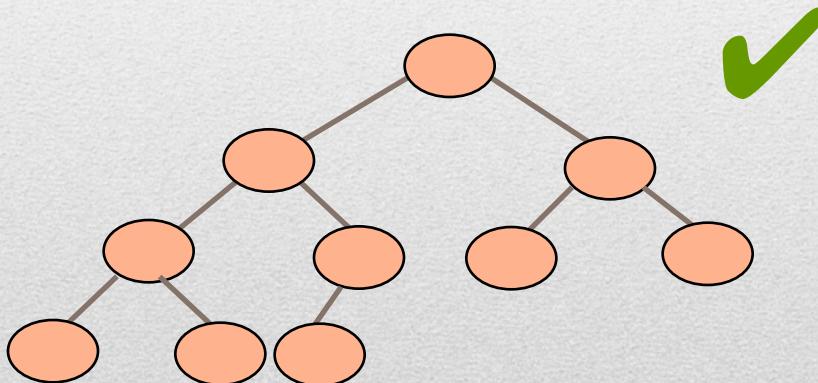
¿Qué es un HEAP?

En el nivel lógico o abstracto, un HEAP es un árbol binario que cumple con las siguientes reglas:

1. Todo nodo padre del árbol contiene un elemento que tiene un valor de mayor prioridad que los valores de sus nodos hijos.
La prioridad está determinada por la aplicación...
Ej. a mayor valor, mayor prioridad.



2. El árbol está completamente balanceado, es decir, todos los niveles del árbol tienen el máximo de nodos posible, excepto el nivel inferior que puede estar incompleto.
3. Los nodos hojas del nivel inferior están lo más a la izquierda posible.



c¿Qué es un Heap continúa...

✓ Implementación de COLAS PRIORIZADAS:

El nivel físico del ADT Cola Priorizada es un Heap.

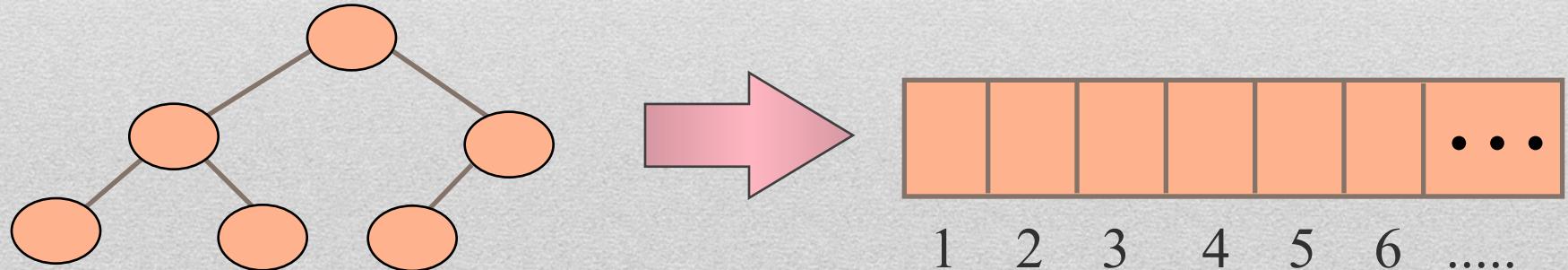
Por lo tanto, muchas de las aplicaciones de Grafos, se pueden implementar en forma óptima utilizando un HEAP.

✓ Implementación del método HEAP SORT para el ordenamiento de información.

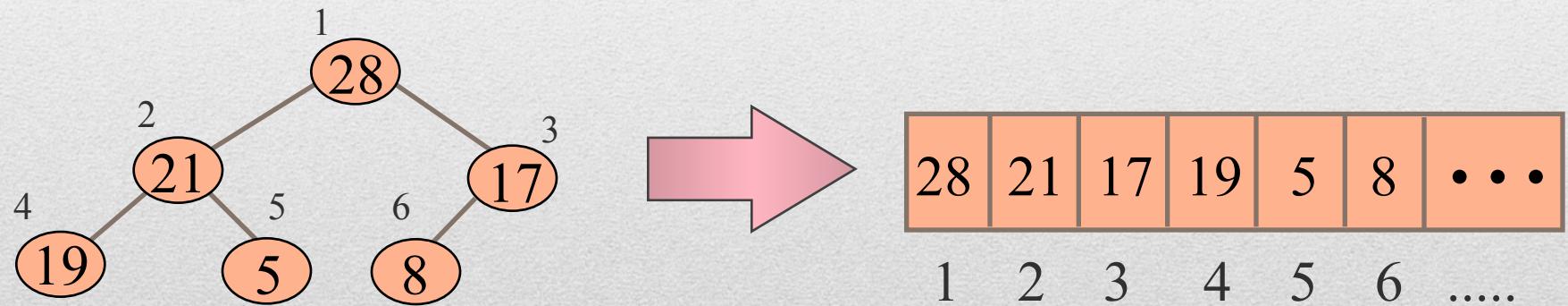
Aplicaciones de un HEAP

Nivel físico de un HEAP...

- ✓ Aunque lógicamente un HEAP es un árbol binario, dadas su características y las aplicaciones en que se ocupa...
- ✓ la representación de un HEAP puede realizarse eficientemente en un ARREGLO unidimensional...

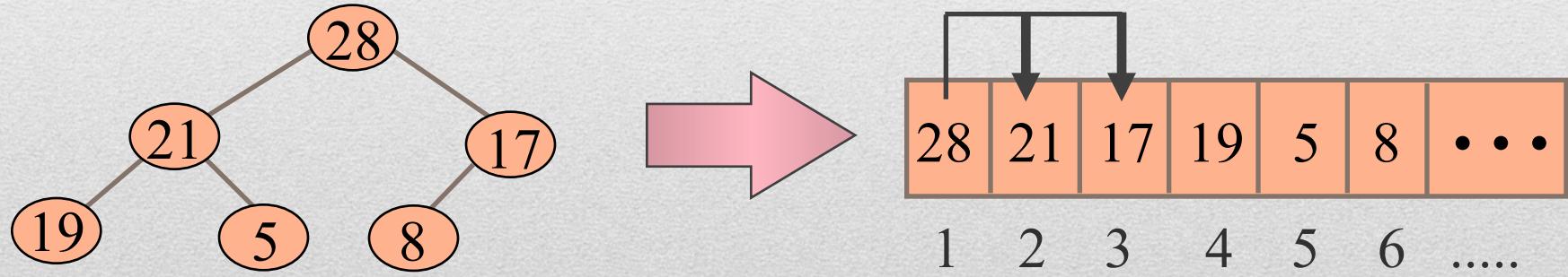


- ✓ Dado que un HEAP es un árbol binario completo...
- ✓ En el arreglo, el elemento de la posición ‘k’ representa a un nodo del árbol cuyos nodos hijos estarán en las posiciones $2k$ y $2k+1$ del arreglo...



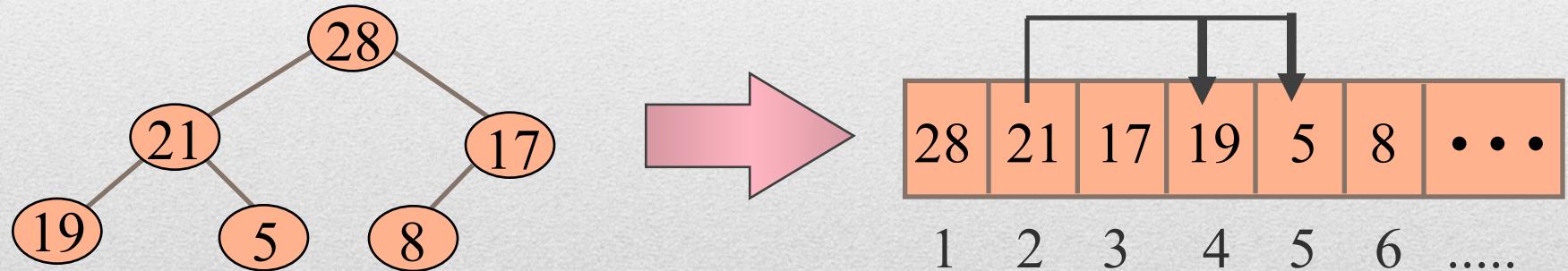
Nivel físico de un HEAP...

