Data Structures and Algorithms

(**CS210A**)

Semester I – **2014-15**

Lecture 28:

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

Data Structures

Range of efficient functions

Lists: (arrays, linked lists)

Simplicity

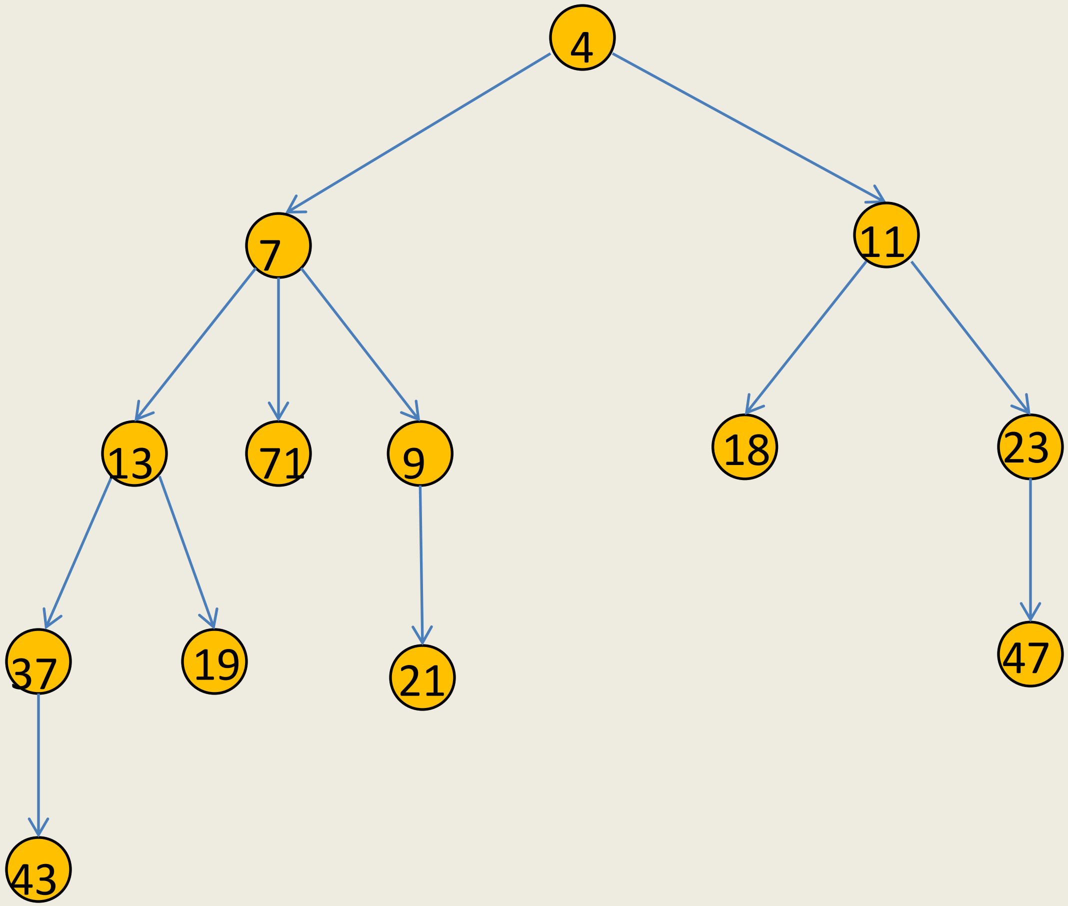
Binary Heap

Binary Search Trees

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**(**H1,H2**) : Merge two heaps **H1** and **H2**.

Why heaps when we can use a binary search tree ?

