Análisis y Diseño de Algoritmos

Ordenamiento – Heapsort y Quicksort

DR. JOSÉ MARTÍNEZ CARRANZA

CIENCIAS COMPUTACIONALES
INAOE

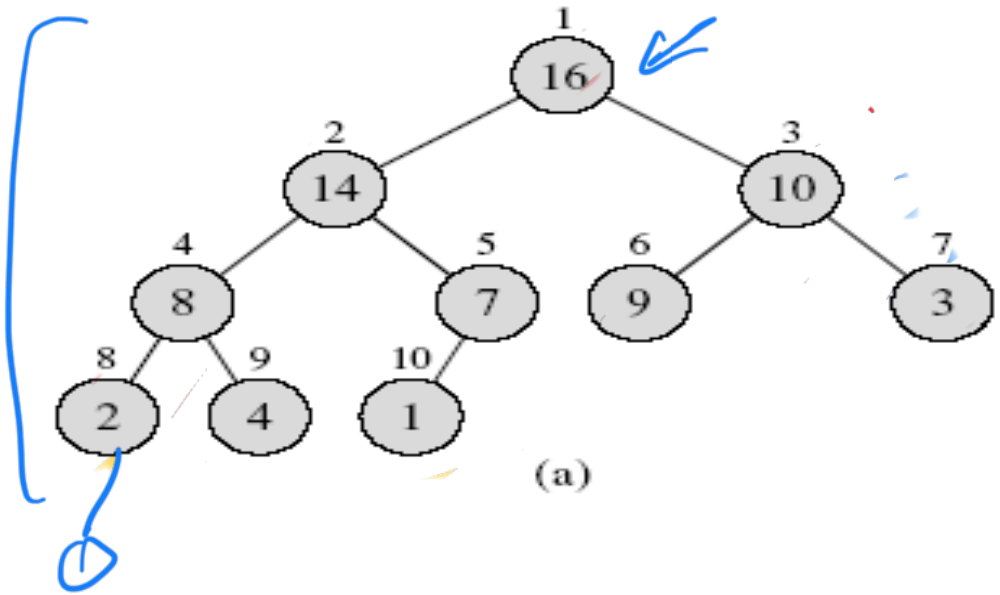
Notas sobre derechos de autor

- Varias de las imágenes o videos utilizados en las presentaciones de este curso son tomadas de sitios públicos en Internet. Los derechos, si existiesen, permanecen con su autor o autores.

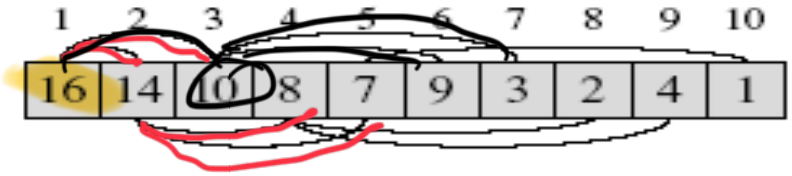
- El uso de éstas imagenes o videos es meramente para fines ilustrativos sin propósito de lucro, publicidad o comercialización.

Heaps

- Un Heap es una estructura de datos binaria
  - Un arreglo que representa un árbol binario completo



(a)



(b)

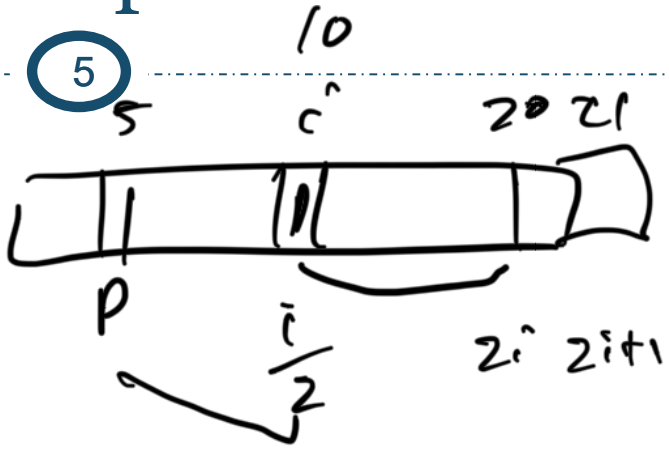
Heaps

- Cada nodo es un elemento del arreglo
  - Almacena el valor en el nodo
  - Árbol completamente lleno, excepto en el último nivel (tal vez)
- Arreglo representando el heap
  - 2 atributos