- ✓ Dado que un HEAP es un árbol binario completo...
- ✓ En el arreglo, el elemento de la posición ‘k’ representa a un nodo del árbol cuyos nodos hijos estarán en las posiciones $2k$ y $2k+1$ del arreglo...



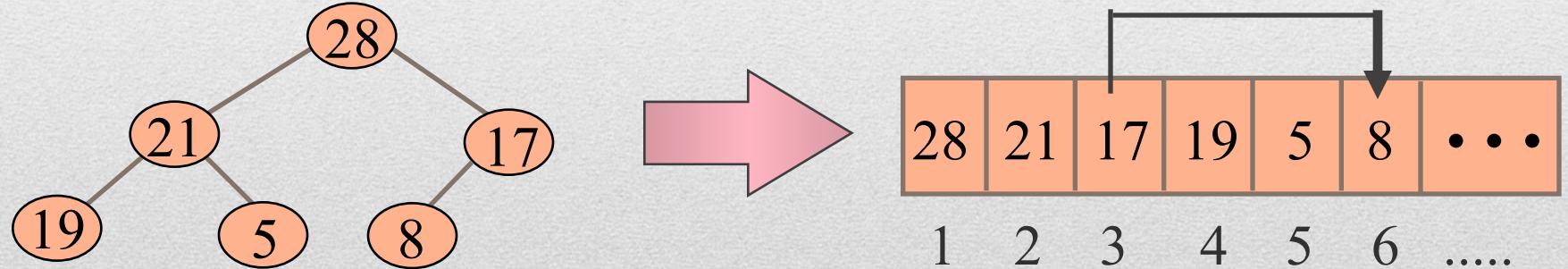
Nivel físico de un HEAP...

- ✓ Dado que un HEAP es un árbol binario completo...
- ✓ En el arreglo, el elemento de la posición ‘k’ representa a un nodo del árbol cuyos nodos hijos estarán en las posiciones $2k$ y $2k+1$ del arreglo...



Nivel físico de un HEAP...

- ✓ Dado que un HEAP es un árbol binario completo...
- ✓ En el arreglo, el elemento de la posición ‘k’ representa a un nodo del árbol cuyos nodos hijos estarán en las posiciones $2k$ y $2k+1$ del arreglo...



Nivel físico de un HEAP...

- ✓ Toda lista ordenada es un HEAP.
- ✓ Toda lista puede convertirse en un HEAP y de ahí obtener las aplicaciones correspondientes.
- ✓ ¿Cuáles de las siguientes listas son un Heap?

8, 3, 2, 1, 2, 1

1, 2, 3, 4, 5, 6, 7, 8

5, 8, 9, 7, 5, 4

3, 4, 3, 4, 4, 3, 5

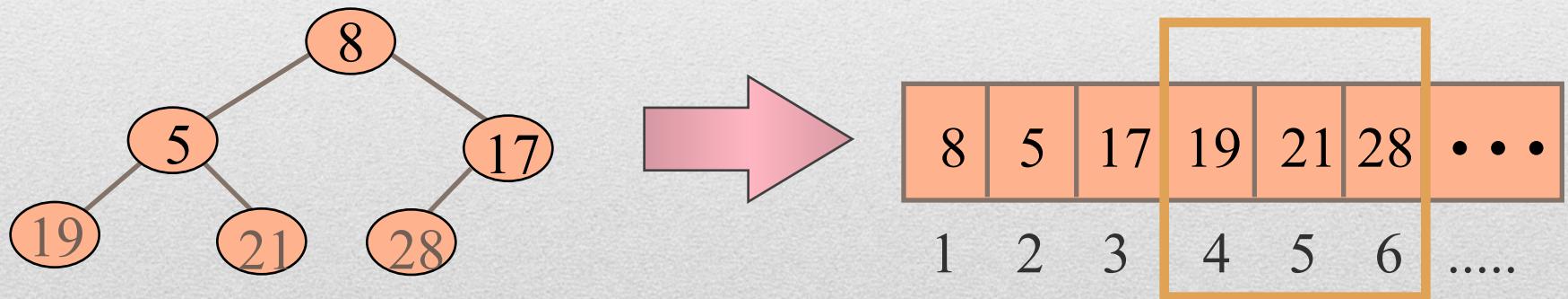
Heap vs. Lista...

- ✓ Toda lista ordenada es un HEAP.
- ✓ Toda lista puede convertirse en un HEAP y de ahí obtener las aplicaciones correspondientes.
- ✓ ¿Cuáles de las siguientes listas son un Heap?

8, 3, 2, 1, 2, 1	✓
1, 2, 3, 4, 5, 6, 7, 8	✓
5, 8, 9, 7, 5, 4	✗
3, 4, 3, 4, 4, 3, 5	✓

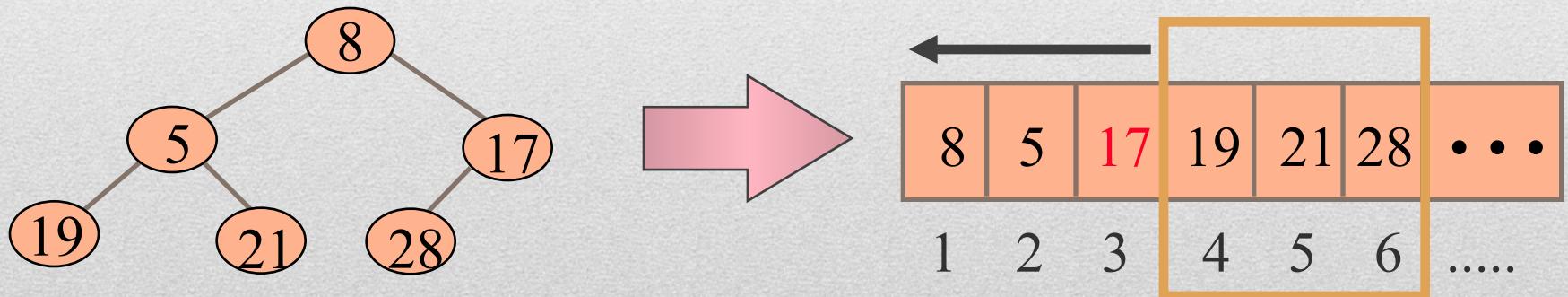
Heap vs. Lista...

- ✓ Inicialmente se puede considerar que todos los nodos hoja del árbol HEAP cumplen con la regla del Heap por default.
- ✓ ¿Dónde se localizan los nodos hojas de un árbol Heap representado en un arreglo?



¿Cómo convertir una lista en Heap?

- Continuar analizando los nodos padre, del nivel inferior hacia arriba , y de derecha a izquierda, reacomodando hacia abajo los valores de los nodos descendientes para cumplir la regla del Heap.
- Tomar la primera mitad del arreglo del subíndice más grande hacia el más pequeño...