Compared to binary search trees, a heap is usually

-- much **simpler** and

-- more **efficient**

Existing heap data structures

- Binary heap
- Binomial heap
- Fibonacci heap
- Soft heap

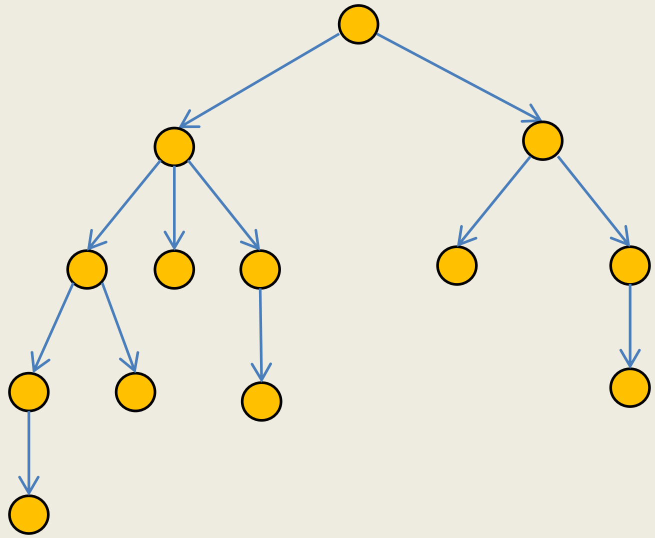
Can we implement a binary tree using an array ?

Yes.

In some special cases

fundamental question

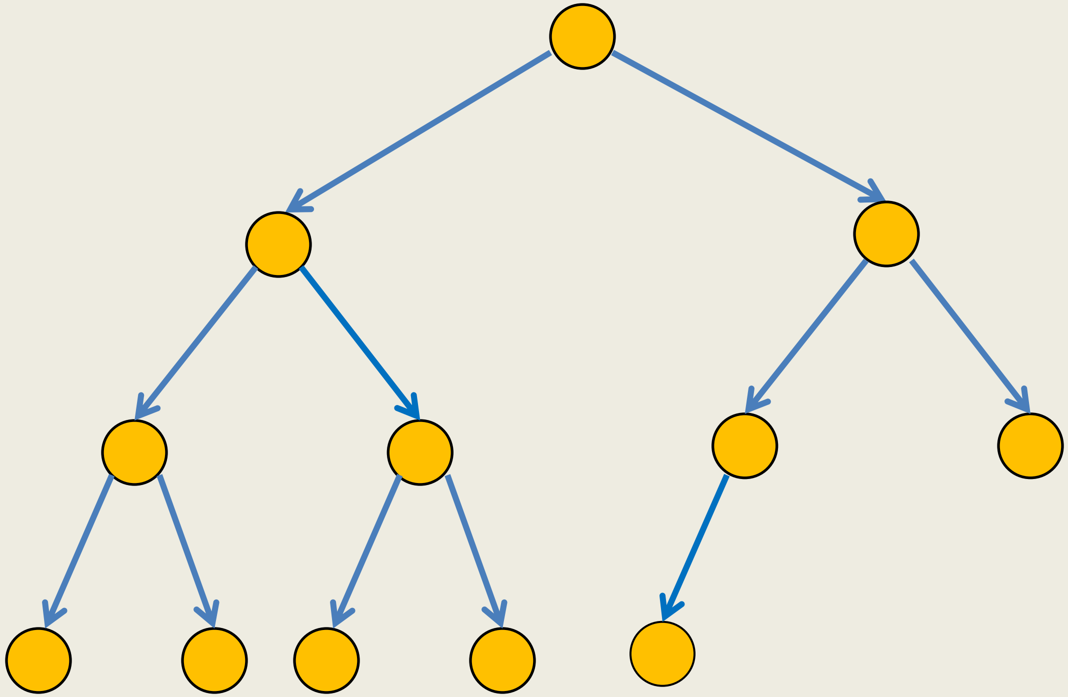
Question: What does the implementation of a tree data structure require ?



Answer: a mechanism to

- access **parent** of a node
- access **children** of a node.