    Length[A] # elementos en el arreglo A

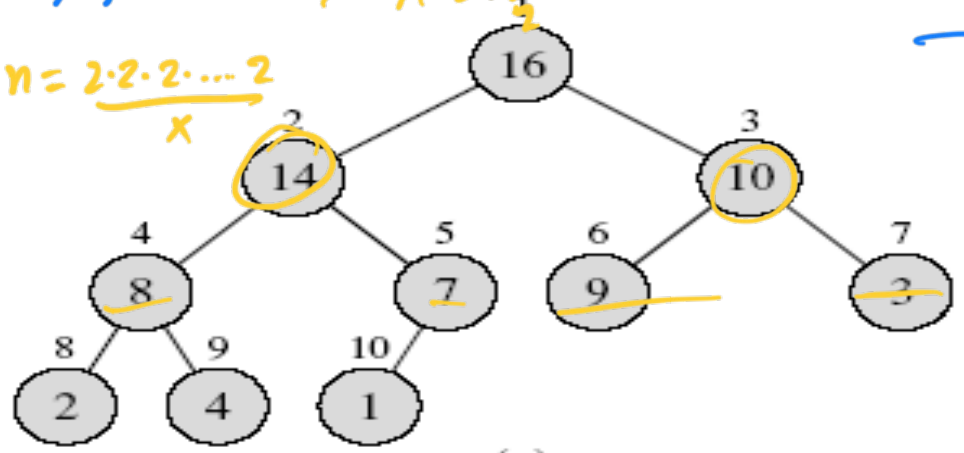
    Heap-size[A] # elementos en el heap almacenados en arreglo A
  - Heap-size [A] ≤ length[A]
  - Raíz del árbol A[1]

Heaps

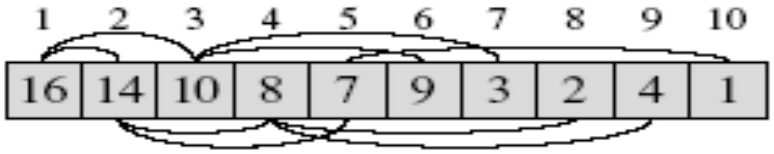


- Dado el índice *i* de un nodo
  - Parent(i) → return floor(i/2)
  - Left(i) → return 2i
  - Right(i) → return (2i + 1)
- Propiedad Heap
  - MAX-HEAP: A[PARENT(i)] ≥ A[i]
  - MIN-HEAP: A[PARENT(i)] ≤ A[i]
- Altura de un nodo
  - # arcos de la ruta simple más larga del nodo a una hoja
- Altura del árbol
  - Altura de la raíz
  - Altura del árbol es $\Theta(\lg n)$

Heaps



(a)

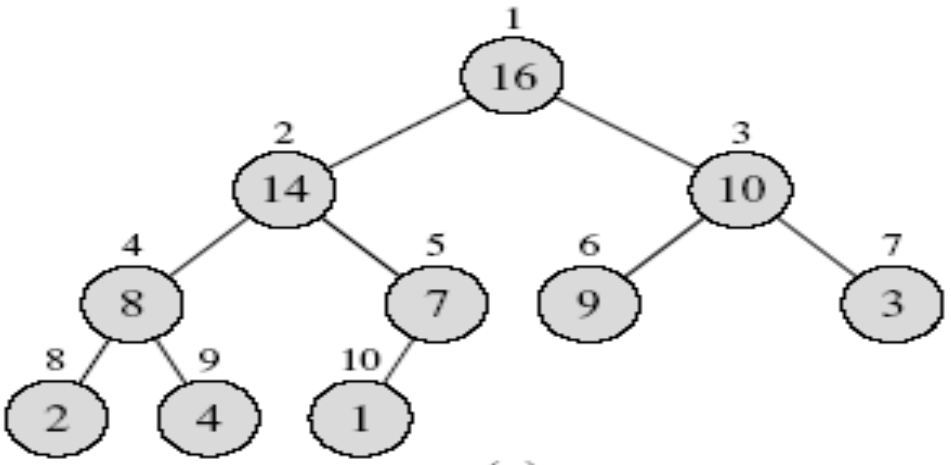


(b)

