Data Structures and Algorithms

(CS210A)

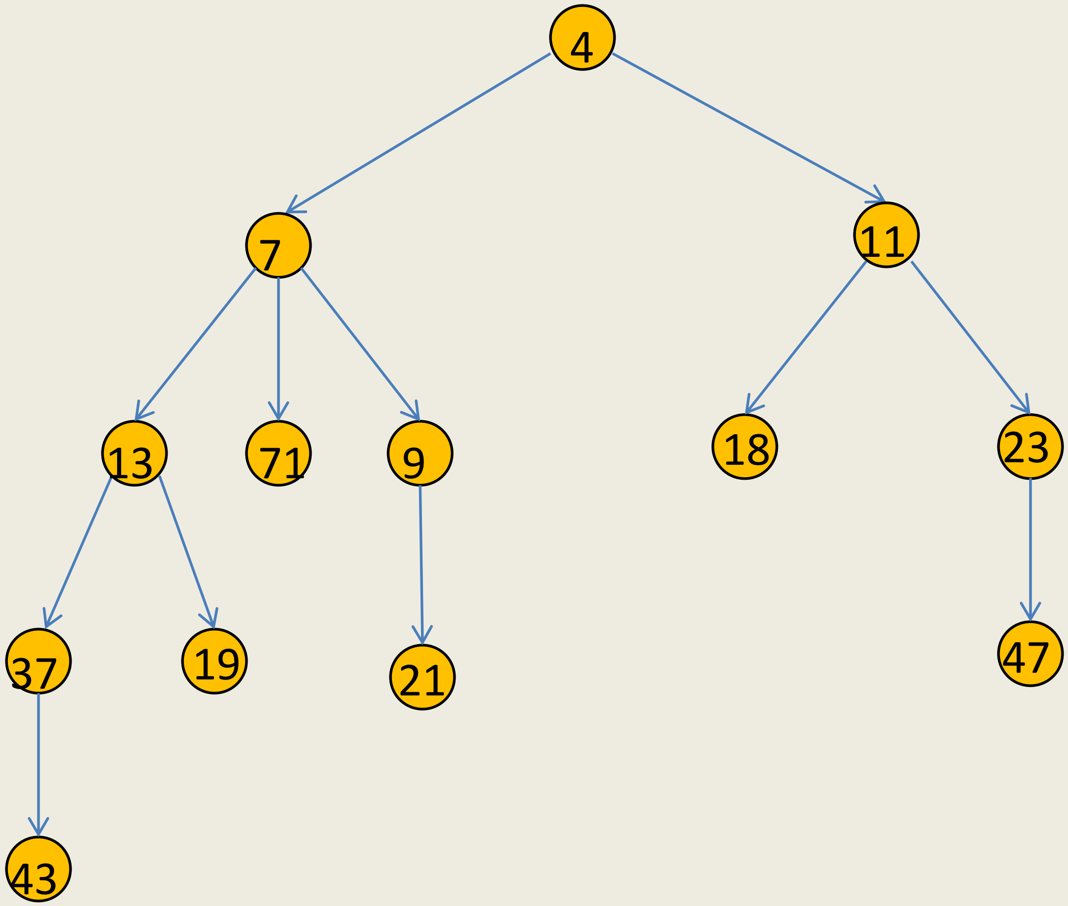
Lecture 28:

- **Heap :** an important tree data structure
- Implementing some **special binary tree** using an **array** !
- **Binary heap**

Heap

Definition: a tree data structure where :

value stored in a node $<$ value stored in each of its children.



Operations on a heap

Query Operations

- **Find-min**: report the smallest key stored in the heap.

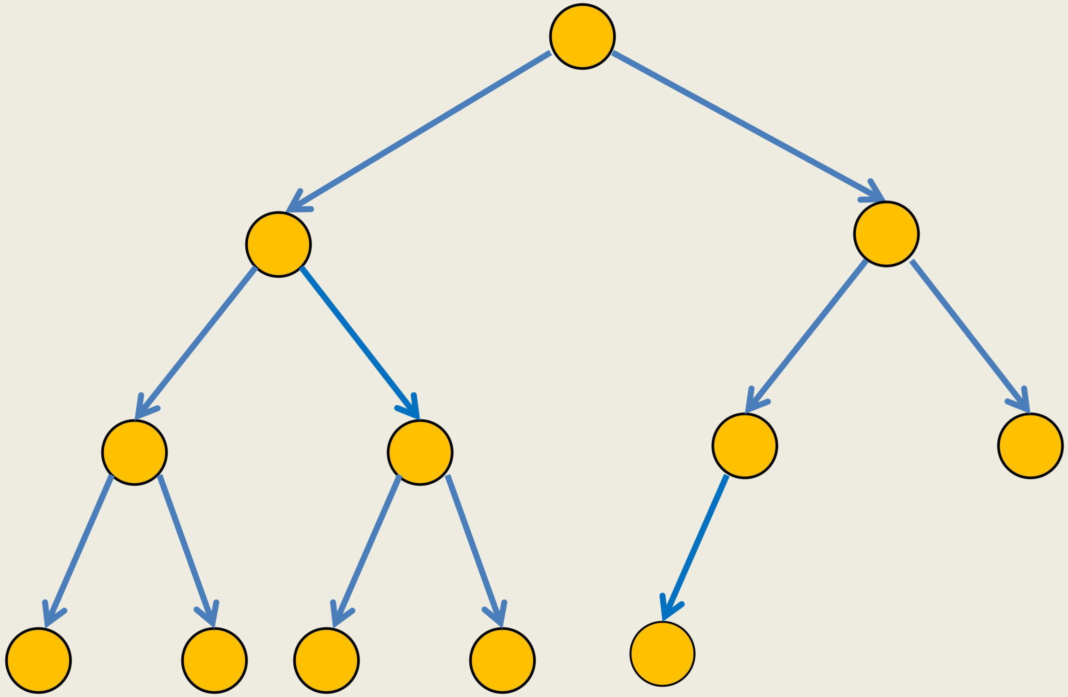
Update Operations

- **CreateHeap**(**H**) : Create an empty heap **H**.
- **Insert**(**x**,**H**) : Insert a new key with value **x** into the heap **H**.
- **Extract-min**(**H**) : delete the smallest key from **H**.
- **Decrease-key**(p, Δ, **H**) : decrease the value of the key p by amount Δ.
- **Merge**(H_1,H_2) : Merge two heaps H_1 **and** H_2.

Can we implement
a binary tree using an array ?

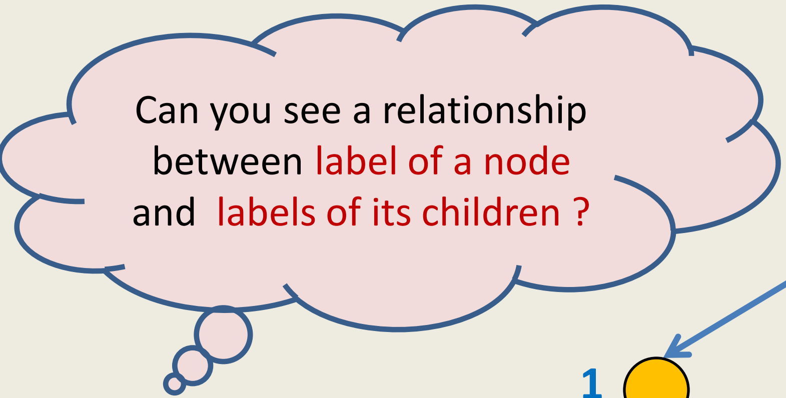
Yes.
In some special cases

A complete binary tree

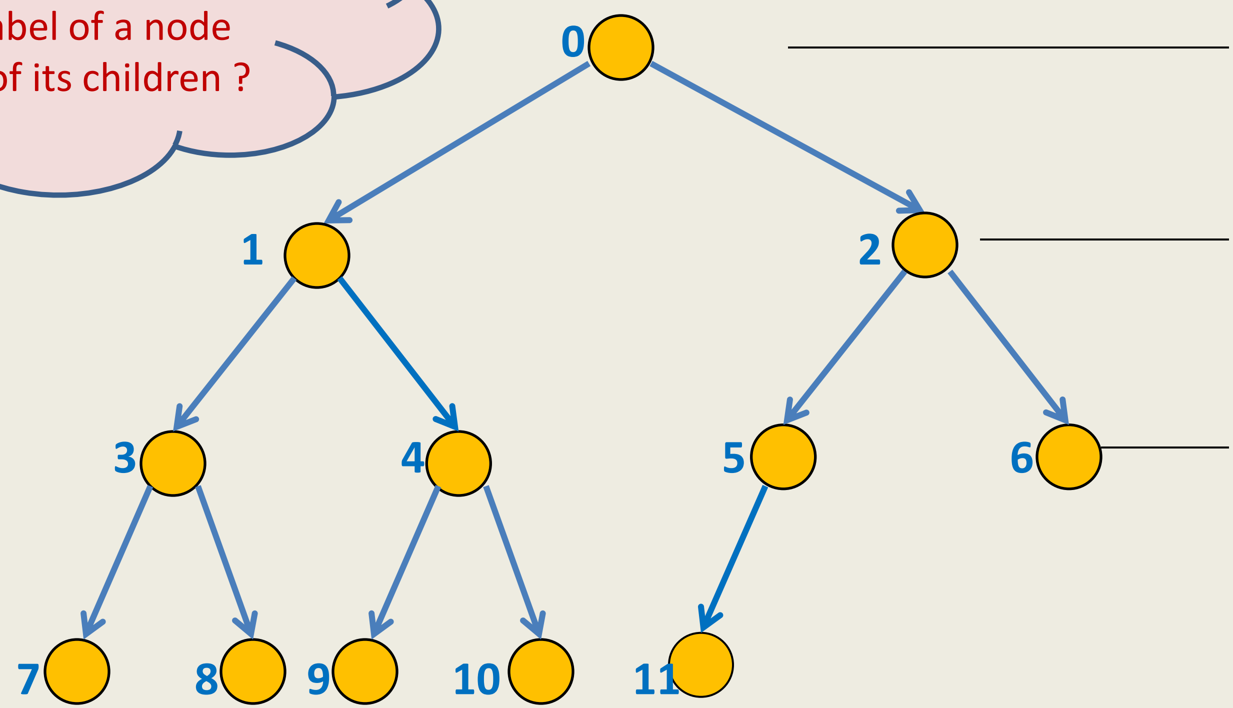


A complete binary of 12 nodes.

