

Ayudantía 7

ESTRUCTURA DE DATOS Y ALGORITMOS

PROFESOR: YERKO ORTIZ

AYUDANTE: VICENTE DIAZ

SECCIÓN 4



Contacto

Discord: trapitokid

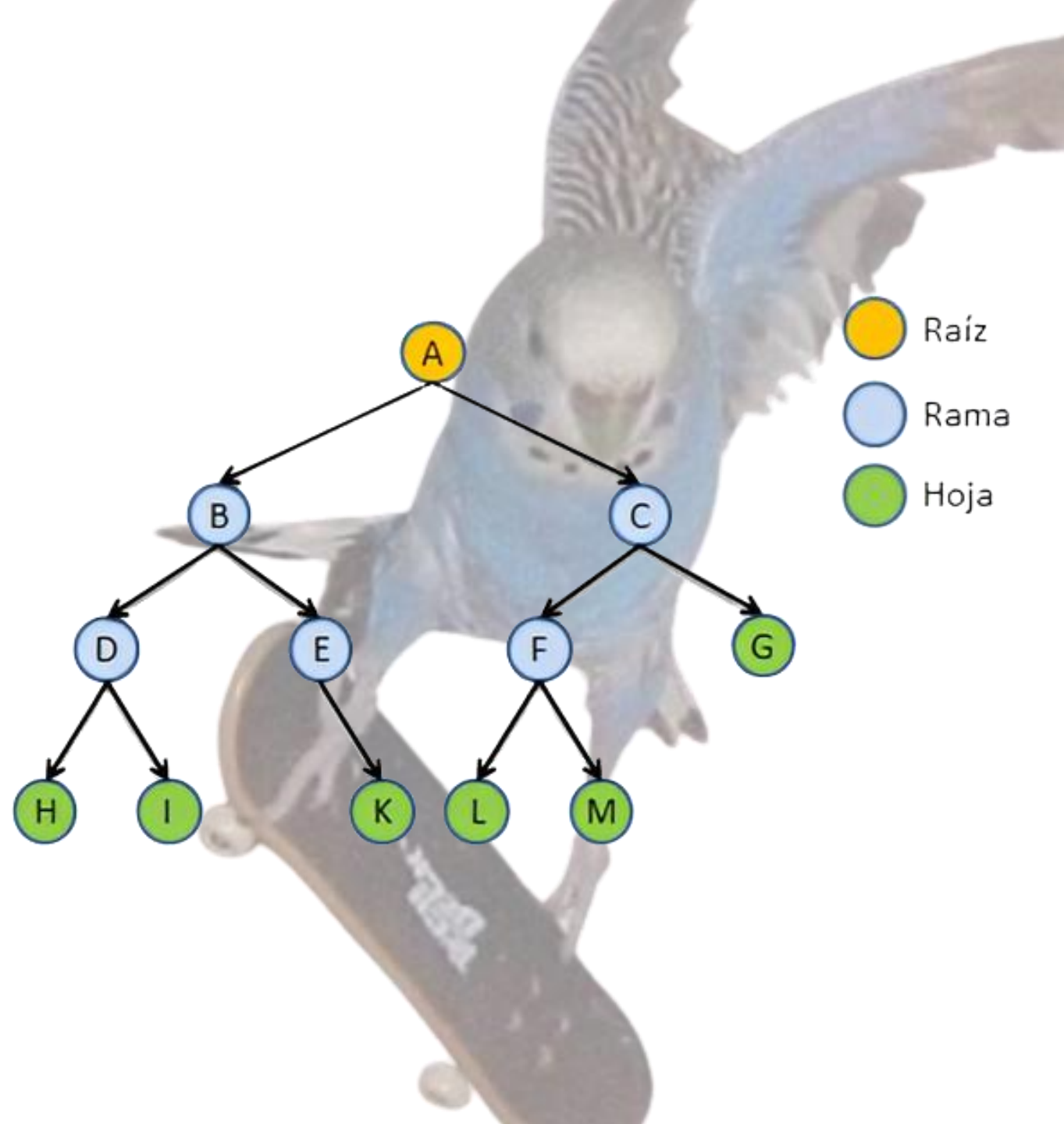
Wsp: +569 5749 6014

LoL: Saki TvT#woof



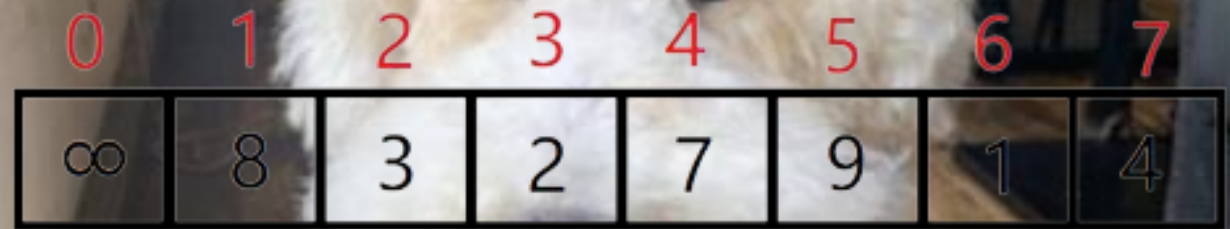
Arboles Binarios

- Cada nodo i mantiene una única referencia a un nodo anterior que se considera padre del nodo i en la jerarquía.
- La altura es la distancia máxima entre la raíz del árbol hacia alguna de sus hojas.