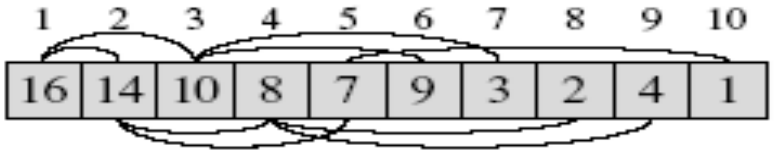
Heaps

- Manipulación de heaps
  - Entrada
    - Arreglo A y un índice i
    - Asume que LEFT(i) y RIGHT(i) son heaps
      - A[i] puede ser menor que sus hijos?
        - Viola la propiedad de un heap (MAX-HEAP)
  - Valor Máximo (MAX-HEAP)
    - Nodo raíz, O(1)
  - Inserción (MAX-HEAP)
    - Respetar la forma del heap
    - Insertar al final del arreglo (heap)
    - Verificar si mantiene la estructura del heap
    - ¿Qué pasa si no?

Heaps



(a)



(b)

Heaps

- Valor Máximo (MAX-HEAP)
  - Nodo raíz, O(1)
- Inserción (MAX-HEAP)
  - Respetar la forma del heap
  - Insertar al final del arreglo (heap)
  - Verificar si mantiene la estructura del heap: O(1)
  - ¿Qué pasa si no?
    - Se compara con el padre, y así hasta la raíz.
    - ¿Complejidad?

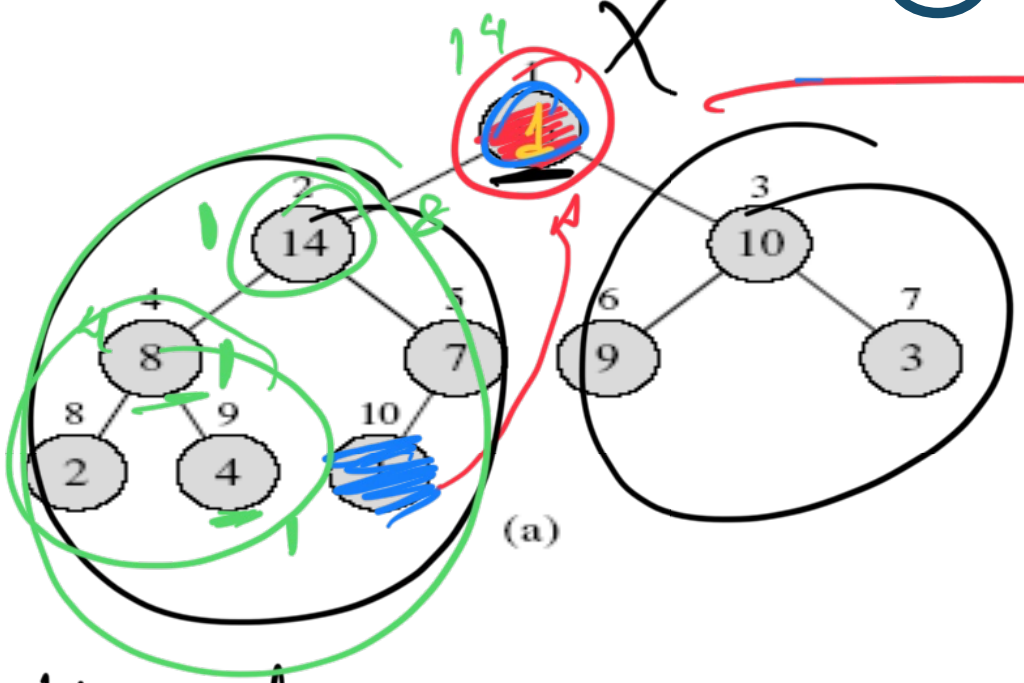
Heaps

- Eliminación (MAX-HEAP)
  - Eliminar último nodo: O(1)
  - ¿Eliminar raíz?
    - Intercambiar el último nodo por la raíz: O(1)
    - Eliminar último nodo (antes la raíz): O(1)
    - Se debe mantener la propiedad de MAX-HEAP
    - ¿costo computacional?
    - HEAPIFY!!!