A complete binary tree



Level	nodes
0	1
1	2
2	4



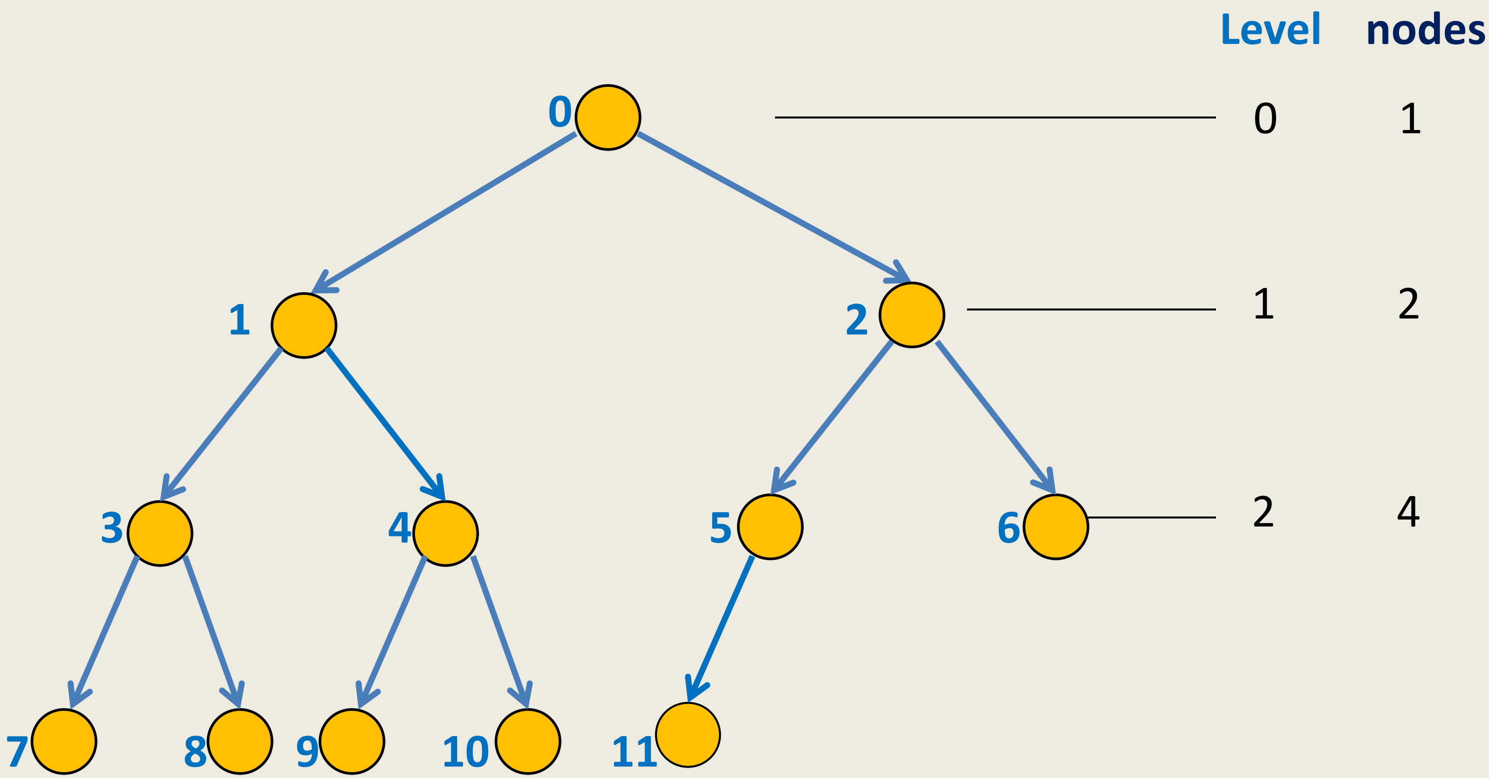
The label of the **leftmost node** at level i = $2^i - 1$

The label of a **node** **v** at level i $\ldots - 2$

The label of the **left** child of **v** is= $2^{i+1} - 1 + 2(k-1)$

The label of the **right** child of **v** is= $2^{i+1} + 2k - 2$

A **complete** binary tree



Let **v** be a node with label j.

Label of **left child(v)** = $2\,j+1$

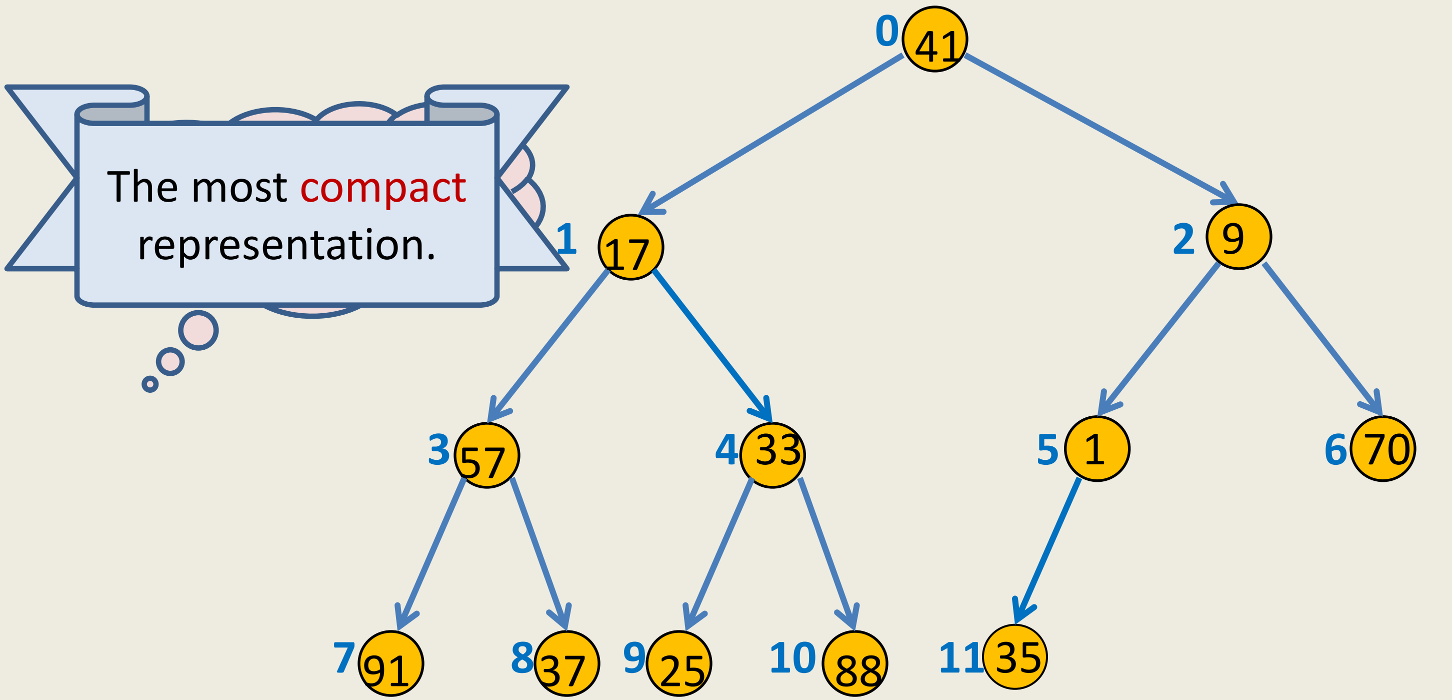
Label of **right child(v)** = $2j+2$

Label of **parent(v)** = $\lfloor (j-1)/2 \rfloor$

A complete binary tree and array

Question: What is the relation between a complete binary trees and an array ?

Answer: A complete binary tree can be **implemented** by an array.

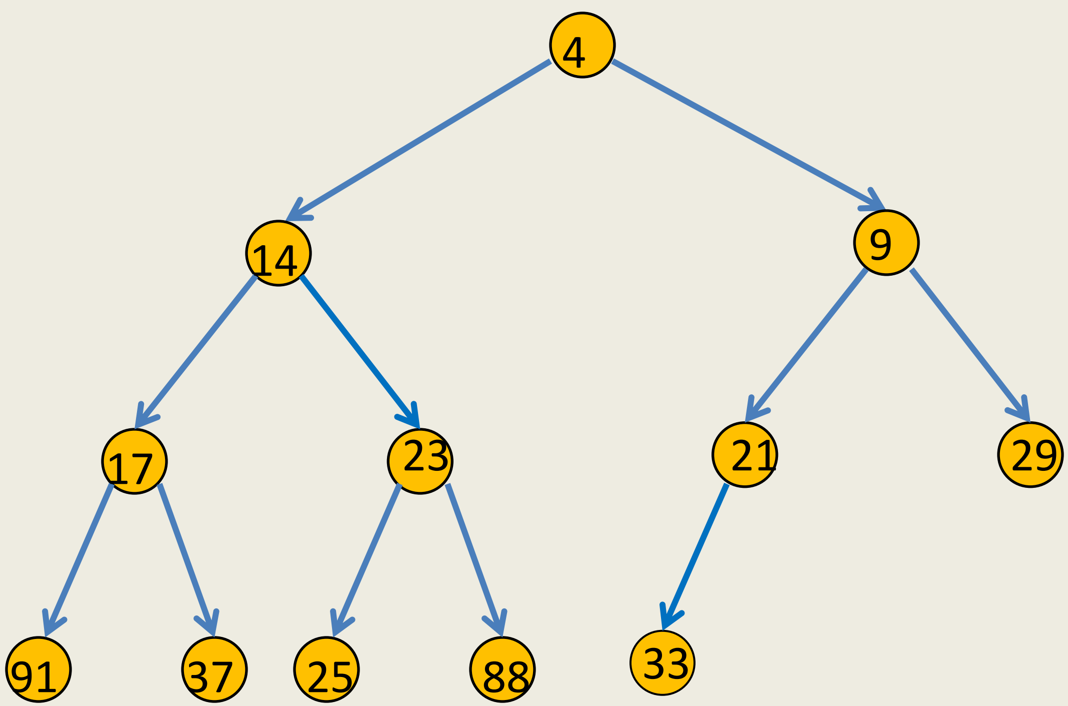


41	17	9	57	33	1	70	91	37	25	88	35
0	1	2	3	4	5	6	7	8	9	10	11

Binary heap

Binary heap

a **complete** **binary tree** satisfying **heap** property at each node.