- Un árbol binario con altura " h " **a lo más** tiene $2^h - 1$ nodos.
- Un árbol binario perfecto con altura " h " tiene **a lo menos** 2^{h-1} nodos.



Heap Sort

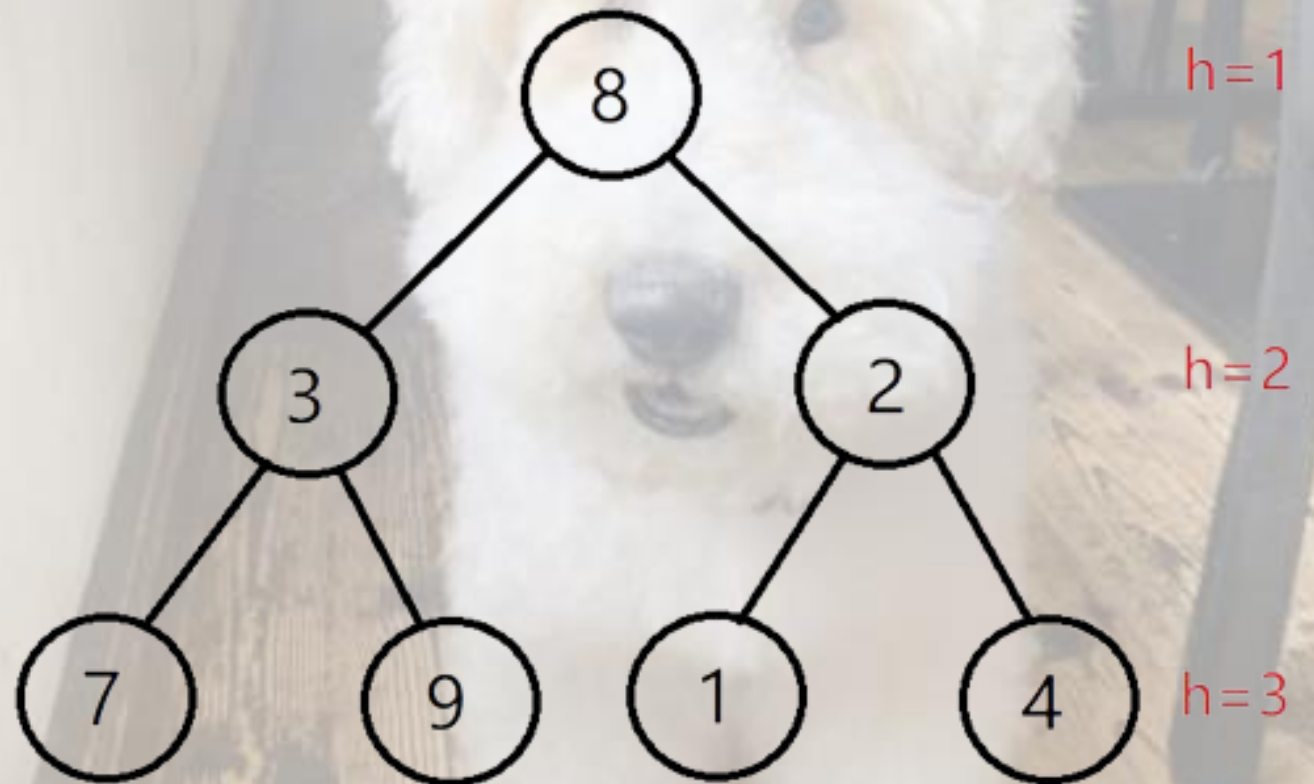
- Busca transformar un arreglo en un heap, en otras palabras, heapificar un arreglo (heapify).
- Hojas en un heap:
 $A[\text{heapsize}/2 + 1 : \text{heapsize}]$
- Root: $A[1]$
Left: $2i$
Right: $2i+1$
Parent: $i/2$
- $O(n \log(n))$
In place
No estable