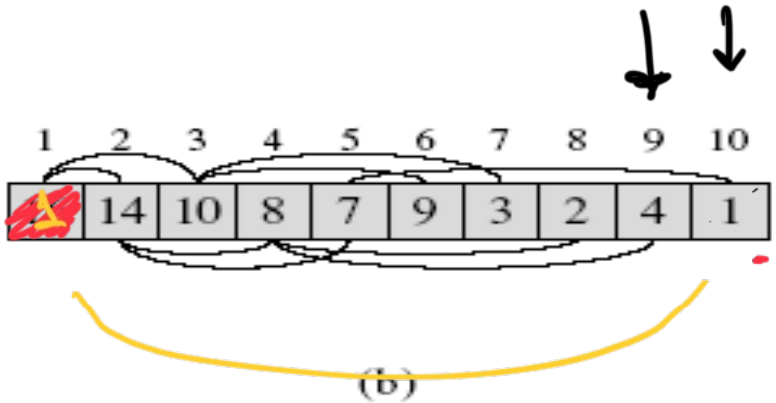
$n = 10$

$\log_2 10 = 3.\text{y}$ algo

Heapify $= O(h)$



16



Heapify

A[0] ↔ A[n]

A.size = A.size − 1

$O(2) \in O(1)$

Heaps

- HEAPIFY
  - Dos sub-árboles válidos en términos de MAX-HEAP
  - Pero la raíz puede no cumplir el MAX-HEAP
  - ¿Qué hacer?

Heaps

- HEAPIFY
  - Dos sub-árboles válidos en términos de MAX-HEAP
  - Pero la raíz puede no cumplir el MAX-HEAP
  - ¿Qué hacer?
  - Buscar el mayor de los hijos e intercambiarlo con el padre!
  - Aquel hijo que haya sido intercambiado, ahora debe ser verificado que cumple MAX-HEAP, si no, se vuelve a aplicar el mismo procedimiento con dicho sub-árbol, con el nodo hijo como raíz.

Resumen MAX-HEAP

- Encontrar Max: O(1)
- Insertar: O(log n)
- Eliminar (raíz/heapify): O(lg n)