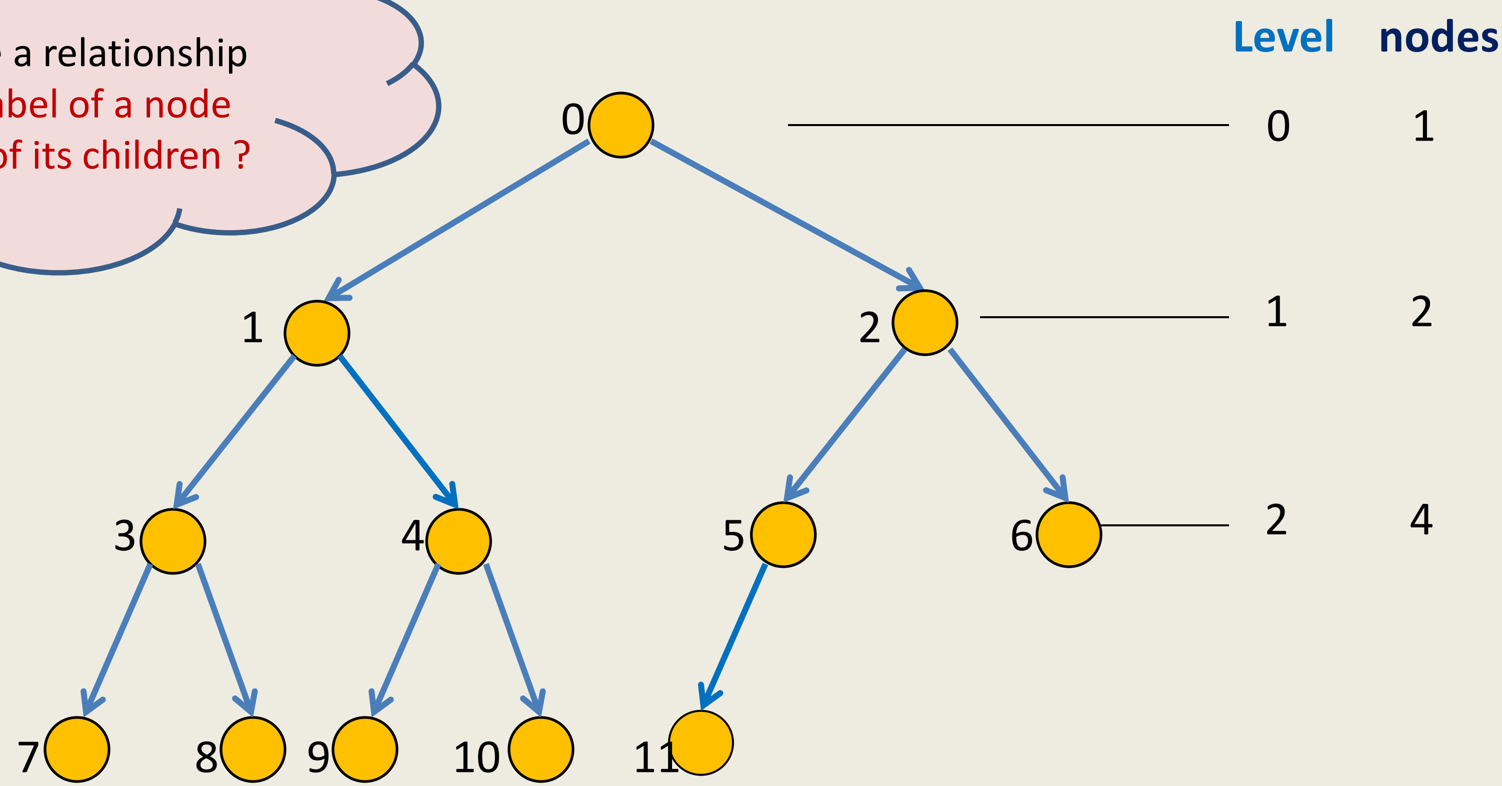
A complete binary tree



A complete binary of 12 nodes.

A **complete** binary tree

Can you see a relationship between label of a node and labels of its children ?