Heap Sort

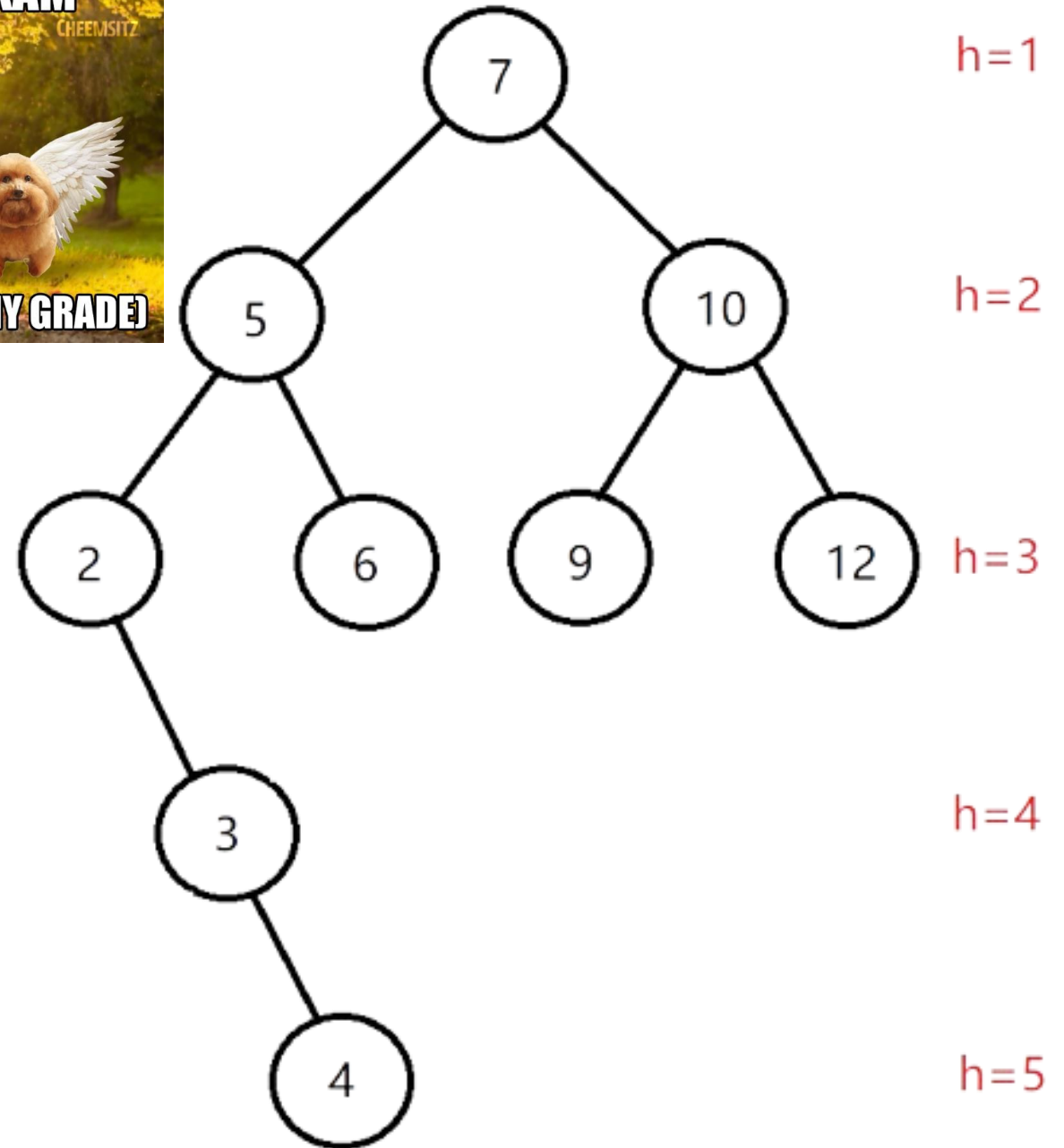
- Busca transformar un arreglo en un heap, en otras palabras, heapificar un arreglo (heapify).
- Hojas en un heap:
 $A[\text{heapsize}/2 + 1 : \text{heapsize}]$
- Root: $A[1]$
Left: $2i$
Right: $2i+1$
Parent: $i/2$
- $O(n \log(n))$
In place
No estable