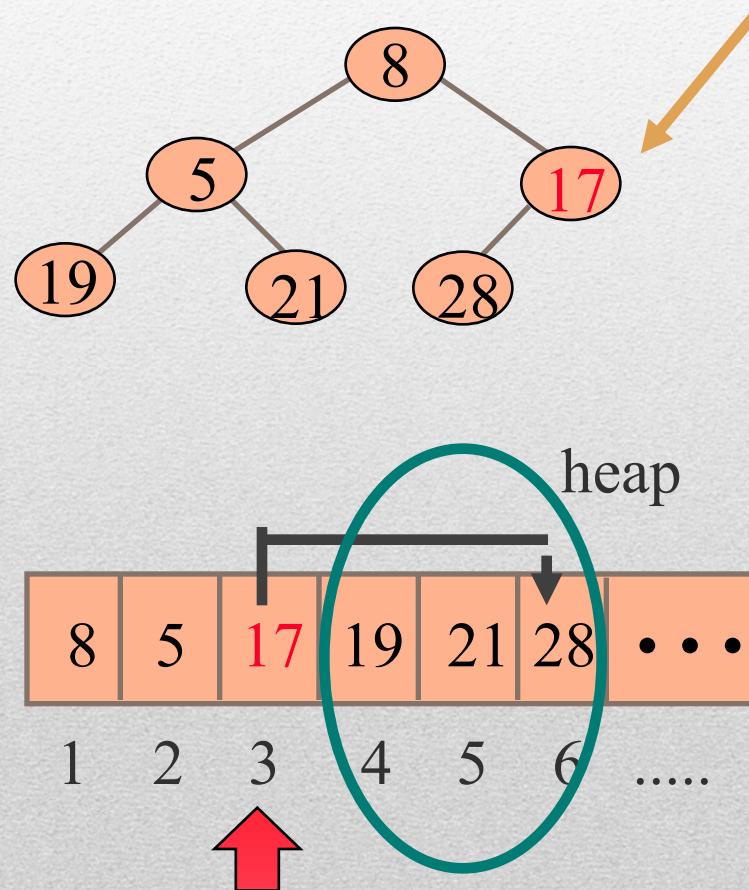
¿Cómo convertir una lista en Heap?



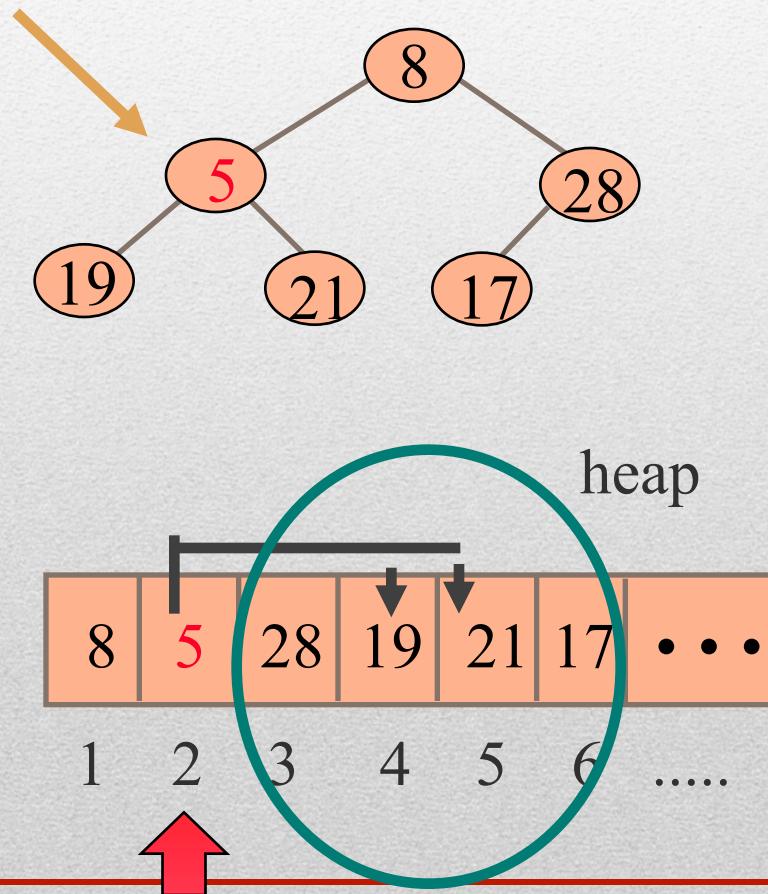
Reacomodar hacia abajo...

- ✓ Comparar contra los valores de los nodos hijo...
 - ✓ Si el valor del padre es de mayor prioridad, el proceso de reacomodo de ese elemento termina.
 - ✓ Si el valor de algún hijo es de mayor prioridad , intercambiarlo con el padre, y repetir el proceso de reacomodo hacia abajo con los hijos del elemento intercambiado.
-

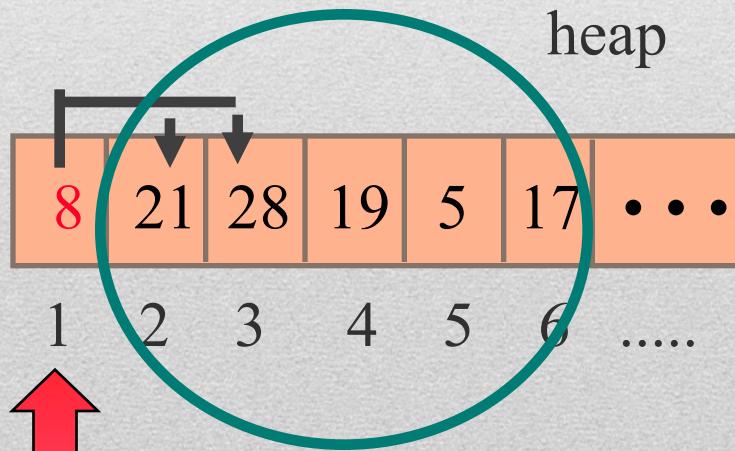
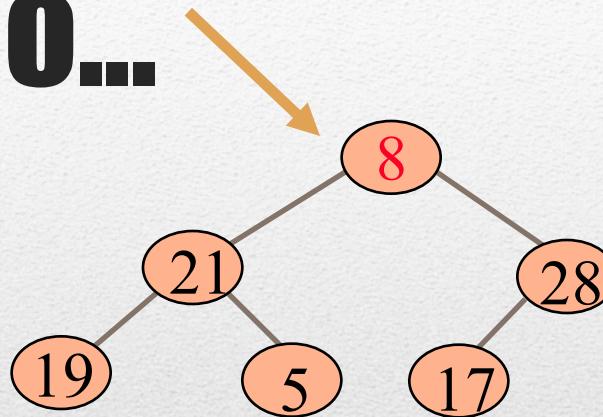
Ejemplo...



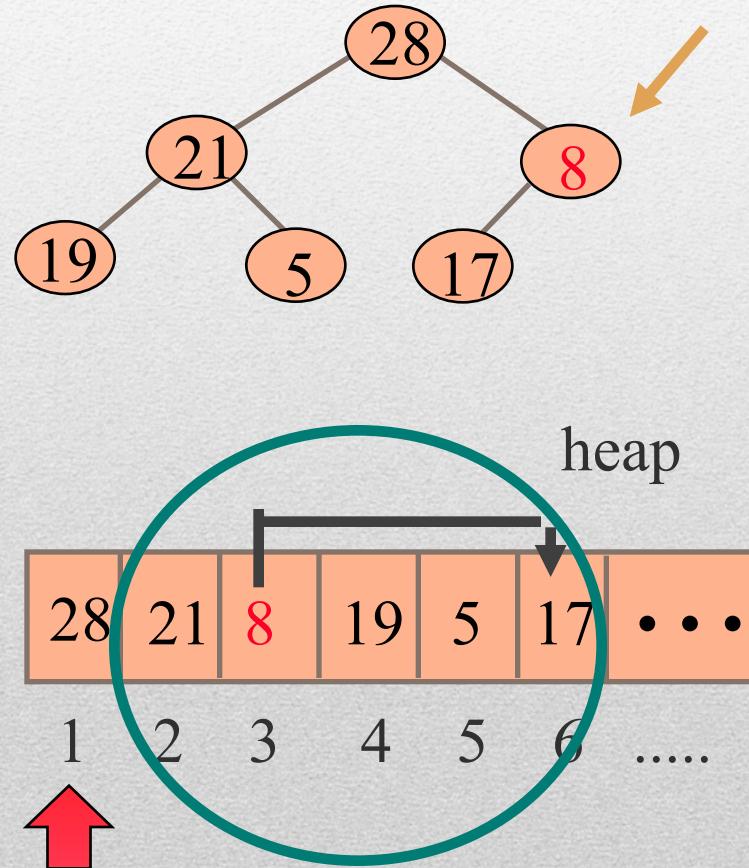
Ejemplo...