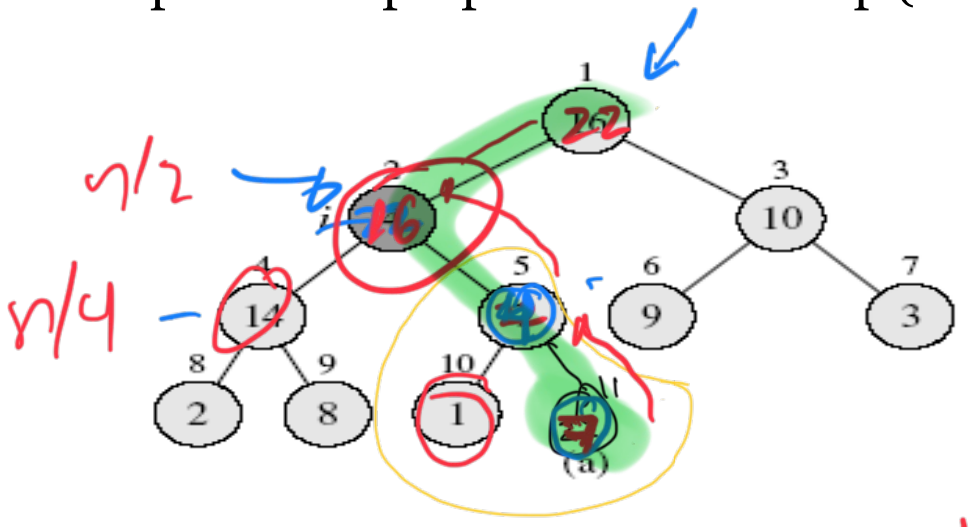
Heapify

- Algoritmo Heapify
  - Llamada recursiva a Heapify para sus i's subárboles, si no cumplen con la propiedad del heap

```
MAX-HEAPIFY(A, i)
 1  l ← LEFT(i)
 2  r ← RIGHT(i)
 3  if l ≤ heap-size[A] and A[l] > A[i]
 4    then largest ← l
 5    else largest ← i
 6  if r ≤ heap-size[A] and A[r] > A[largest]
 7     then largest ← r
 8  if largest ≠ i
 9     then exchange A[i] ↔ A[largest]
10          MAX-HEAPIFY(A, largest)
```

Heapify

- Heap-size[A] = 10
- A[2], i = 2 no cumple con la propiedad de un heap (MAX-HEAP)



$n/2$ → $n/4$

$h = 3$ $\log_2 n$

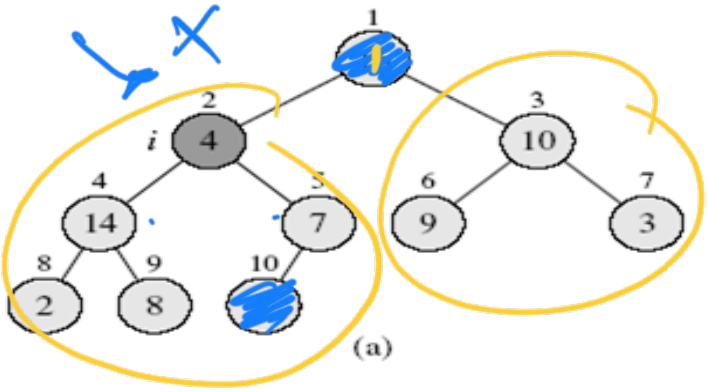
$h = 2$ $\log_2 (n/2)$

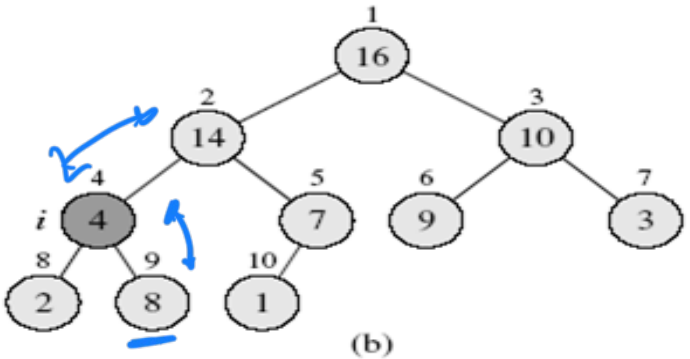
$h = 1$ $\log_2 (n/4)$

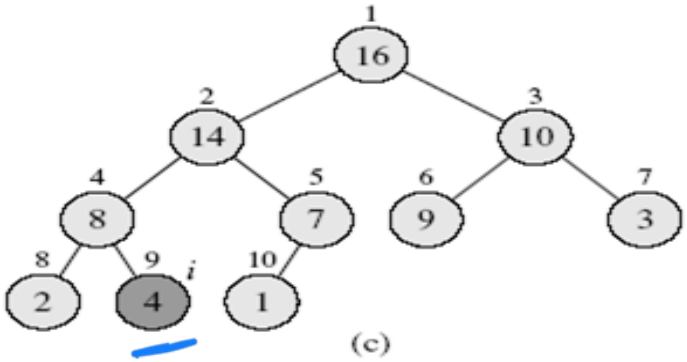
$h = 0$ $\log_2 (n/16)$

$O(h)$ $\log_2 n$ $h \log_2 n = \sum_{i=0}^{h} \log \left(\frac{n}{2^i}\right)$

Heapify


1
2
i
4
3
10
4
14
5
7
6
9
7
3
8
2
9
8
10
(a)