0	1	2	3	4	5	6	7
∞	8	3	2	7	9	1	4



Binary Search Tree BST

- Se debe cumplir la siguiente propiedad:
 $x.key \geq x.left.key$
 $x.key < x.right.key$
- Existen métodos para operar con los nodos como:
insert(key)
search(key)
- Los BST tienen 3 manera de recorrerse:
InOrder(root)
PreOrder(root)
PostOrder(root)



InOrder

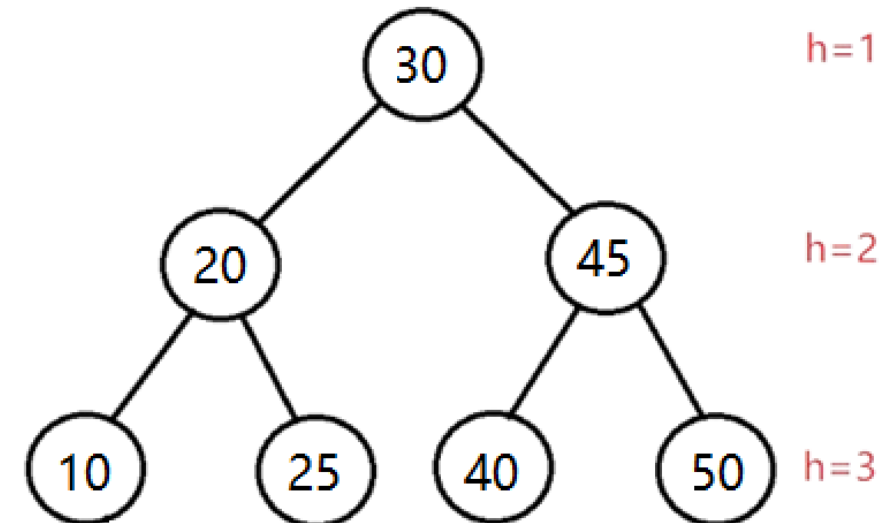
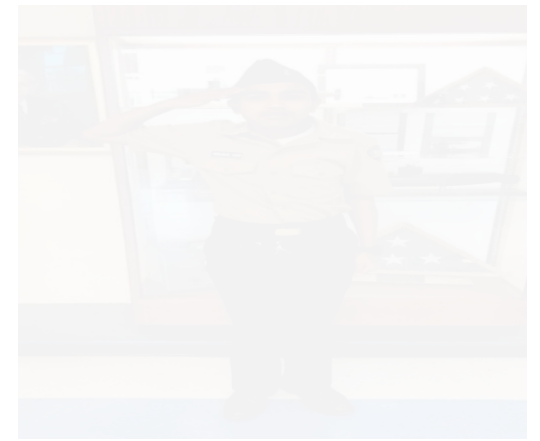
```
InOrder(root)
if(root == null){
    return
}
InOrder(root.left)
Print(root.key)
InOrder(root.right)
```

PreOrder

```
PreOrder(root)
if(root == null){
    return
}
Print(root.key)
PreOrder(root.left)
PreOrder(root.right)
```

PostOrder

```
PostOrder(root)
if(root == null){
    return
}
PostOrder(root.left)
PostOrder(root.right)
Print(root.key)
```



InOrder

```
InOrder(root)
if(root == null){
    return
}
InOrder(root.left)
Print(root.key)
InOrder(root.right)
```