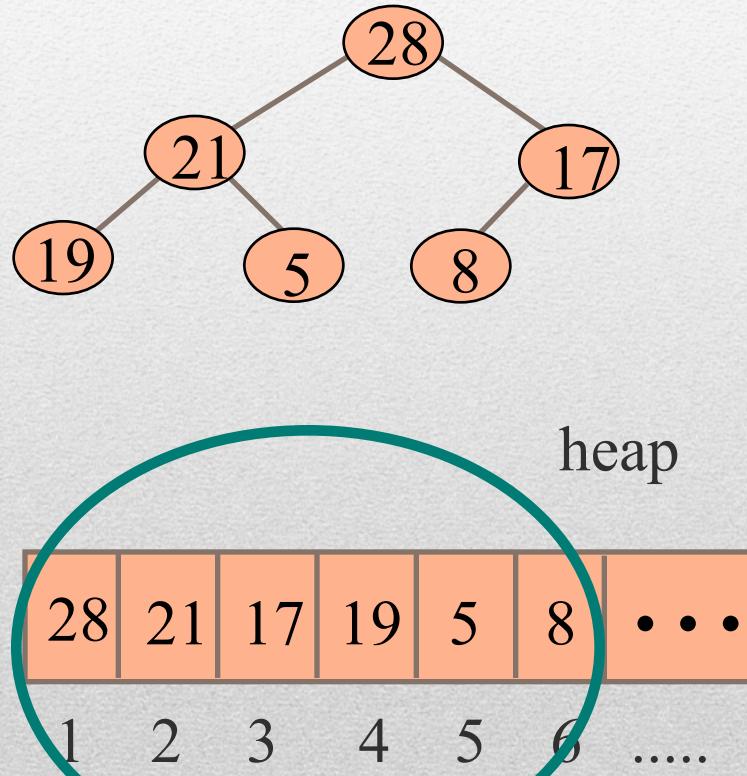
Ejemplo...



Ejemplo...



Ejemplo...



Sea AP el apuntador al elemento a acomodar:

1. $AUX = AP$
2. $HIJOS = 2 * AP$ //HIJOS apunta al hijo izquierdo de nodo apuntado por AP
3. *Mientras haya hijos de AUX (HIJOS <= n) y alguno de ellos sea mayor:*

Encontrar el hijo mayor de AUX y apuntarlo por HIJOMAY.

Si LISTA[HIJOMAY] > LISTA[AUX] entonces

intercambiar LISTA[AUX] y LISTA[HIJOMAY]

AUX = HIJOMAY

*HIJOS = 2 * AUX*

si no salir del ciclo.

O(log₂ n)

Algoritmo para reacomodar un elemento hacia abajo

Sea n la cantidad de elementos en un arreglo llamado LISTA:

1. APUNTADOR = $n \text{ div } 2$

//se analizan sólo nodos que no son hojas

2. Mientras APUNTADOR ≥ 1 :

*Reacomodar dato apuntado por APUNTADOR llamando a la rutina
ACOMODA_ABAJO*

APUNTADOR = APUNTADOR - 1

O($n \log_2 n$)

Algoritmo para crear un heap en una lista

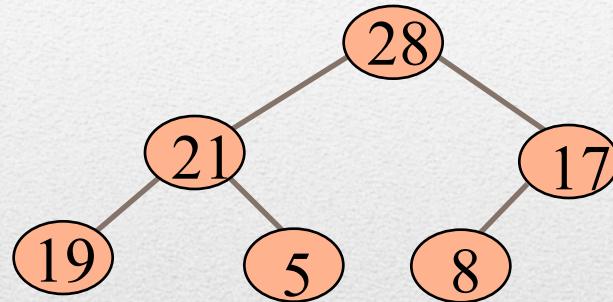
- ✓ Las aplicaciones de un HEAP (Colas Priorizadas, HeapSort) generalmente requieren dar de baja al nodo de mayor prioridad en el Heap.
- ✓ Esto quiere decir, sólo necesitan dar de baja a la raíz del Heap (el primer elemento del arreglo).

Eliminación en un HEAP

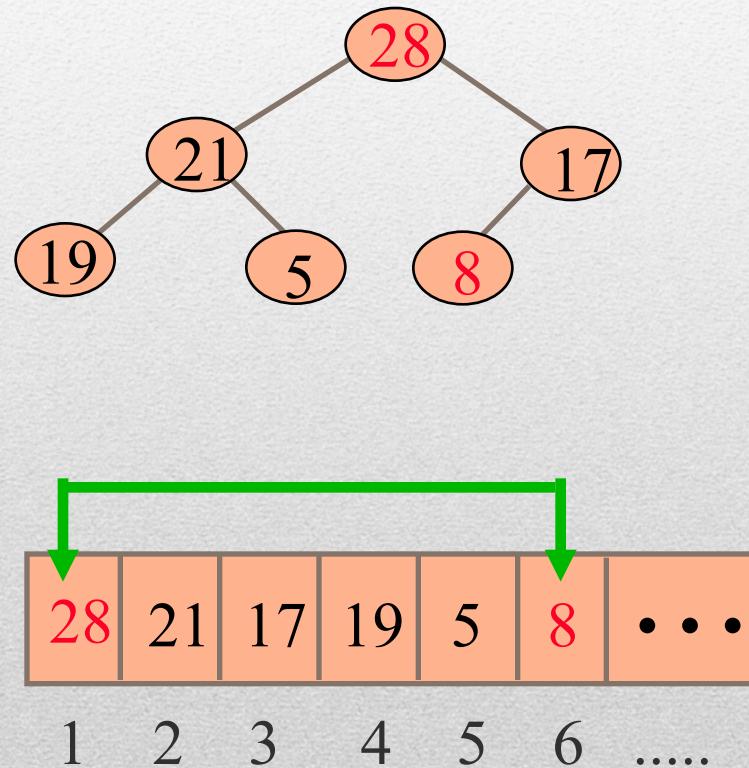
¿Cómo eliminar la raíz de un HEAP?

- ✓ Se toma el valor de la última hoja del Heap, es decir, el último elemento de la lista.
 - ✓ Se intercambia este valor con el que está en la raíz.
 - ✓ Físicamente se elimina al nodo que representa a la última hoja.
 - ✓ Y por último, se reacomoda hacia abajo al elemento que quedó como raíz del Heap. Esto equivale a ejecutar el último paso de la construcción del Heap.
-

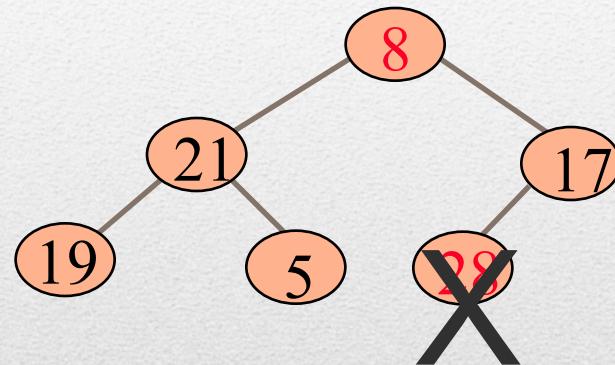
Ejemplo, eliminar a la Raíz...



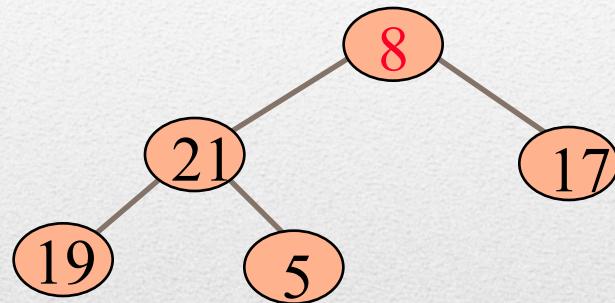
Ejemplo, eliminar a la Raíz...