H															
	4	14	9	17	23	21	29	91	37	25	88	33			

Implementation of a Binary heap

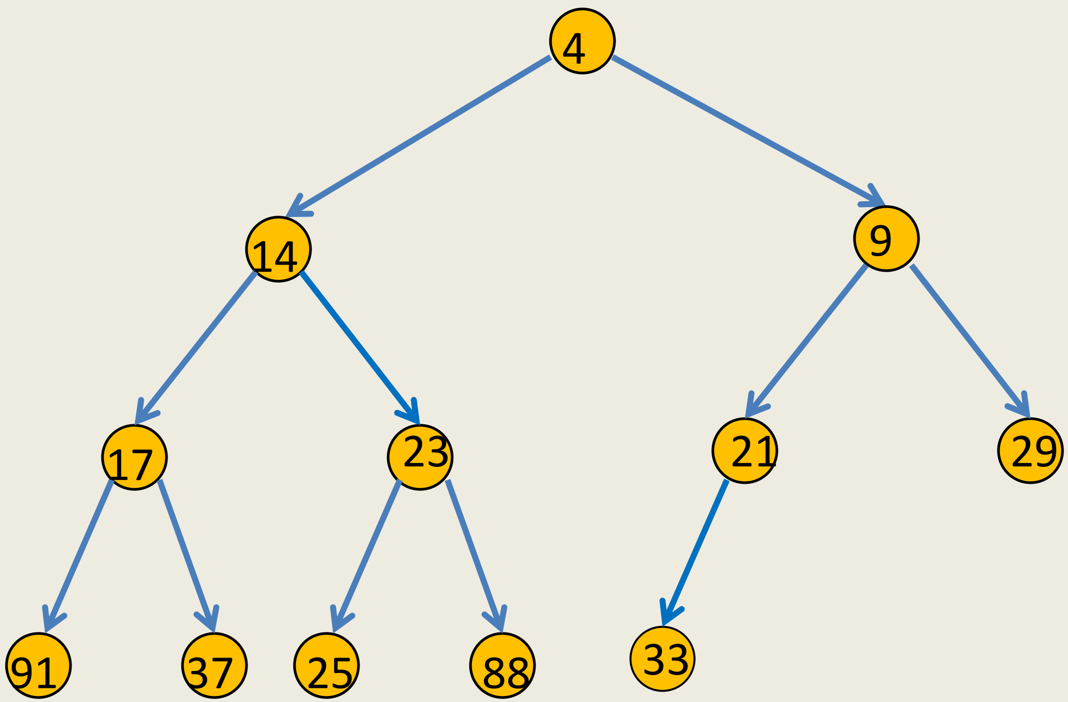
n

then we keep

- **H**[] : an **array** of size n used for storing the binary heap.
- **size** : a **variable** for the total number of keys <u>**currently**</u> in the heap.

Find_min(H)

Report **H**[0].



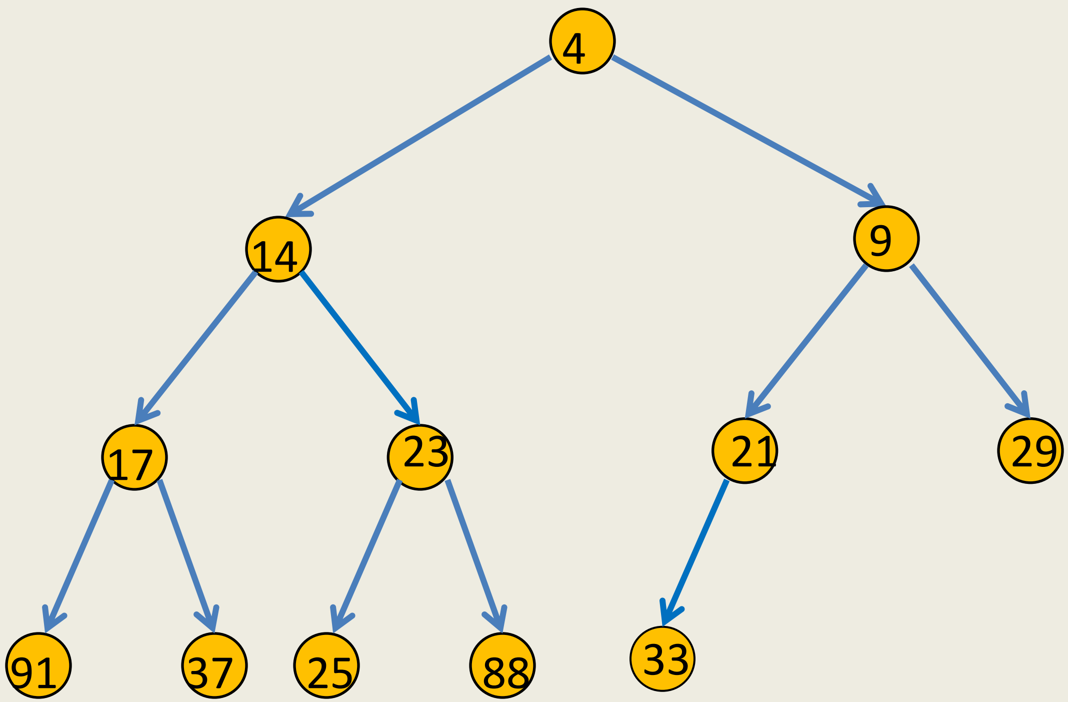
H	4	14	9	17	23	21	29	91	37	25	88	33			

Extract_min(H)

Think hard on designing efficient algorithm for this operation.

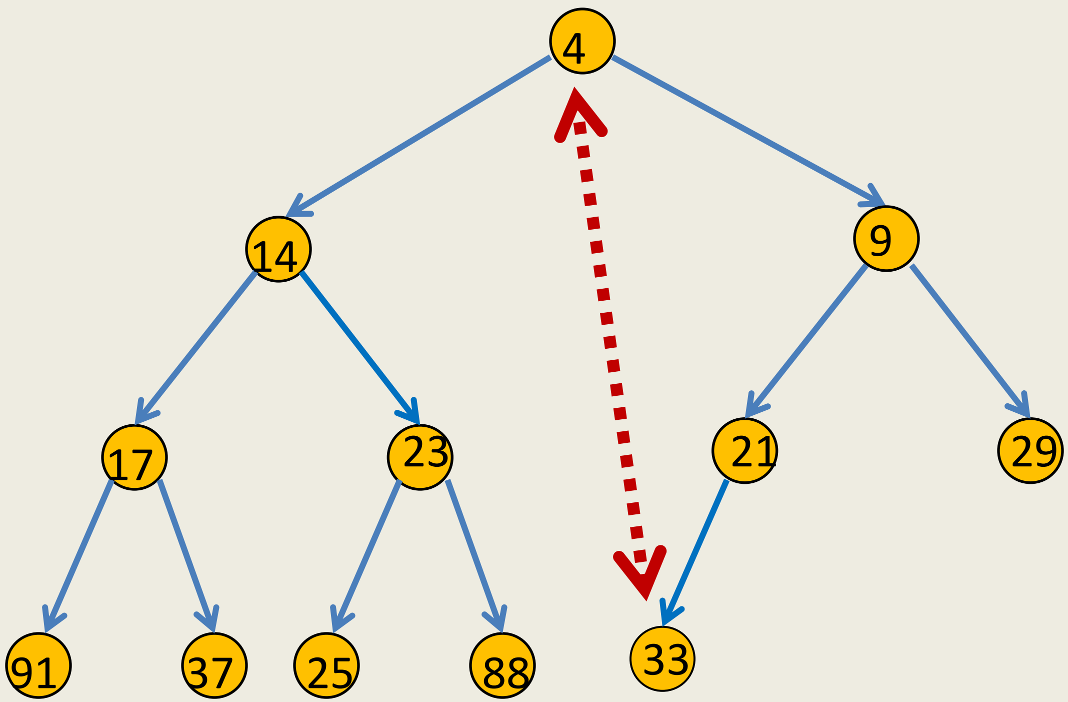
The challenge is:

how to preserve the complete binary tree structure as well as the heap property ?