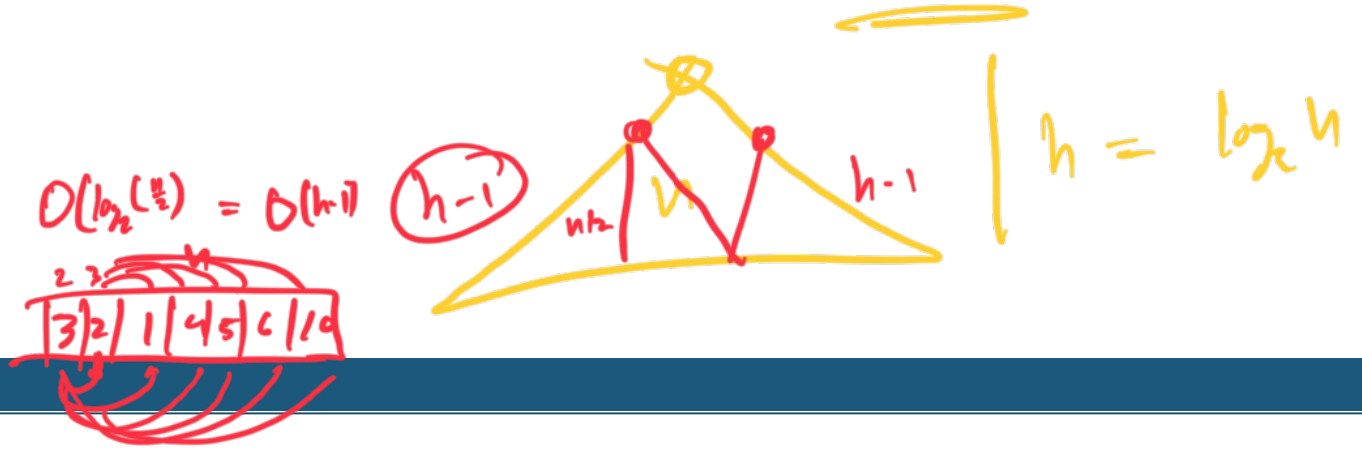

1
16
2
14
3
10
i
4
5
7
6
9
7
3
8
2
9
8
10
1
(b)


1
16
2
14
3
10
4
8
5
7
6
9
7
3
8
2
9
i
4
10
1
(c)

Heapify

Tiempo de ejecución de heapify

- En un subárbol de tamaño n, con raíz en el nodo i
  - Θ(1), relación fija entre A[i], A[LEFT(i)], A[RIGHT(i)] más
  - Tiempo de ejecución de HEAPIFY en un subárbol con raíz en uno de los hijos de i
    - T(n) = O(lgn)
  - Tiempo de ejecución de HEAPIFY en un nodo de altura h = O(h) $= O(\log_2 n)$



Construcción del Heap

- Insertar elementos de forma iterativa
  - Insertar al final del arreglo
  - Verificar que MAX-HEAP se cumple
  - ¿Complejidad?