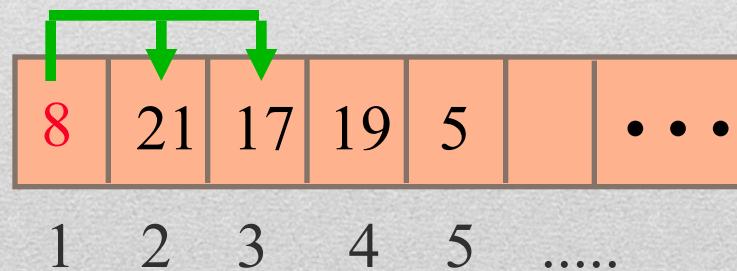
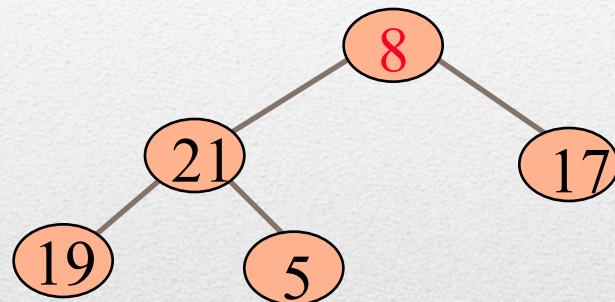
Ejemplo, eliminar a la Raíz...



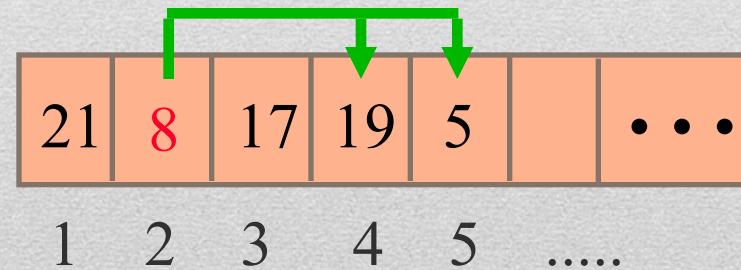
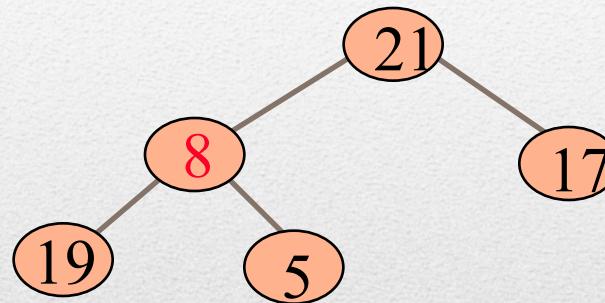
Ejemplo, eliminar a la Raíz...



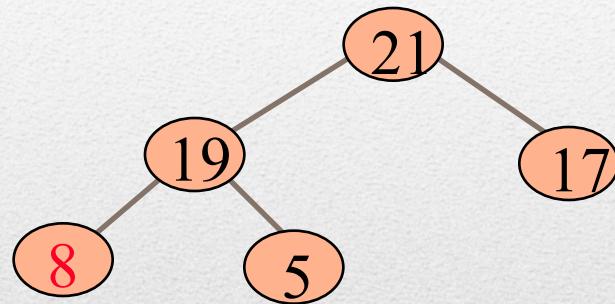
Ejemplo, eliminar a la Raíz...



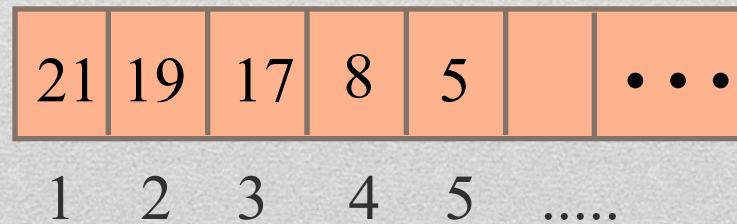
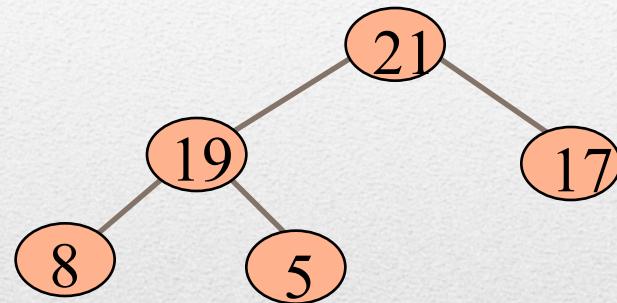
Ejemplo, eliminar a la Raíz...



Ejemplo, eliminar a la Raíz...



Ejemplo, eliminar a la Raíz...



Sea n la cantidad de elementos en un arreglo llamado LISTA:

1. *LISTA[1] es el elemento que sale.*
2. *Intercambiar LISTA[1] con LISTA[n].*
3. $n = n - 1$
4. *Reacomodar dato de la posición 1
llamando a la rutina
ACOMODA_ABAJO.*

O($\log_2 n$)

Algoritmo para eliminar dato de un HEAP

- ✓ Las aplicaciones de un HEAP (Colas Priorizadas, HeapSort) generalmente requieren ir agregando nuevos elementos en el Heap.
- ✓ Estos elementos serán insertados como una nueva hoja en el Heap, es decir será el último elemento en la lista que representa al Heap.

Inserción en un HEAP

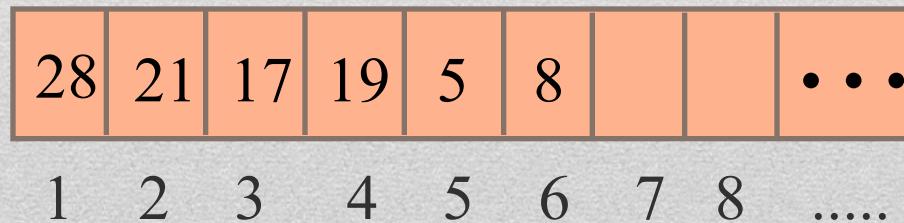
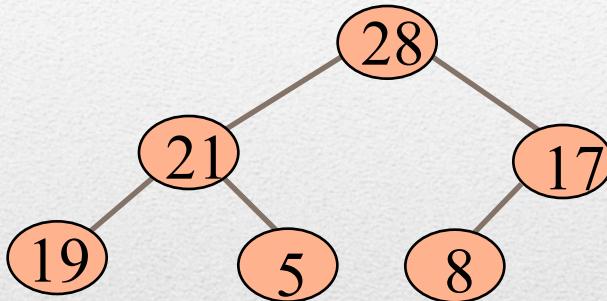
¿Cómo agregar un nuevo elemento al HEAP?

- ✓ Se inserta el nuevo elemento como la última hoja del Heap, es decir, como último elemento de la lista.
 - ✓ Sin embargo, puesto que se requiere que no se pierda el “orden” existente en el Heap...
 - ✓ Se tiene que reacomodar al nuevo elemento con respecto al valor de sus ancestros...
 - ✓ Esto quiere decir, reacomodar el elemento hacia arriba.
-

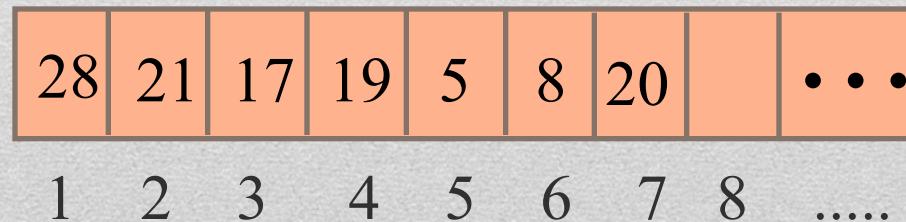
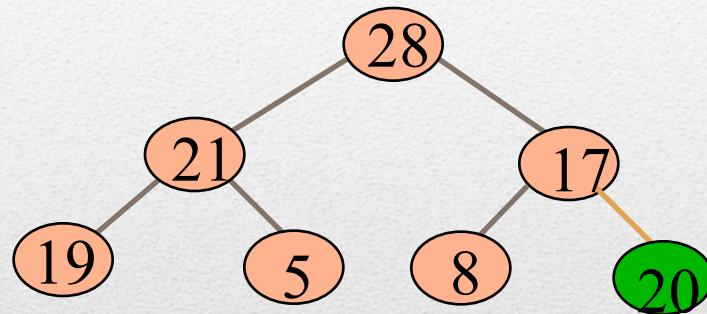
Reacomodar hacia arriba...