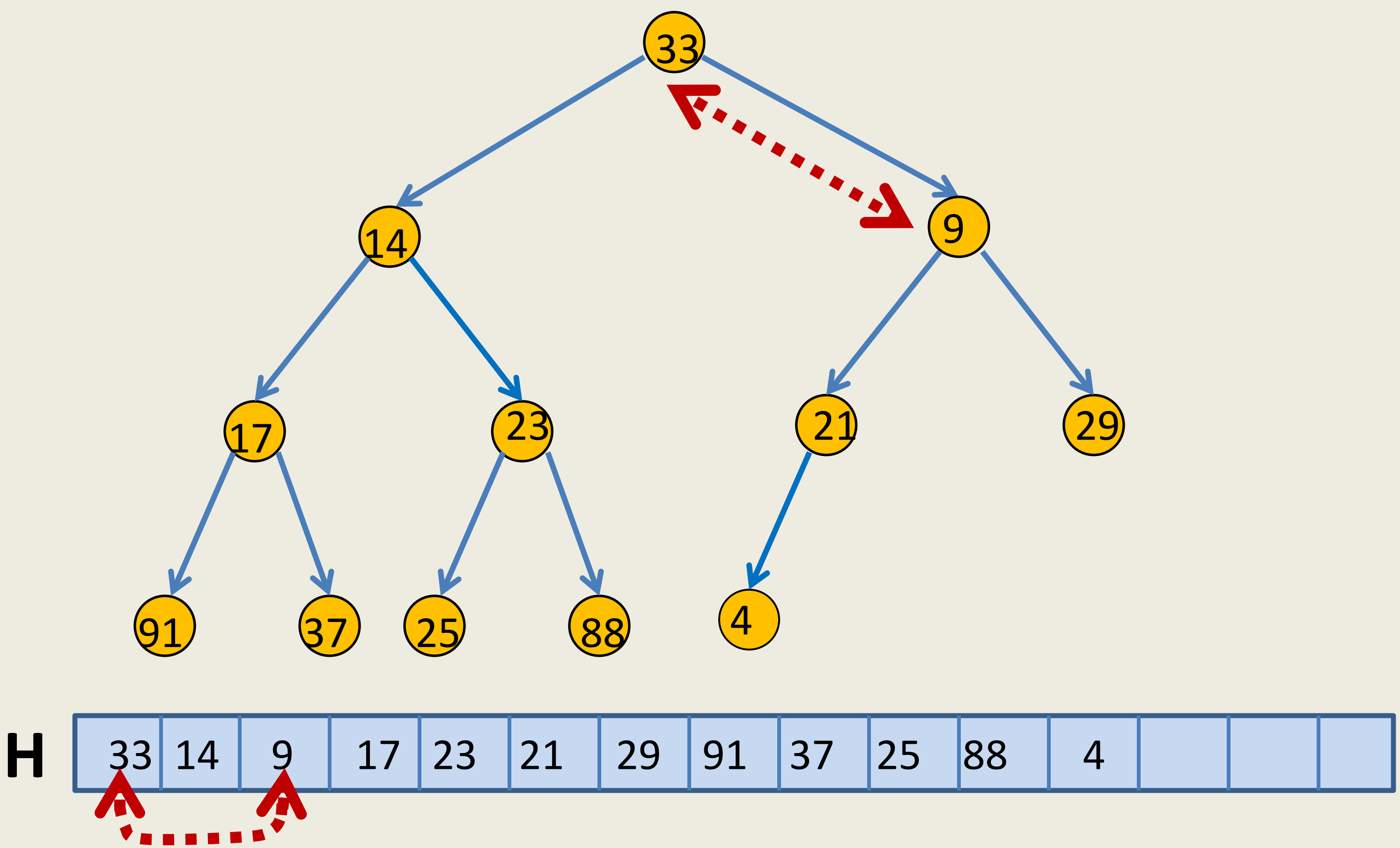
H	4	14	9	17	23	21	29	91	37	25	88	33			

Extract_min(H)

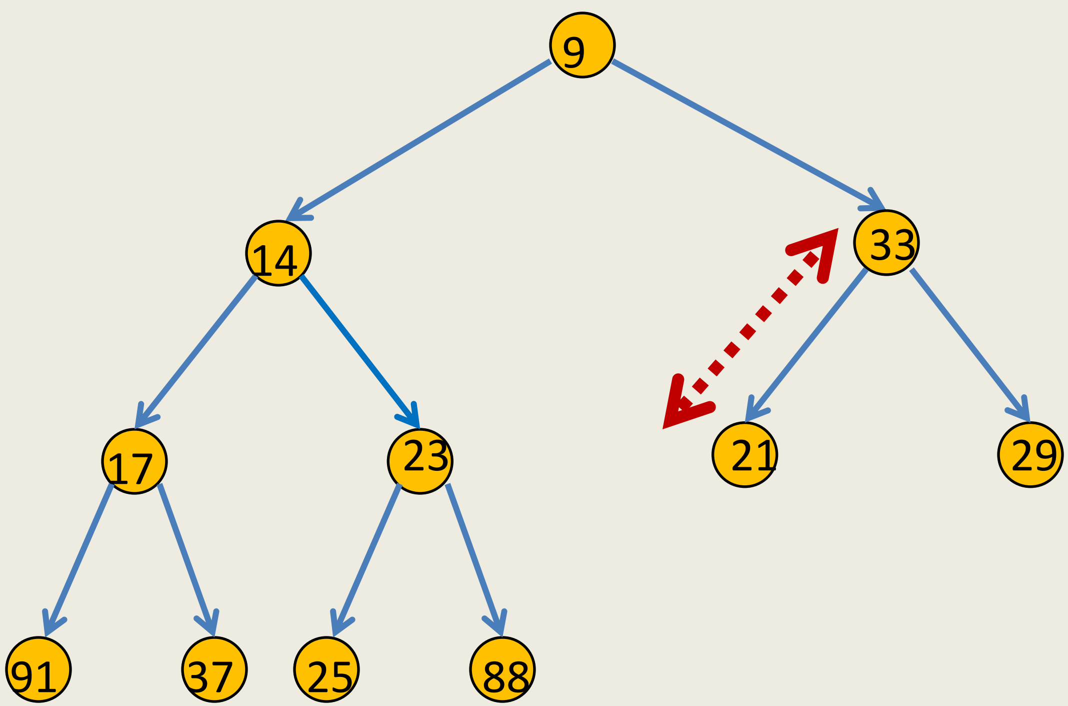


H														
4	14	9	17	23	21	29	91	37	25	88	33			

Extract_min(H)



Extract_min(H)



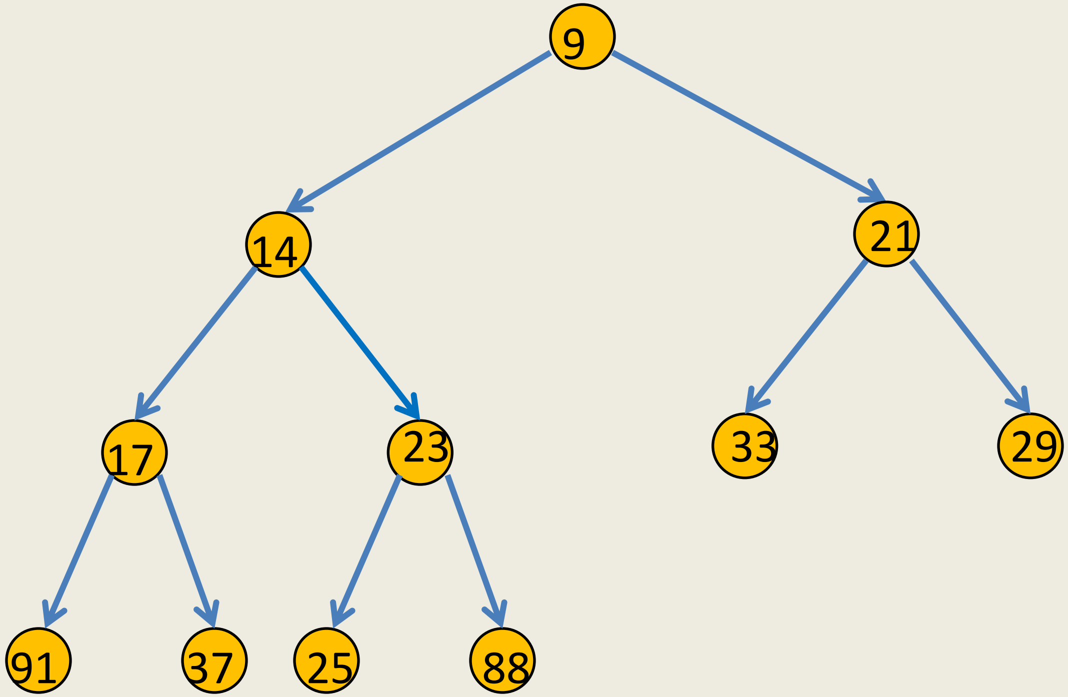
H	9	14	33	17	23	21	29	91	37	25	88				

Extract_min(H)

We are done.

The no. of operations performed = **O**(no. of levels in binary heap)

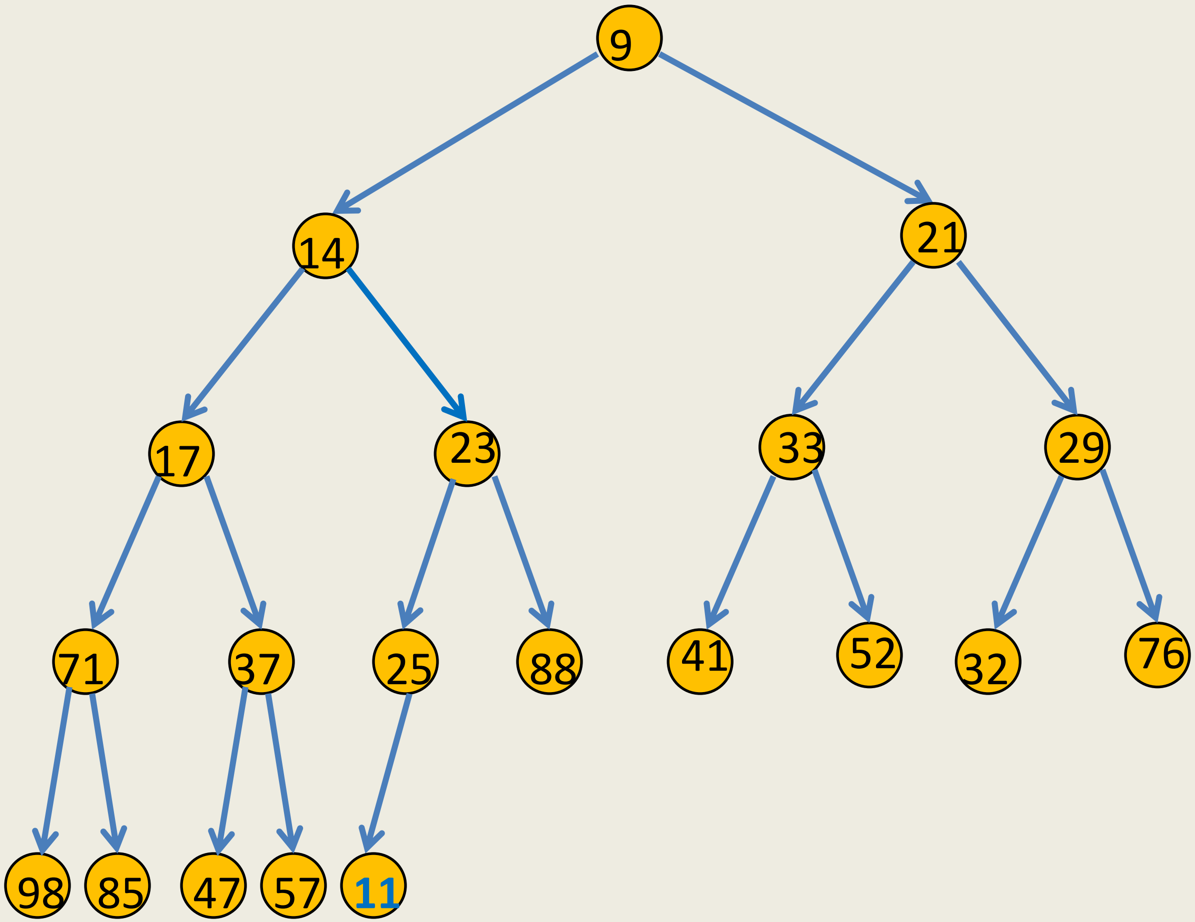
= **O**(log n) …show it as an **homework exercise**.



