

DecreaseKey (x, y, H)

decrementar elem en pos x de heap H en una cant y

IncreaseKey (x, y, H)

incrementar la clave.

DeleteKey (x)

eliminar elem en pos x de H

de H

Insertar en un árbol B de orden k (log en el árbol) los datos recorren desde

haga buena raíz

Entonces, para crear un árbol perfecto insertar de a uno $(n \log n)$ operaciones Total

BuildHeap Filtra hacia abajo cada elem.

Se elige $<$ de los hijos, se compara el menor con el padre

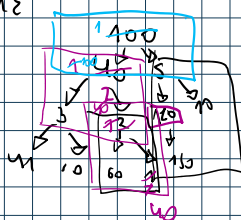
Se empieza filtrando desde lo q' está en raíz

Se filtra nodes con hijos, el resto son hojas

100, 40, 5, 32, 120, 10, 41, 50, 60, 1, 110

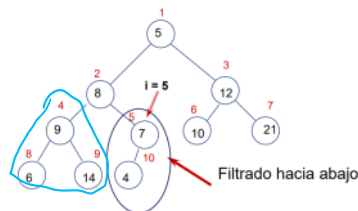
heap \rightarrow array

array \rightarrow 100, 40, 5, 32, 120, 10, 41, 50, 60, 1, 110



un gollomb pero es fácil.

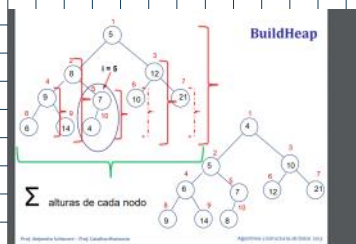
5	8	12	9	7	10	21	6	14	4
1	2	3	4	5	6	7	8	9	10



en Filtrado de cada node recorremos

su array.

Para cada elemento se genera un array
array de todos los nodes



en cada rango último hijo

se fija el rango q' filtra, elijo al $<$ de los hijos

voy al elem $i-1$ en el array, repito

El Filtrado hacia abajo para cuando llego a una hoja.

en árbol binario lleno de altura h con $2^{h+1} - 1$ nodes, suma $\sum_{i=1}^{h+1} (2^i - 1) = (h+1)2^h$

El árbol binario completo tiene entre 2^h y $2^{h+1} - 1$ nodos, el teorema implica que esta suma es de $O(n)$ donde n es el número de nodos.

$O(n)$

$2^h \leq n < 2^{h+1}$

$h = \lceil \log_2(n) \rceil$

Con el árbol, de un árbol lleno

El árbol binario completo tiene entre 2^h y $2^{h+1} - 1$ nodos, el teorema implica que esta suma es de $O(n)$ donde n es el número de nodos.

El árbol binario completo tiene entre 2^h y $2^{h+1} - 1$ nodos, el teorema implica que esta suma es de $O(n)$ donde n es el número de nodos.

Oficinas vector usando heap

a) Algoritmo que usa una heap y requiere una cantidad aproximada de $(n \log n)$ operaciones.

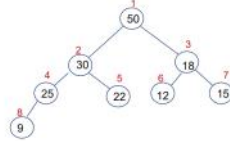
➤ Construir una MinHeap, realizar n DeleteMin operaciones e ir guardando los elementos extraídos en otro arreglo.

➤ Desventaja: requiere el doble de espacio

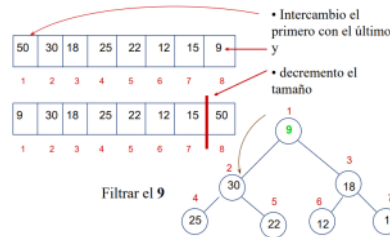
b) Algoritmo **HeapSort** que requiere una cantidad aproximada de $(n \log n)$ operaciones, pero **menos espacio**.

➤ Construir una MaxHeap con los elementos que se desean ordenar, intercambiar el último elemento con el primero, decrementar el tamaño de la heap y filtrar hacia abajo. Usa sólo el espacio de almacenamiento de la heap.

Ejemplo:



HeapSort (cont.)



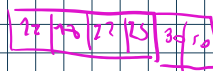
Para ordenar la forma crearse, max heap

1 2 3 4 5 6 7 8

decremento, min heap

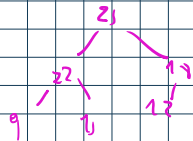
50 30 18 25 22 12

n veces borrar min



Con Pire algo, min heap para max heap mayor

HeapSort ➤ Crear max heap
decrec. - min



50 30 18 25 22 12 15 9
9 30 18 25 22 12 15 50
30 25 18 9 22 12 15 50
18 25 18 9 22 12 15 30 50
25 22 18 9 15 12 30 50

1 2 3 4 5 6
18 22 18 25 30 10
1 2 3
25 30 10

➔ 18 18 22 21 30 10

$$\begin{array}{cccccc|c} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 30 & 25 & 18 & 9 & 22 & 12 & 15 \end{array} \Bigg| 50$$

$$\begin{array}{cccccc|c} 1 & 2 & 3 & 4 & 5 & 6 \\ 25 & 22 & 18 & 9 & 11 & 18 \end{array} \Bigg| 30 \ 50$$

$$\begin{array}{cccccc|c} 18 & 22 & 18 & 9 & 11 & 25 & 30 \ 50 \end{array}$$

$$\begin{array}{cccccc|c} 12 & 15 & 18 & 9 & 12 & & \end{array}$$

$$\begin{array}{cccccc|c} 12 & 11 & 18 & 9 & 22 & 21 & 30 \ 10 \end{array}$$

$$\begin{array}{cccccc|c} 18 & 11 & 12 & 9 & & & \end{array}$$

$$\begin{array}{cccccc|c} 9 & 15 & 18 & 18 & 22 & 21 & 30 \ 50 \end{array}$$

$$\begin{array}{cccccc|c} 18 & 9 & 18 & & & & \end{array}$$

$$\begin{array}{cccccc|c} 12 & 9 & 11 & 18 & 22 & 21 & 30 \ 50 \end{array}$$

$$\begin{array}{cccccc|c} 9 & 15 & 11 & 18 & 22 & 21 & 30 \ 50 \end{array}$$

U