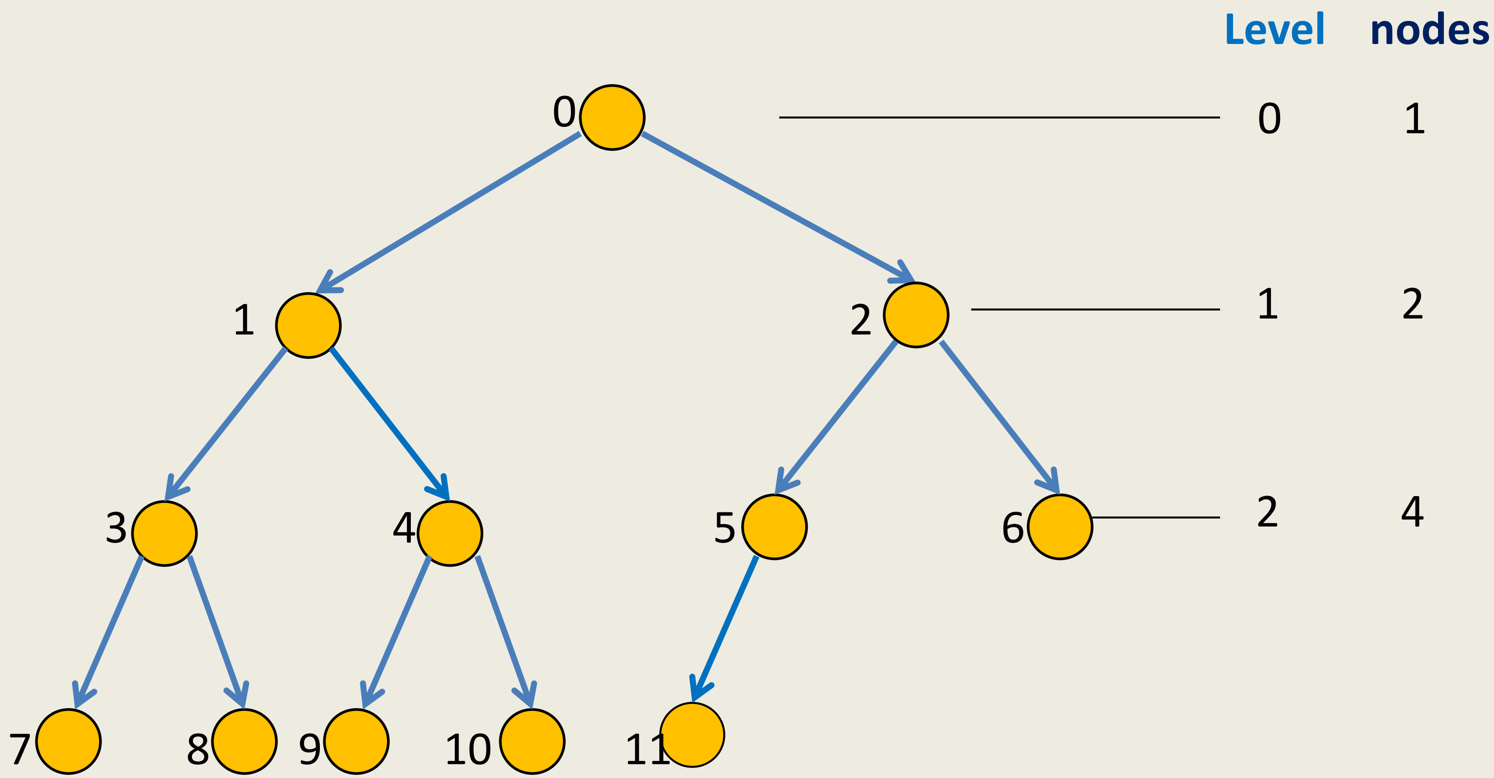
The label of the **leftmost node** at level i = $2^i - 1$

The label of a **node v occurring at** k**th place from left** at level i = $2^i + k - 2$