Construcción del Heap

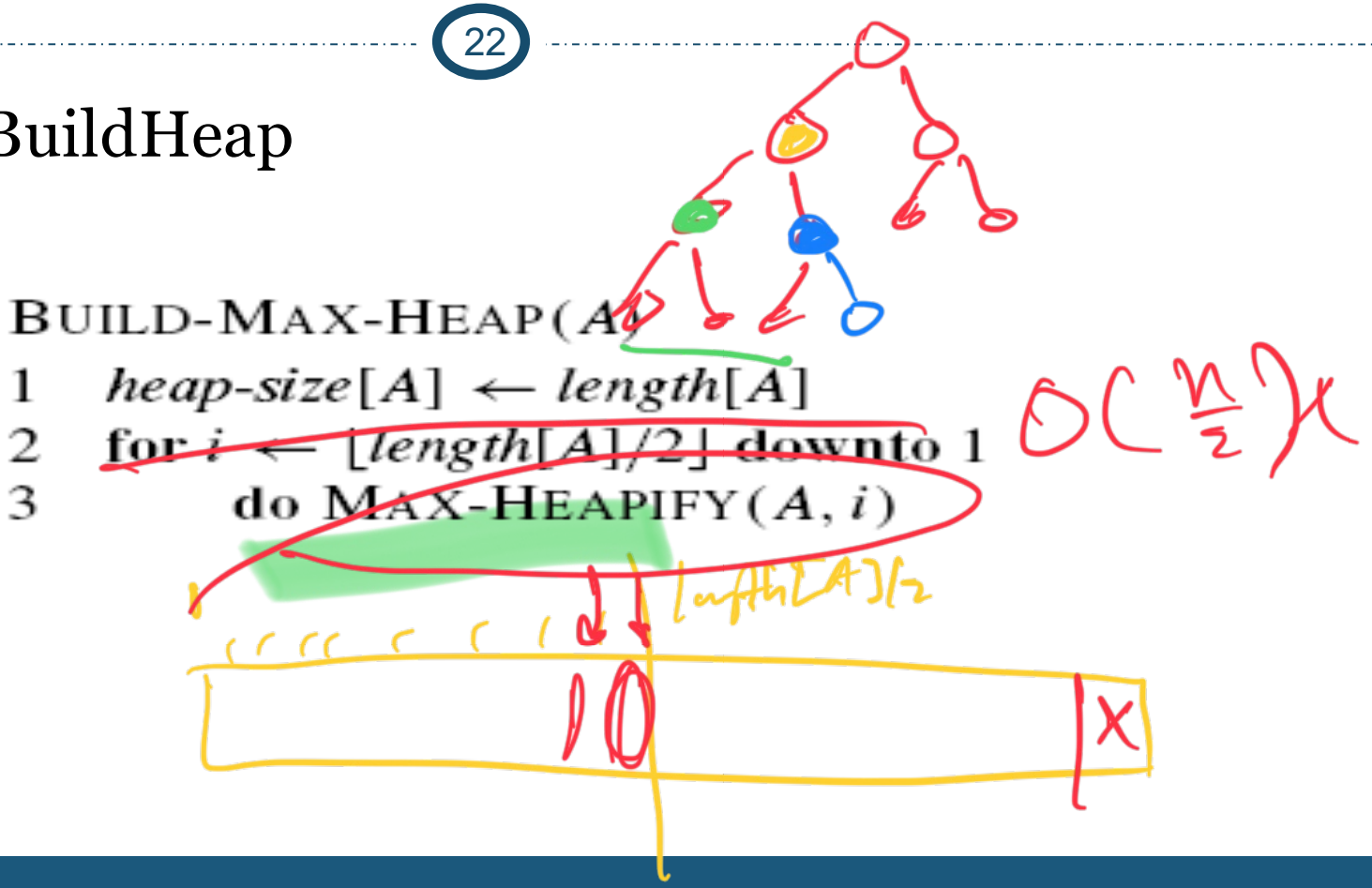
- Insertar elementos de forma iterativa
  - Insertar al final del arreglo
  - Verificar que MAX-HEAP se cumple
  - ¿Complejidad? **O(n lg n)**