- ✓ Comparar el nuevo elemento contra su padre...
 - *¿En dónde se encuentra su padre?....*
 - *Para el elemento de la posición 'k', en la posición $k / 2$*
 - ✓ Si el valor del padre es de menor prioridad
 - Intercambia los valores y sigue comparando hacia arriba.
 - ✓ Si el valor de su padre es de mayor prioridad
 - el proceso de reacomodo termina.
-

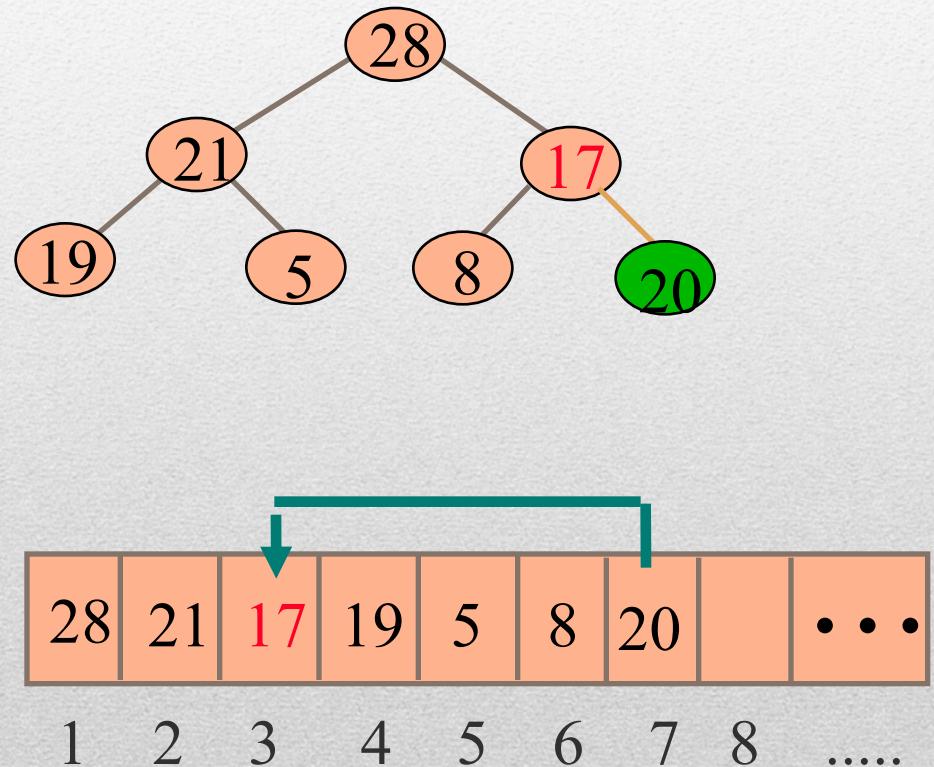
Ejemplo, insertar 20...



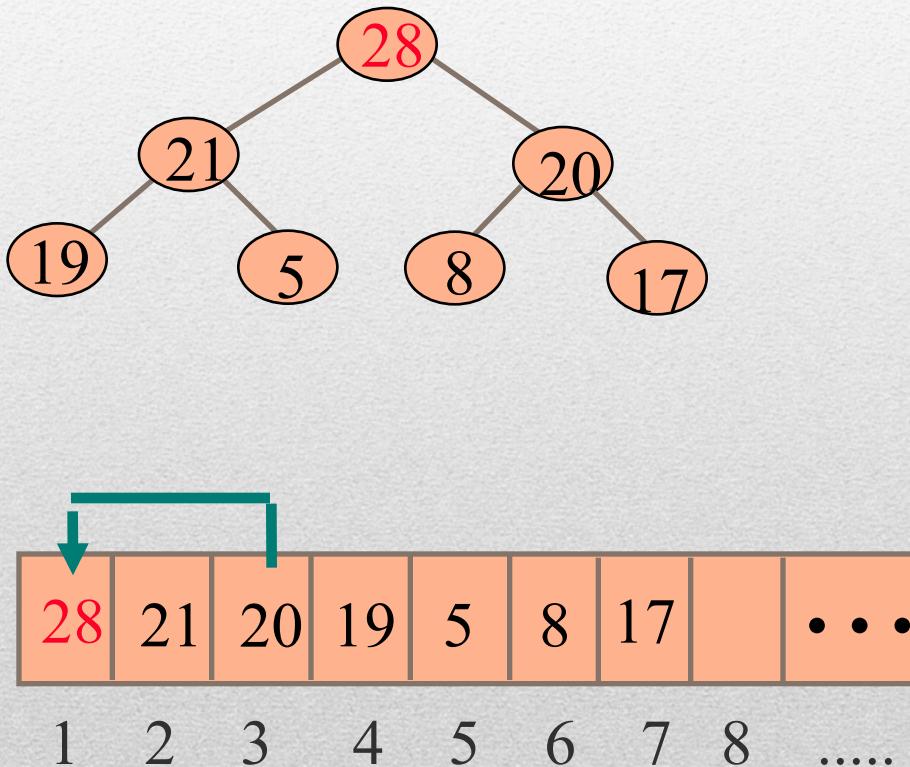
Ejemplo, insertar 20...



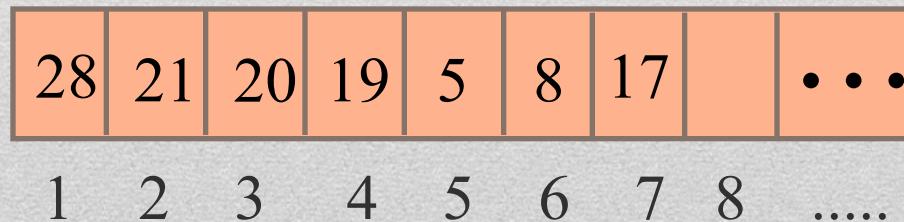
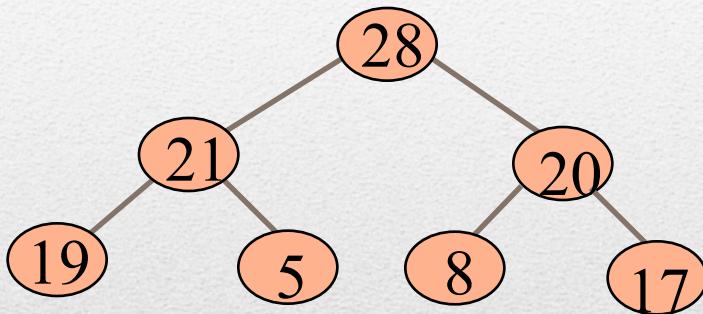
Ejemplo, insertar 20...



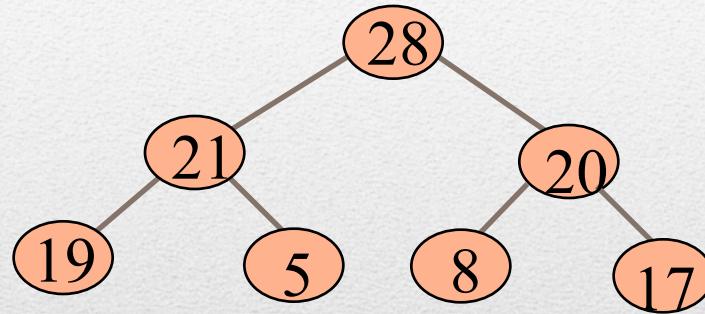
Ejemplo, insertar 20...