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)** = $2j+2$

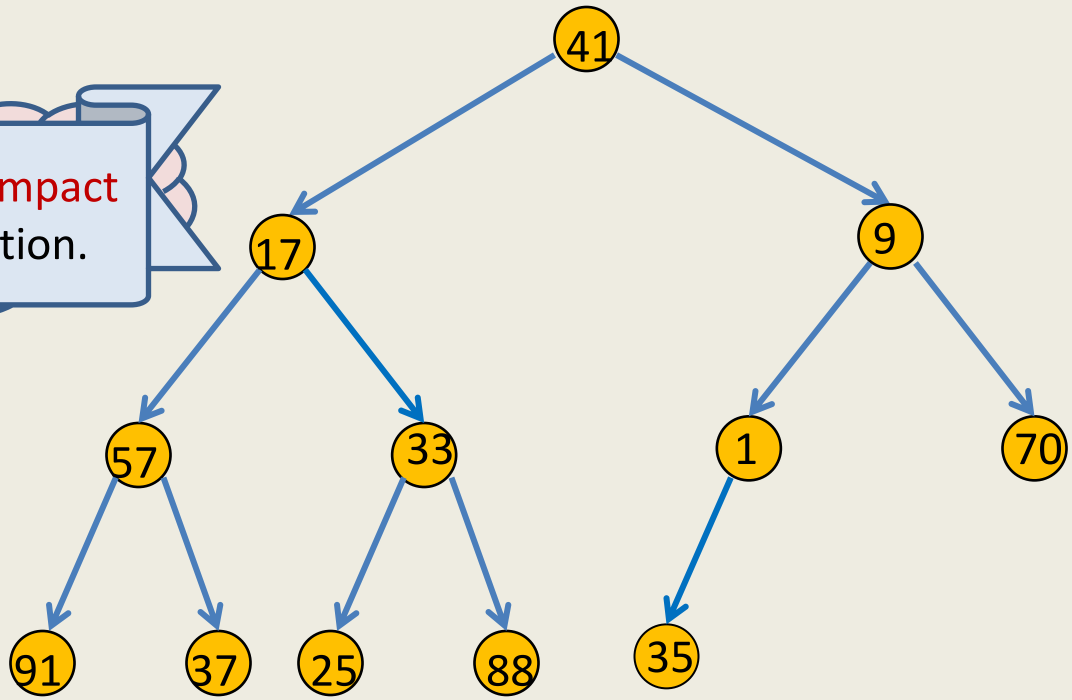
Label of **parent(v)** = $\lfloor (j-1)/2 \rfloor$

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.

The most compact representation.