H

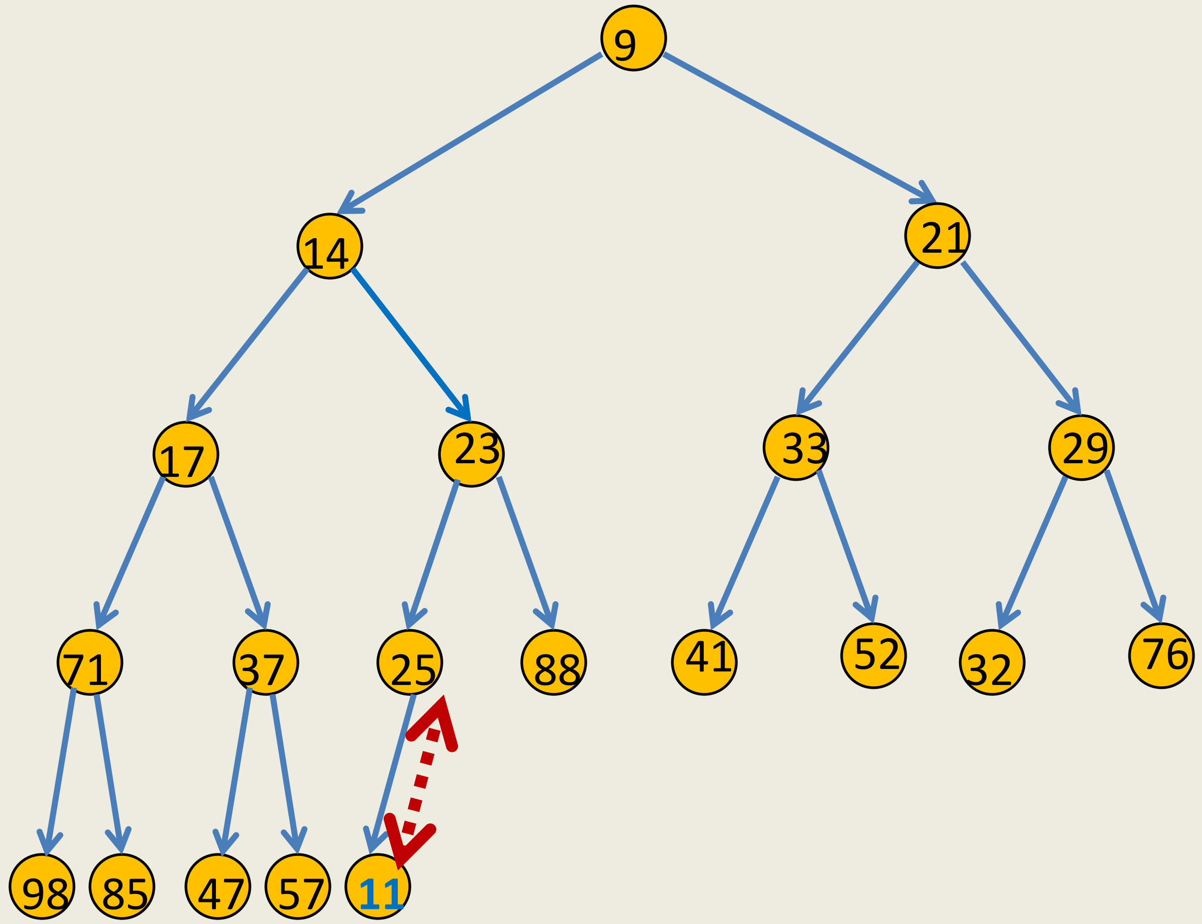
9	14	21	17	23	33	29	91	37	25	88				

Insert(x,H)



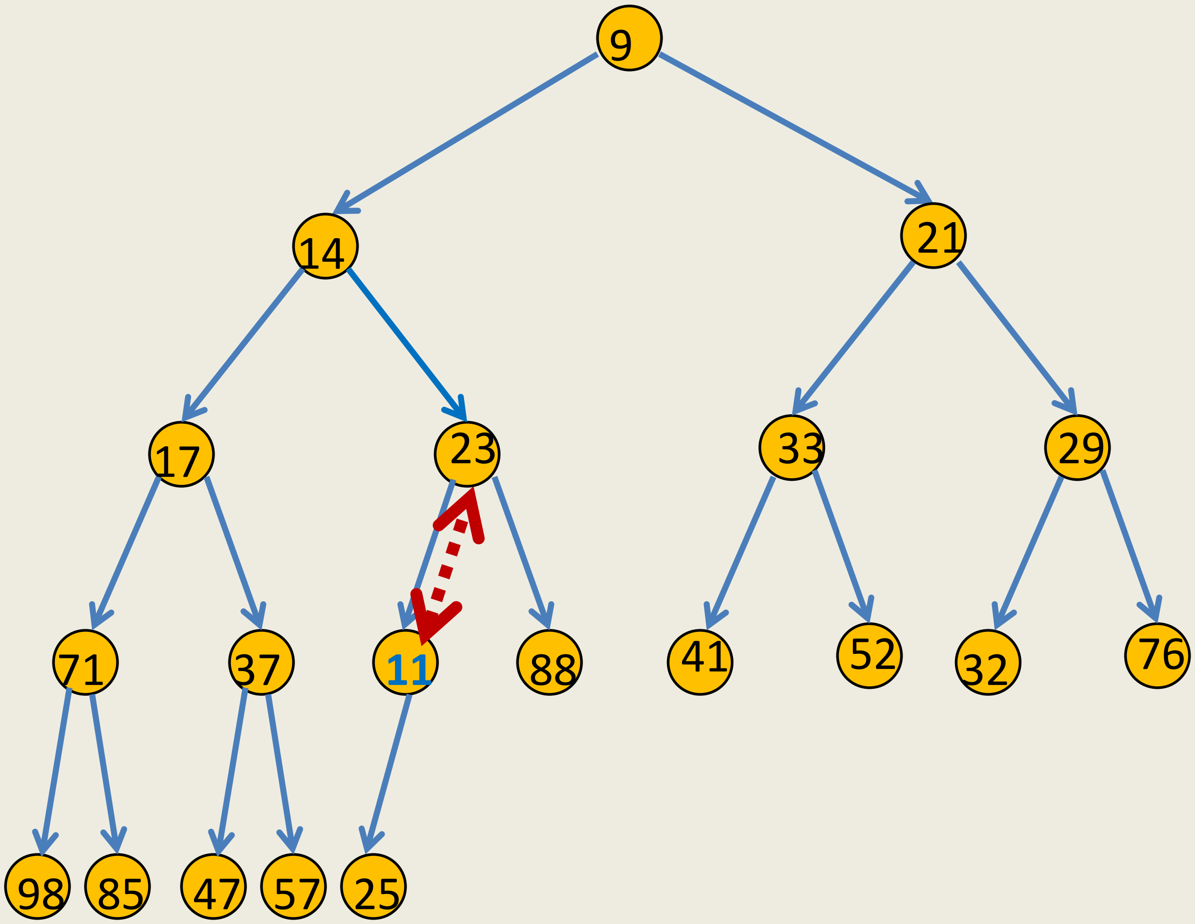
H	9	14	21	17	23	33	29	71	37	25	88	41	52	32	76	98	85	47	57	11	

Insert(x,H)



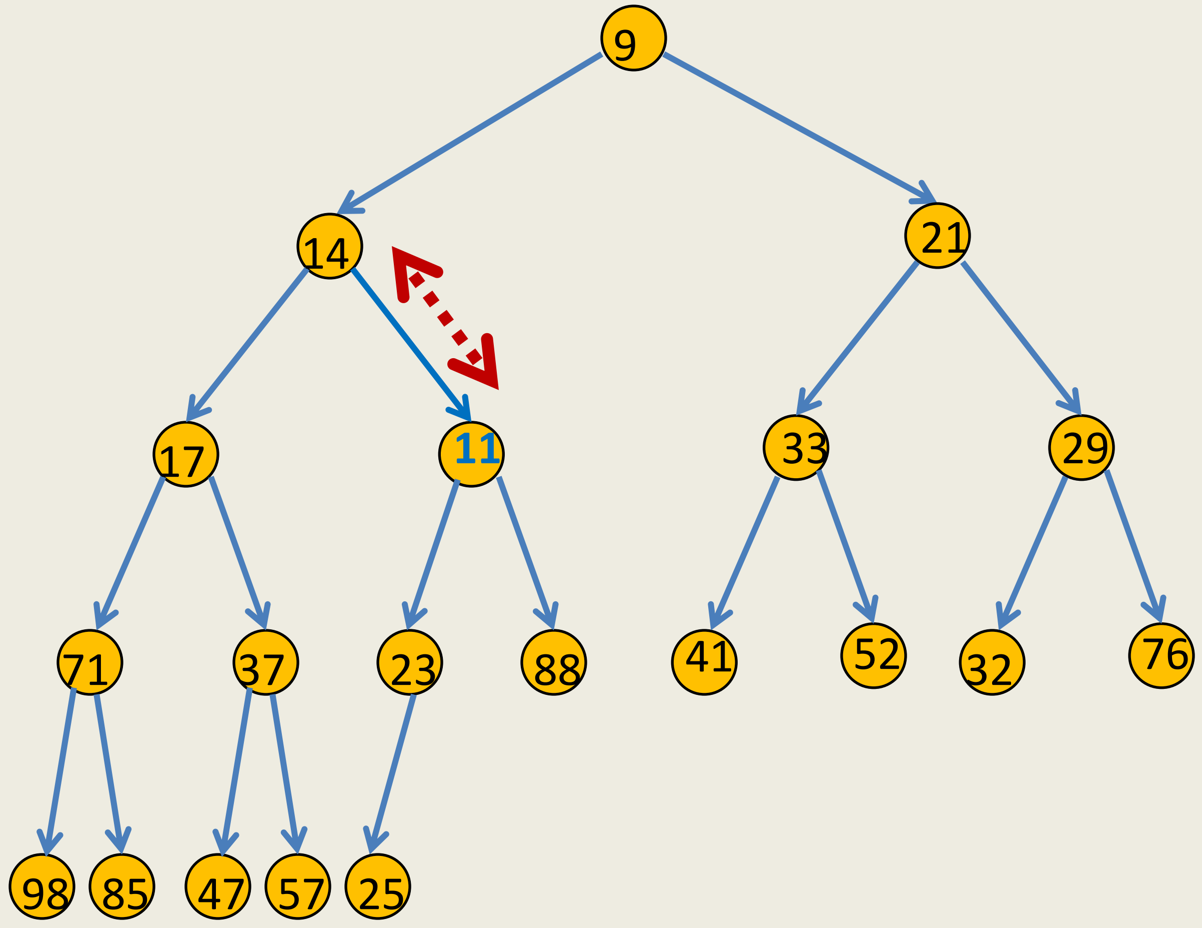
H																				
9	14	21	17	23	33	29	71	37	25	88	41	52	32	76	98	85	47	57	11	