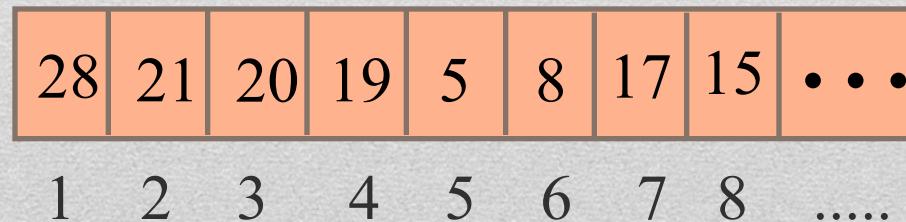
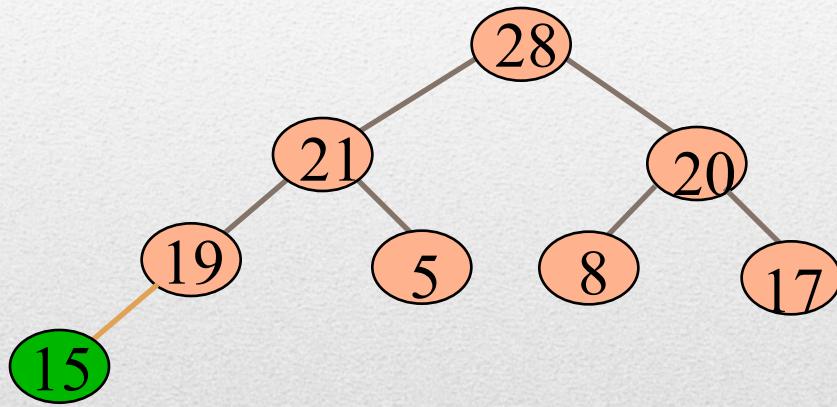
Ejemplo, insertar 20...



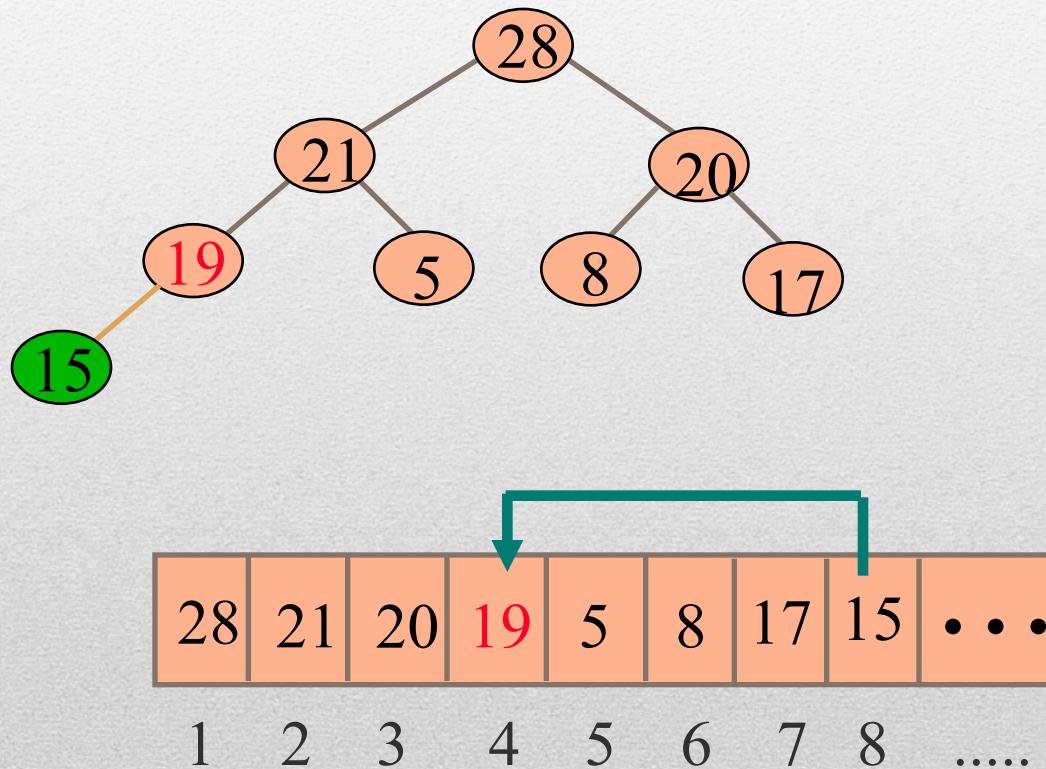
Ejemplo, insertar 15...



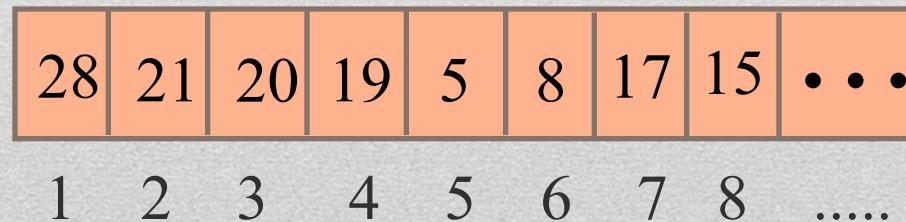
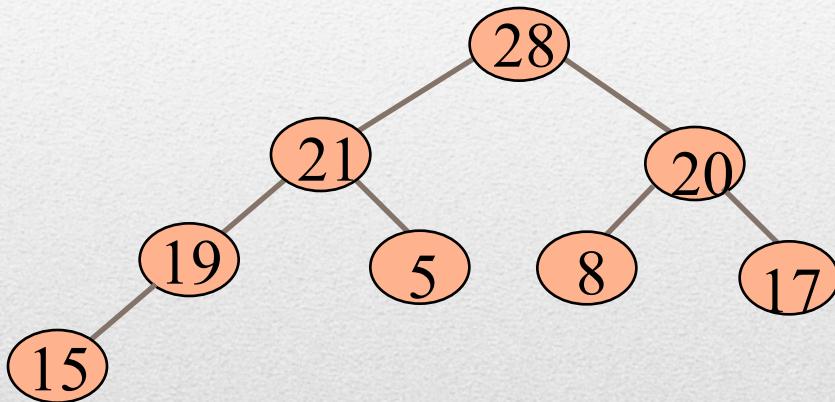
Ejemplo, insertar 15...



Ejemplo, insertar 15...



Ejemplo, insertar 15...



Sea AP el apuntador al elemento a acomodar:

Mientras ($AP > 1$ y

LISTA[$AP \text{ div } 2$] > LISTA[AP]):

Intercambiar LISTA[$AP \text{ div } 2$] con LISTA[AP];

$AP = AP \text{ DIV } 2;$

O($\log_2 n$)

Algoritmo para reacomodar hacia arriba

Sea v el valor que se desea insertar en el HEAP, y n la cantidad de elementos en un arreglo llamado LISTA:

- 1. Si existe espacio en el arreglo entonces $LISTA[n+1] = v$; sino, salir.*
- 2. Reacomodar dato de la posición $n+1$ llamando a la rutina ACOMODA_ARRIBA*

O($\log_2 n$)

Algoritmo para la inserción en un Heap

HeapSort

- Algoritmo competitivo para ordenar un conjunto de datos.
- La prioridad de los elementos a ordenar se establece de acuerdo al tipo de ordenamiento a realizar (ascendente o descendente).
 - ORDEN ASCENDENTE: Mayor valor, mayor prioridad.
 - ORDEN DESCENDENTE: Menor valor, mayor prioridad.