PreOrder

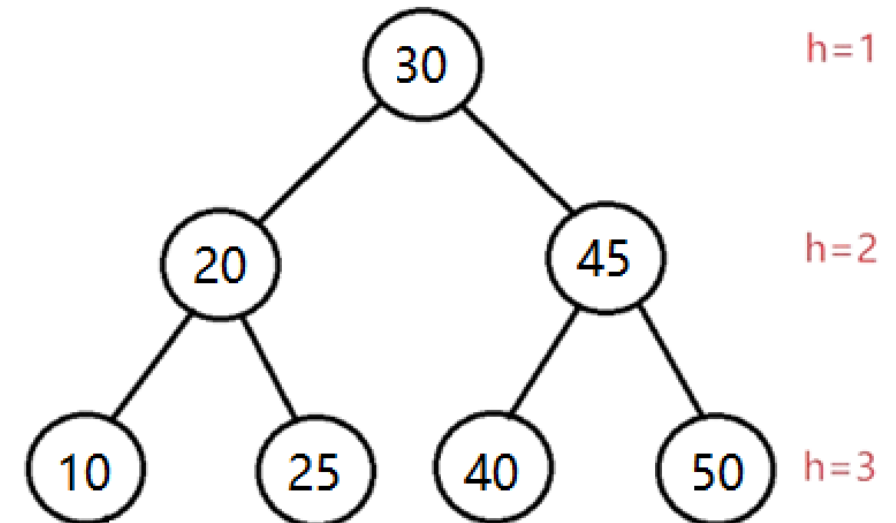
```
PreOrder(root)
if(root == null){
    return
}
Print(root.key)
PreOrder(root.left)
PreOrder(root.right)
```

PostOrder

```
PostOrder(root)
if(root == null){
    return
}
PostOrder(root.left)
PostOrder(root.right)
Print(root.key)
```



[10,20,25,30,40,45,50]



InOrder

```
InOrder(root)
if(root == null){
    return
}
InOrder(root.left)
Print(root.key)
InOrder(root.right)
```

[10,20,25,30,40,45,50]



PreOrder

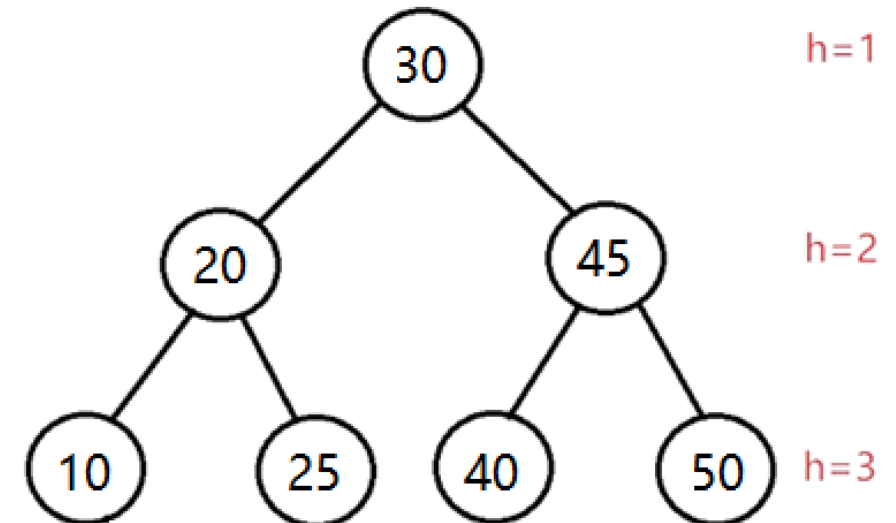
```
PreOrder(root)
if(root == null){
    return
}
Print(root.key)
PreOrder(root.left)
PreOrder(root.right)
```

[30,20,10,25,45,40,50]



PostOrder

```
PostOrder(root)
if(root == null){
    return
}
PostOrder(root.left)
PostOrder(root.right)
Print(root.key)
```



InOrder

```
InOrder(root)
if(root == null){
    return
}
InOrder(root.left)
Print(root.key)
InOrder(root.right)
```

[10,20,25,30,40,45,50]



PreOrder

```
PreOrder(root)
if(root == null){
    return
}
Print(root.key)
PreOrder(root.left)
PreOrder(root.right)
```

[30,20,10,25,45,40,50]



PostOrder

```
PostOrder(root)
if(root == null){
    return
}
PostOrder(root.left)
PostOrder(root.right)
Print(root.key)
```

[10,25,20,40,50,45,30]