Insert(x,H)



H																					
H	9	14	21	17	23	33	29	71	37	11	88	41	52	32	76	98	85	47	57	25	

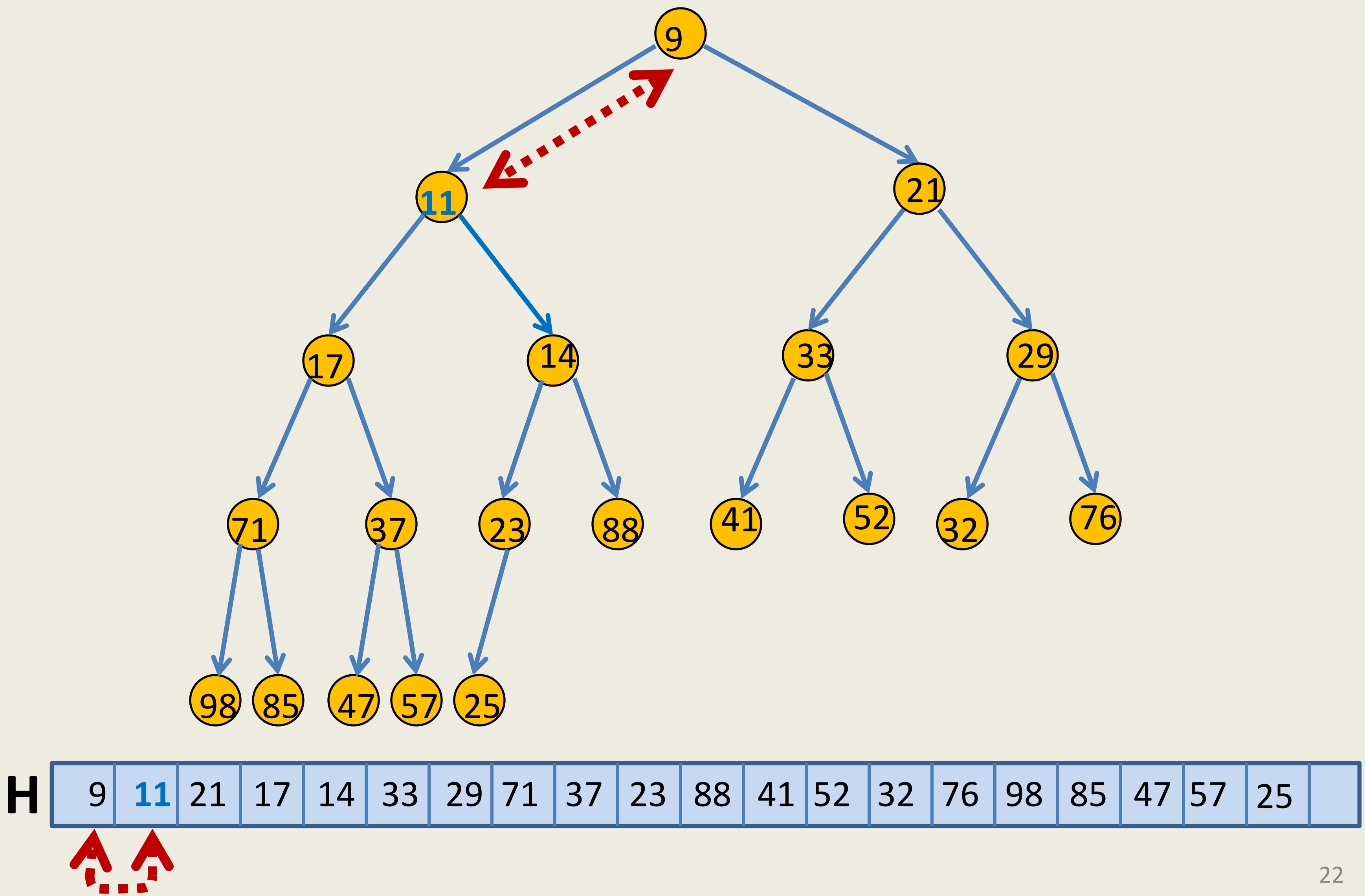
Insert(x,H)



H

9	14	21	17	**11**	33	29	71	37	23	88	41	52	32	76	98	85	47	57	25	

Insert(x,H)


9
11
21
17
14
33
29
71
37
23
88
41
52
32
76
98
85
47
57
25
H
9 11 21 17 14 33 29 71 37 23 88 41 52 32 76 98 85 47 57 25

Insert(x,H)

```
Insert(x,H)
{     i ← size(H);
      H(size) ←x;
      size(H) ← size(H) + 1;
      While(          i > 0          and  H(i) < H(⌊(i − 1)/2⌋)  )
      {
            H(i) ↔ H(⌊(i − 1)/2⌋);
            i  ←  ⌊(i − 1)/2⌋ ;
      }
}
```

Time complexity: O(log n)

The remaining operations on Binary heap

- **Decrease-key**($\boldsymbol{p}$, Δ, **H**): decrease the value of the key $\boldsymbol{p}$ by amount Δ**.**
  - Similar to **Insert(x,H).**
  - **O**(log n) **time**
  - Do it as an exercise

- **Merge**(**H1,H2**): Merge two heaps **H1 and H2**.
  - **O**(n) **time** where n = total number of elements in **H1** and **H2**

    (This is because of the array implementation)

Other heaps

Fibonacci heap : a **link** based data structure.

	Binary heap	Fibonacci heap
Find-min(H)	O(1)	O(1)
Insert(x,H)	O(log n)	O(1)
Extract-min(H)	O(log n)	O(log n)
Decrease-key(p, Δ, H)	O(log n)	O(1)
Merge(H_1,H_2)	O(n)	O(1)

Excited to study **Fibonaccci heap** ?
We shall study it during **CS345**.

Building a Binary heap

Building a Binary heap

Problem: Given **n** elements $\{x_0, \ldots, x_{n-1}\}$, build a binary **heap H** storing them.

Trivial solution:

(Building the Binary heap incrementally**)**

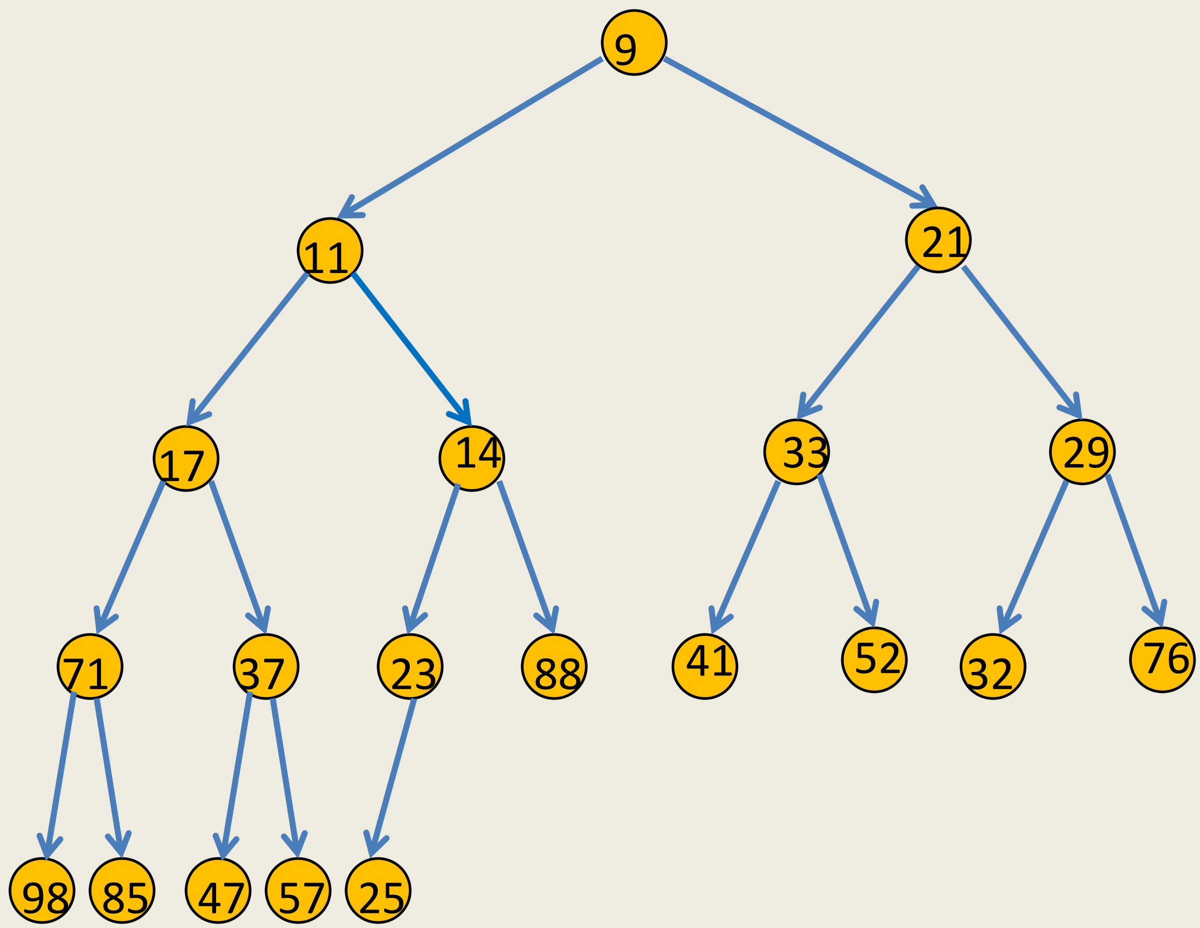
CreateHeap(**H**);

For($\boldsymbol{i}$ = 0 to $\boldsymbol{n-1}$ **)**

Insert(x_i,**H);**

What is the time complexity of this algorithm?