Construcción del Heap

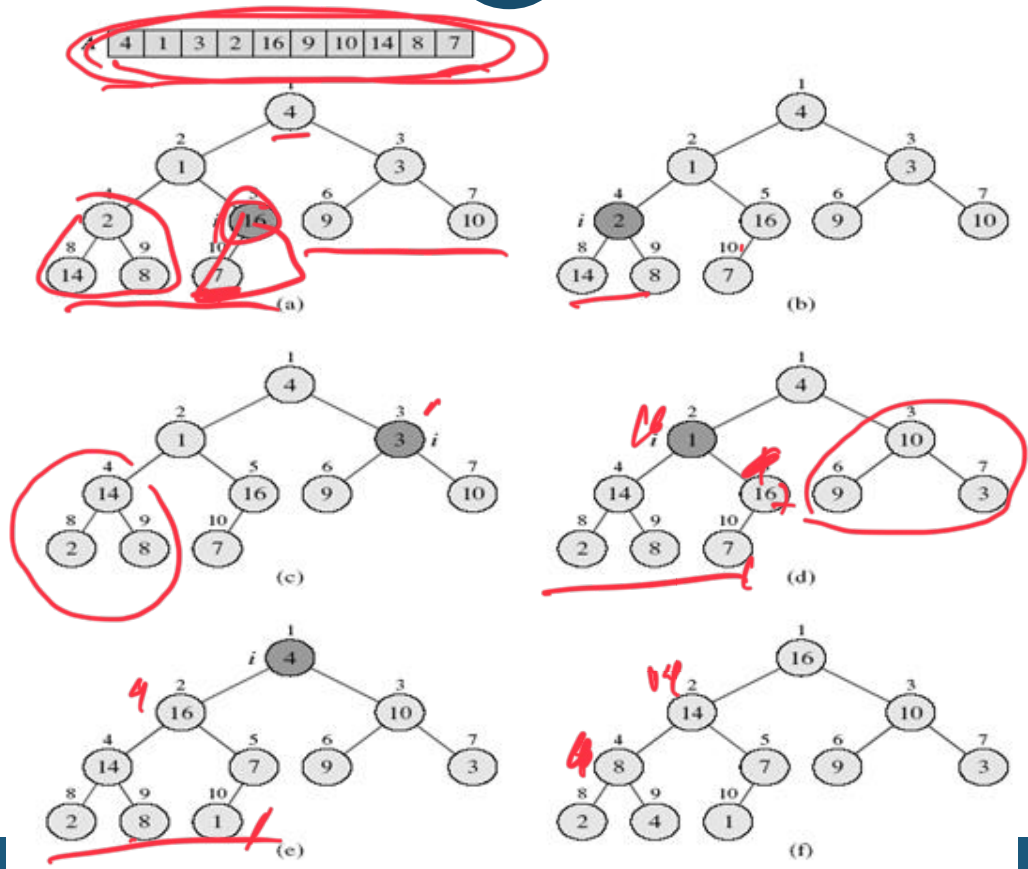
- Uso de heapify para construir el heap (¿peor caso?)
  - Los elementos A[(floor(n/2) + 1) … n] son hojas del árbol
  - Orden en que se procesan los nodos garantiza que los subárboles con raíz en los hijos del nodo i son heaps antes de que HEAPIFY se ejecute en dicho nodo

Construcción del Heap

- Algoritmo BuildHeap



Construcción del Heap



¿Se entendió?

Análisis BuildHeap

- Cada llamada a HEAPIFY □ O(lgn), hay O(n) llamadas
  - Cuando mucho O(nlgn)
    - Cota superior pero no justa
  - Propiedad: En un heap de n-elementos hay cuando mucho $(n/2^{h+1})$ nodos de altura h
    - Tiempo de ejecución de HEAPIFY en nodo altura h = O(h)
    - Costo de BuildHeap

$O(\frac{n}{2}) \in O(n)\, O(\lg n)$

$O(n \log n)$

Análisis BuildHeap

Del resultado anterior obtenemos el tiempo de ejecución de BuildHeap

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left(\frac{n}{2^{h+1}} \right) O(h) = O\left(n * 1/2 \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \right)$$

Note que $\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$, para $|x| < 1$. La derivada de ambos lados,

$\frac{d}{dx}\left(\sum_{k=0}^{\infty} x^k \right) = \frac{d}{dx}\left(\frac{1}{1-x} \right)$, es igual a $\sum_{k=0}^{\infty} k x^{k-1} = \frac{1}{(1-x)^2}$.

Multiplicando ambos lados de la equivalencia por x obtenemos:

$$\sum_{h=0}^{\infty} k x^k = \frac{x}{(1-x)^2}.$$

En nuestro caso k = h y x = 1/2.

Entonces $\sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{1/2}{(1-1/2)^2} = 2, x = 1/2$.

Entonces el tiempo de ejecución es $O(n*1/2*2) = O(n)$.