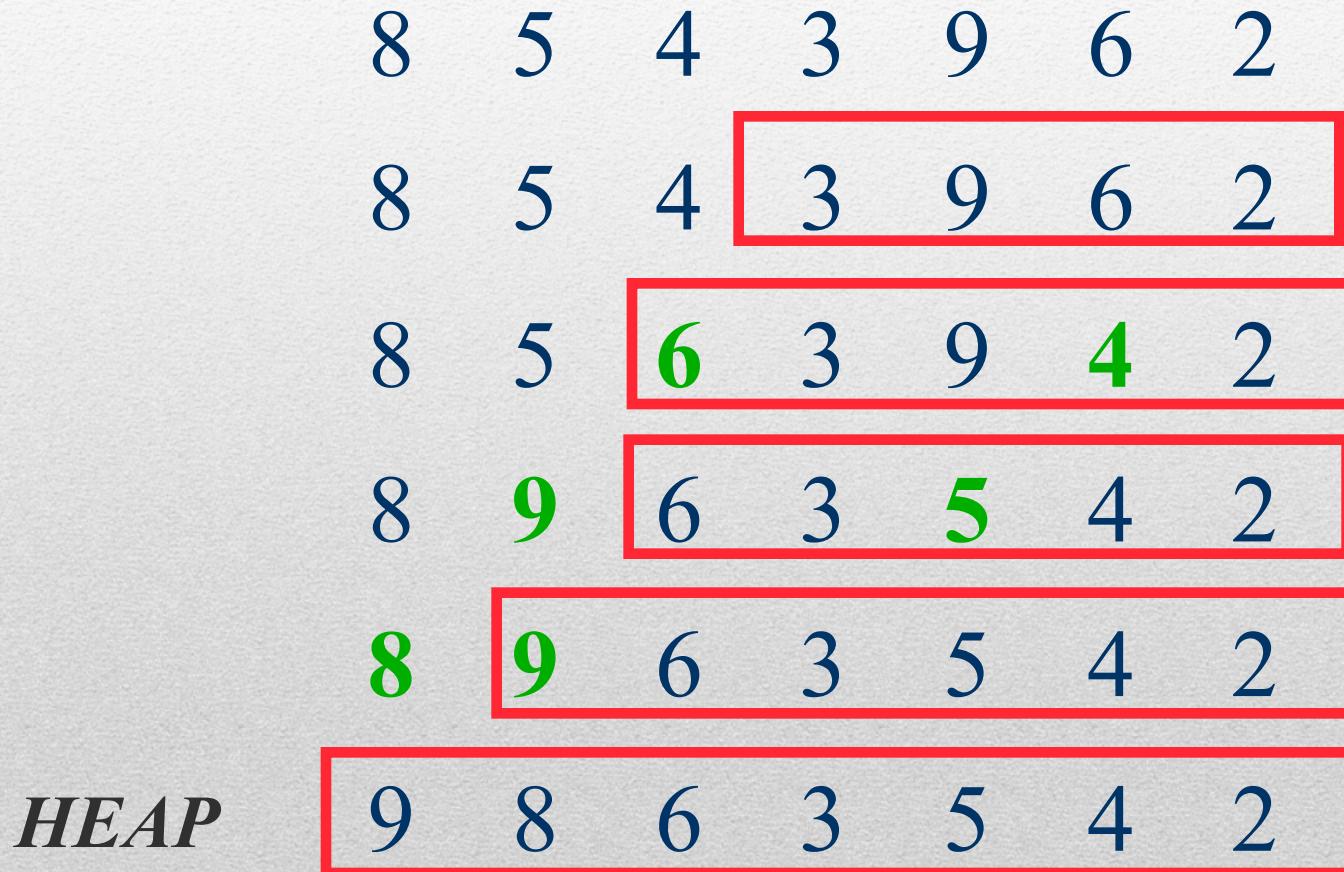
1. Convertir la lista de elementos en un Heap (de acuerdo a la prioridad).
2. Para cada elemento del Heap (iniciando en la Raíz) hacer:
 - ✓ Intercambiar la Raíz con la última hoja. Con esto queda ordenado pues es el de mayor prioridad.
 - ✓ Disminuir el tamaño del Heap (ignorar el ya acomodado).
 - ✓ Reestablecer el Heap, acomodando hacia abajo al elemento que quedó temporalmente como Raíz.

Esto se repite hasta que el heap tenga 1 elemento.

Algoritmo general del HeapSort

Ejemplo: Ordenar Ascendentemente

- ✓ Primero: Construir un Heap.



Ejemplo: Ordenar Ascendentemente

- ✓ Segundo: Eliminar consecutivamente la Raíz del Heap.

	9	8	6	3	5	4	2
	2	8	6	3	5	4	9
HEAP	8	5	6	3	2	4	9
	4	5	6	3	2	8	9
HEAP	6	5	4	3	2	8	9
	2	5	4	3	6	8	9

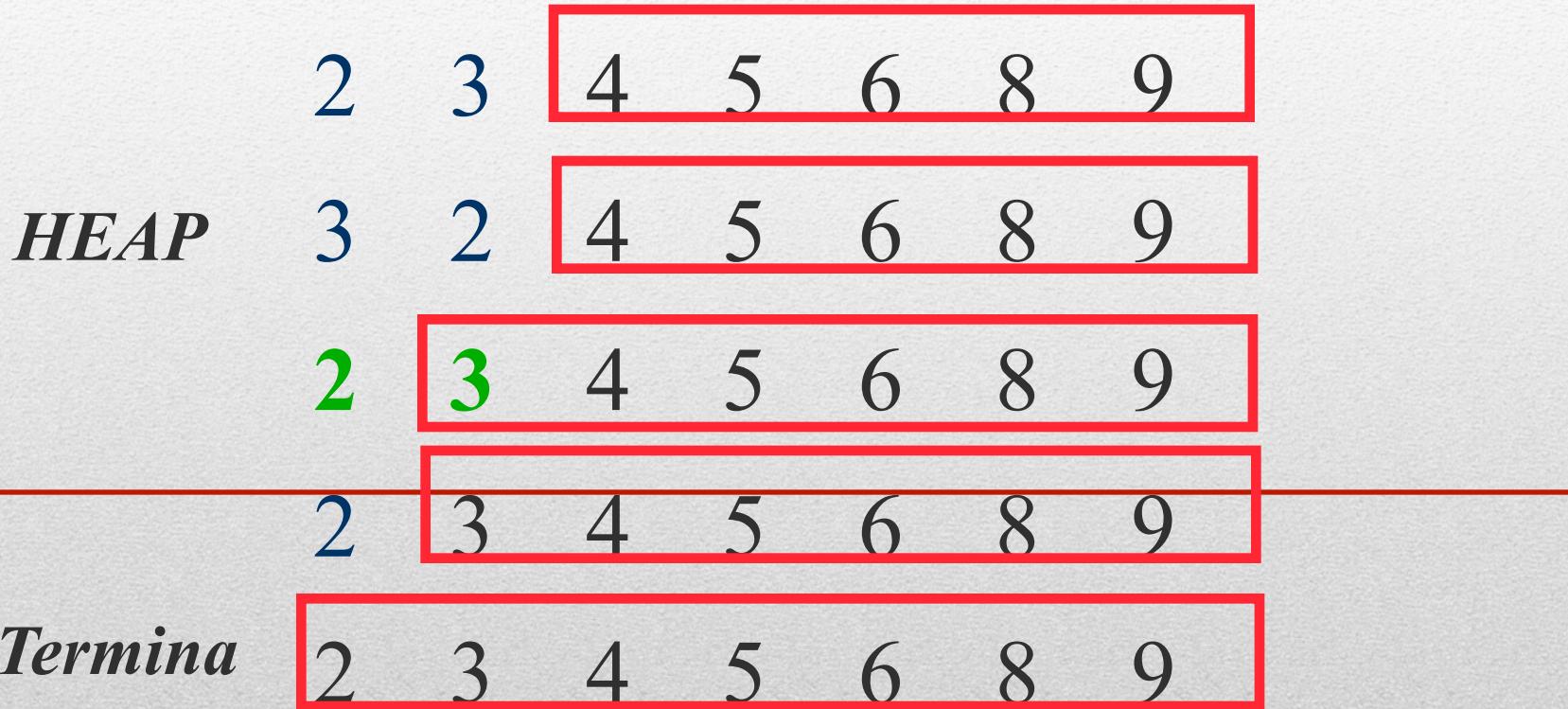
Ejemplo: Ordenar Ascendentemente

Continuación.....

Continuacion.....								
	2	5	4	3	6	8	9	
<i>HEAP</i>	5	3	4	2	6	8	9	
	2	3	4	5	6	8	9	
<i>HEAP</i>	4	3	2	5	6	8	9	
	2	3	4	5	6	8	9	
<i>HEAP</i>	3	2	4	5	6	8	9	

Ejemplo: Ordenar Ascendentemente

Continuación.....



- Por intercambio $O(n^2)$
- Por inserción $O(n^2)$
- Por selección $O(n^2)$
- Merge Sort $O(n \log n)$
- Quick Sort $O(n \log n)$ *su peor caso tiende a ser n^2
- Heap Sort $O(n \log n)$

Comparación de Sorts