Building a Binary heap incrementally



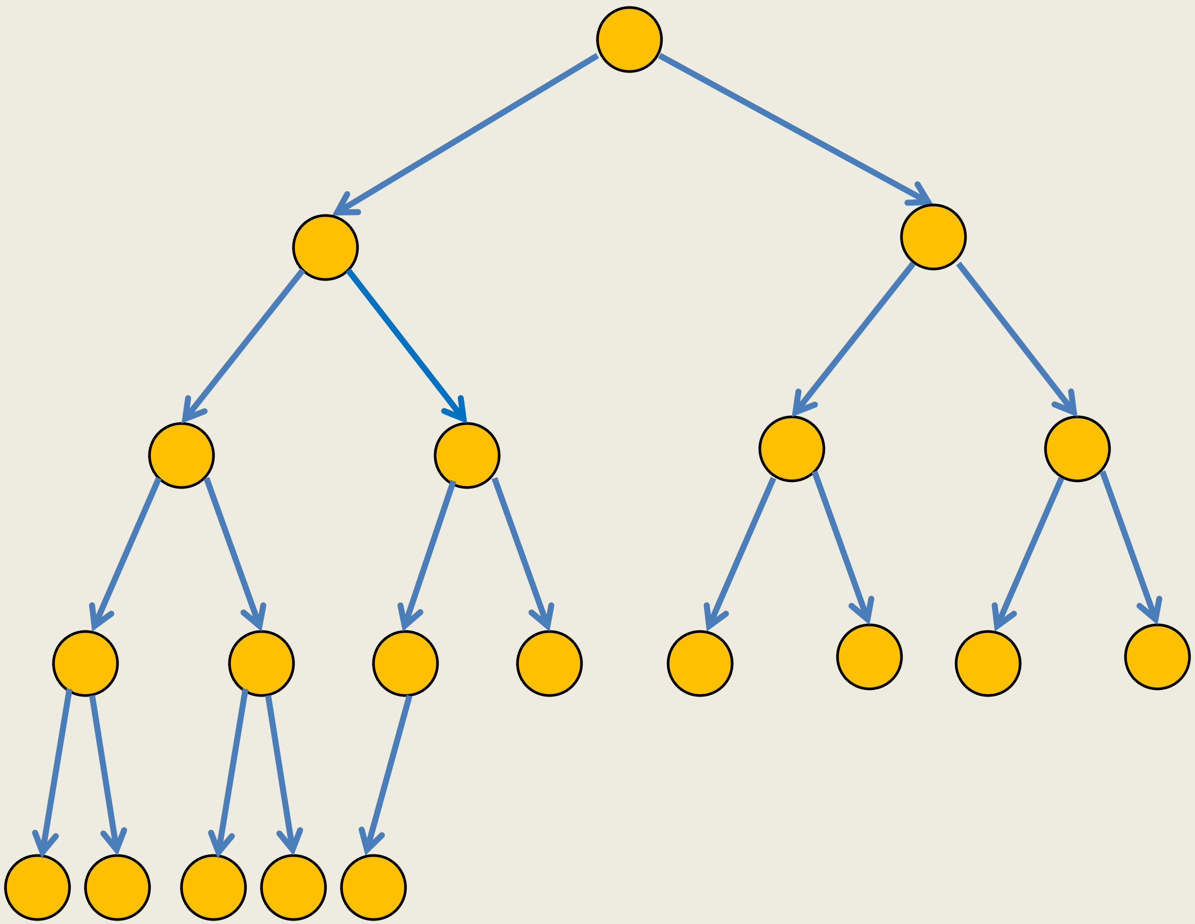
H																			
	9	11	21	17	14	33	29	71	37	23	88	41	52	32	76	98	85	47	57	25

Time complexity

Theorem: Time complexity of building a binary heap **incrementally** is **O**(n log n).

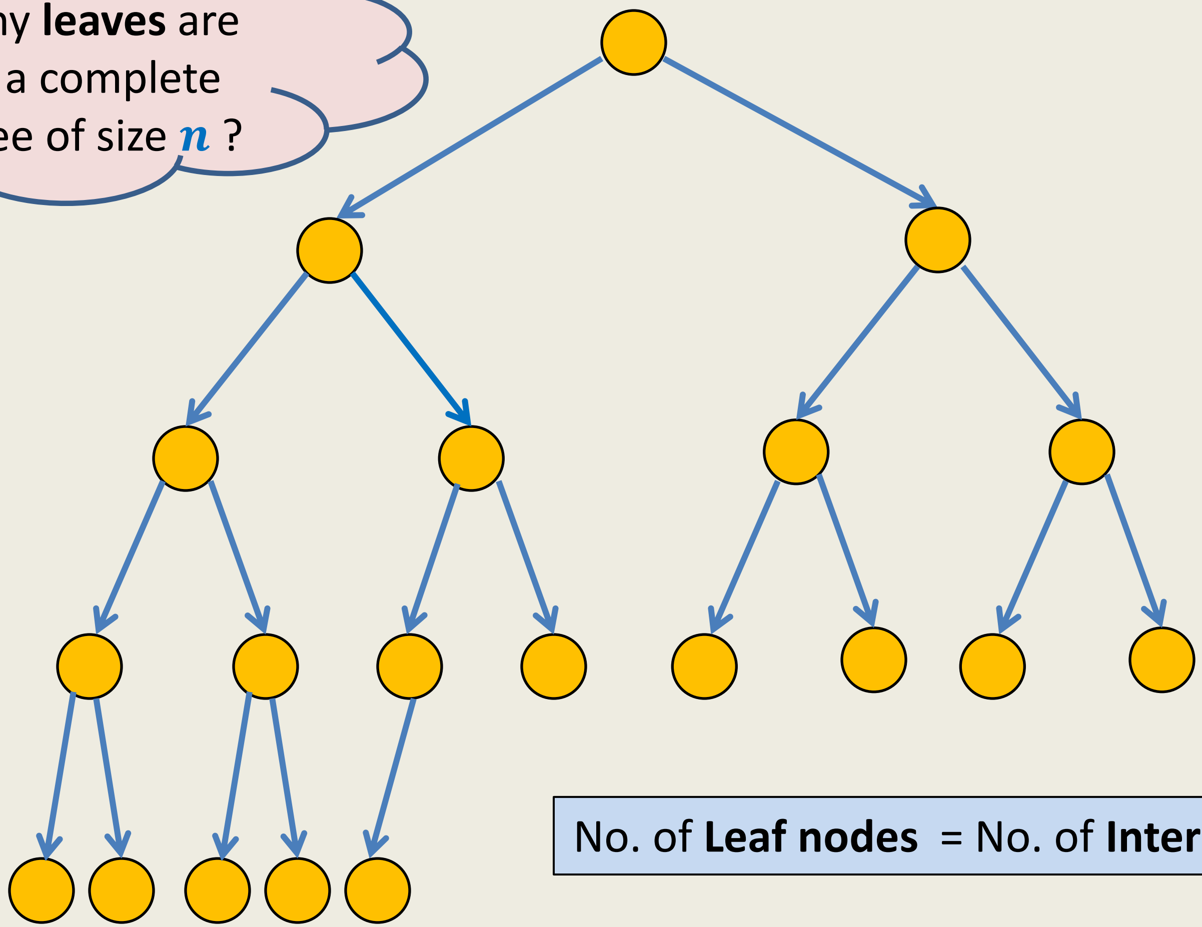
A binary heap can be built in **O**(n).

A complete binary tree



A complete binary tree

How many **leaves** are there in a complete Binary tree of size n ?

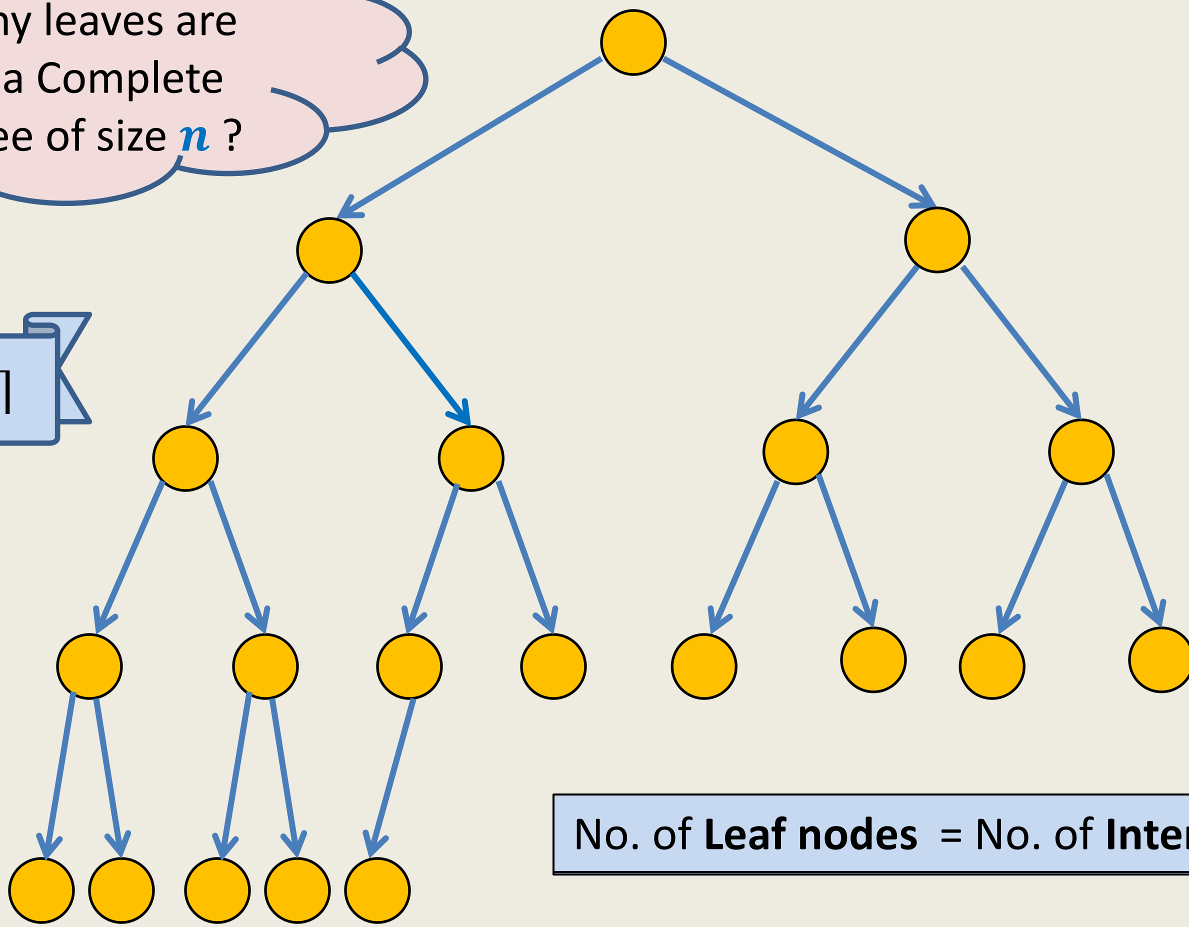


No. of **Leaf nodes** = No. of **Internal nodes** + **1**

A complete binary tree

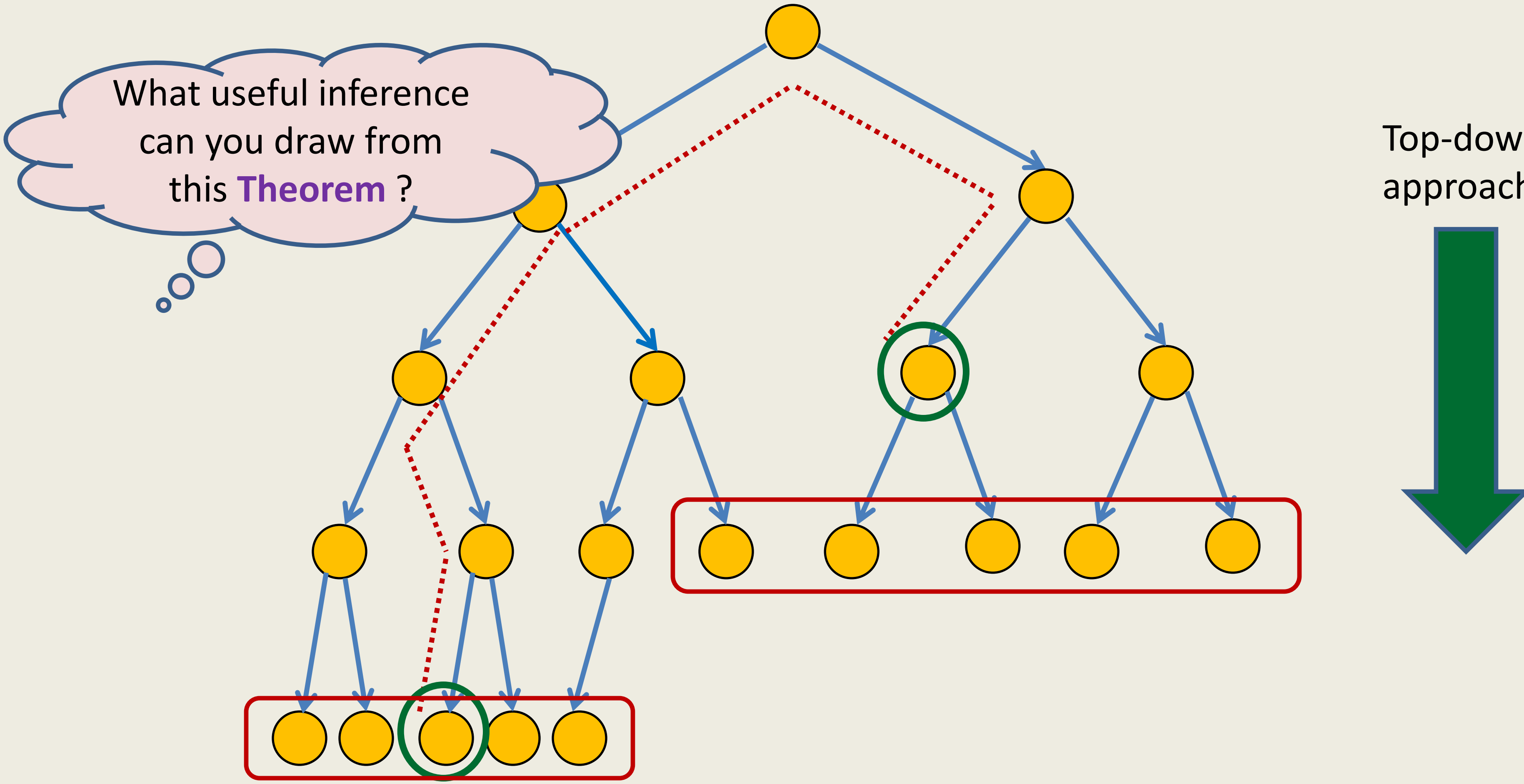
How many leaves are there in a Complete Binary tree of size n ?

$\lceil n/2 \rceil$



No. of **Leaf nodes** = No. of **Internal nodes** + 1

Building a Binary heap incrementally



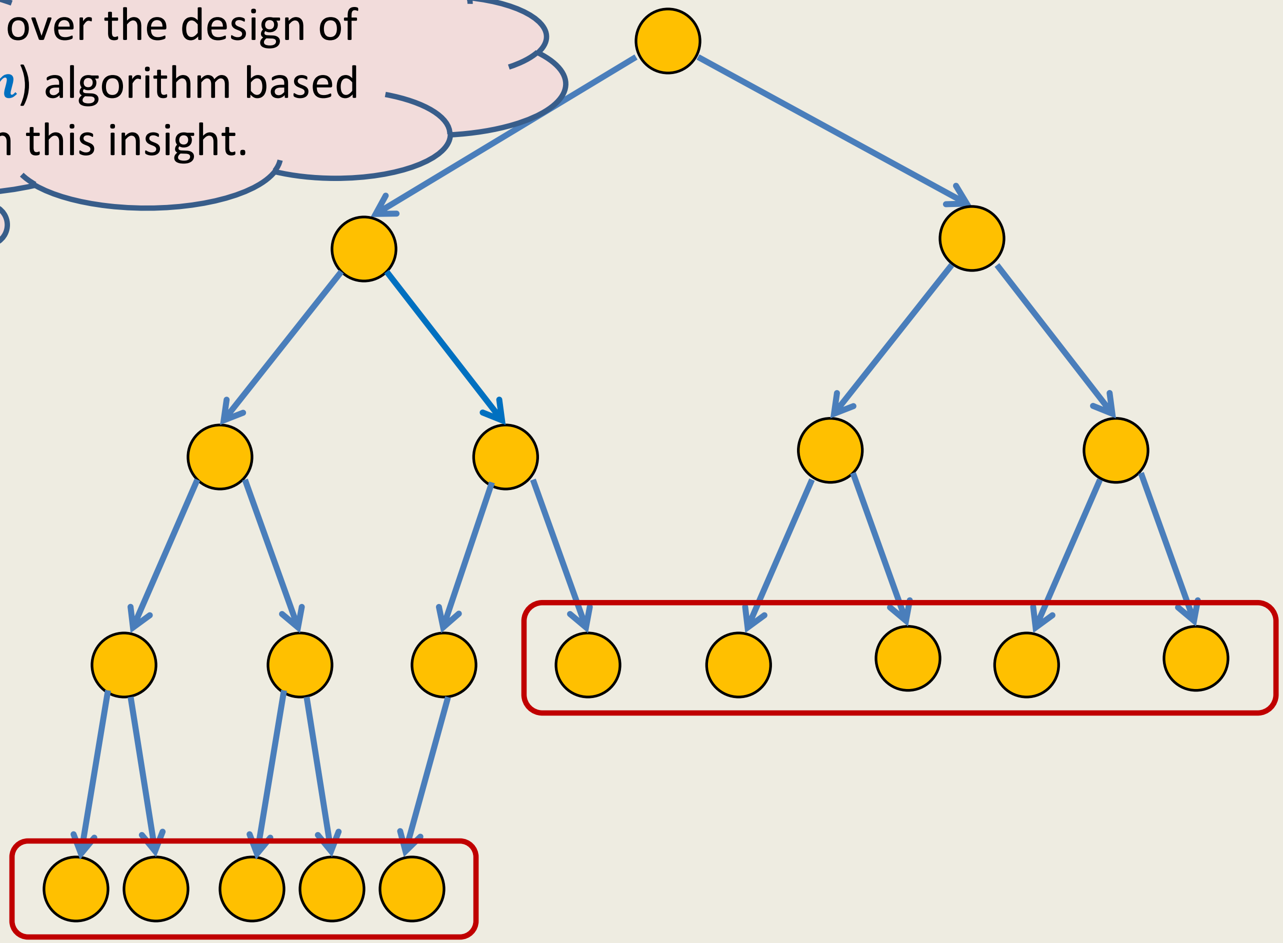
The time complexity for inserting a leaf node = **O**(log n)

leaf nodes = $\lceil n/2 \rceil$,

➔ **Theorem**: Time complexity of building a binary heap incrementally is **O**(n log n).

Building a Binary heap incrementally

Ponder over the design of the **O**(n) algorithm based on this insight.



Top-down approach

The **O**(n) time algorithm must take **O**(1) time for each of the $\lceil n/2 \rceil$ leaves.