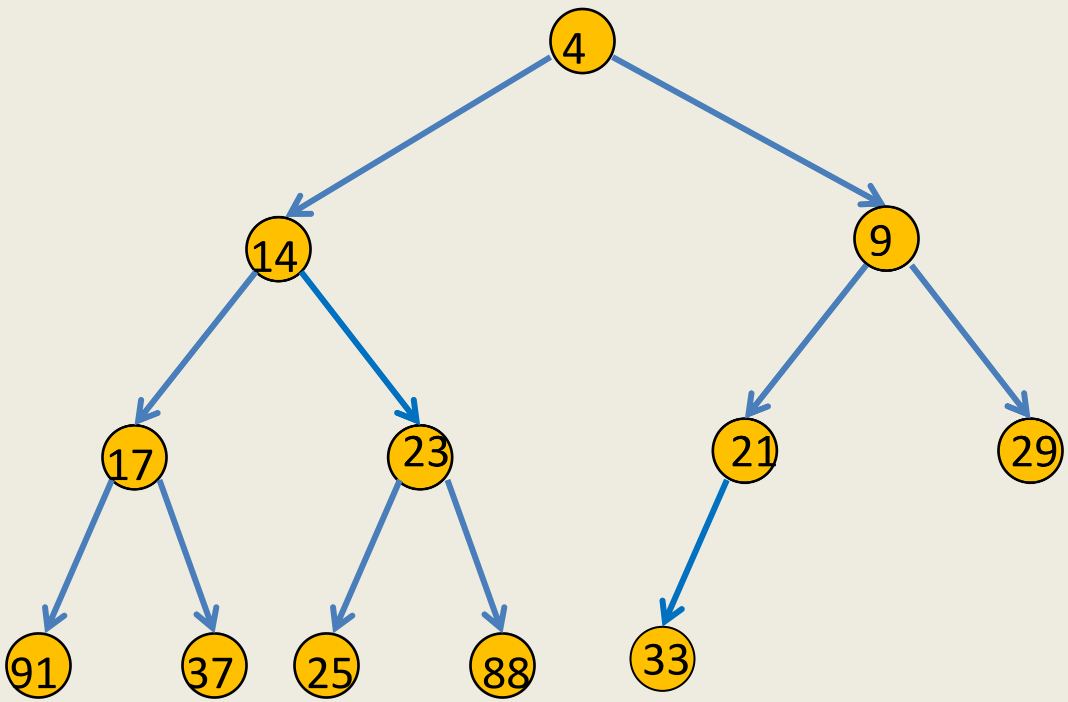


41	17	9	57	33	1	70	91	37	25	88	35

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

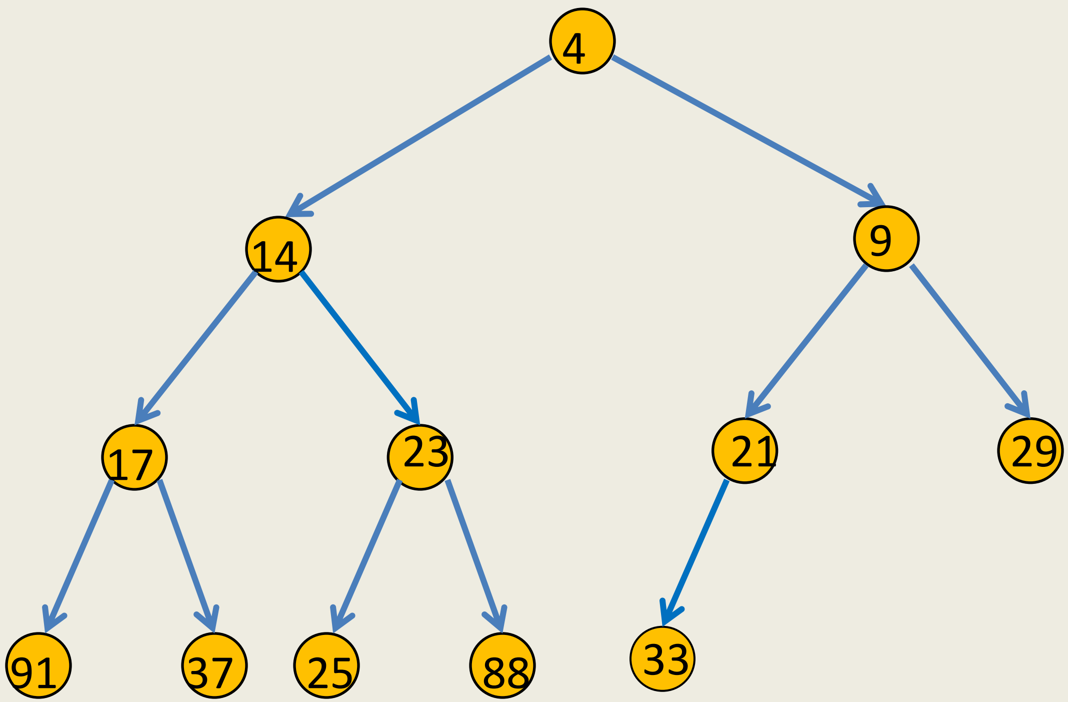
Let n : the maximum number of keys at any moment of time,

then we keep

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

Find_min(H)

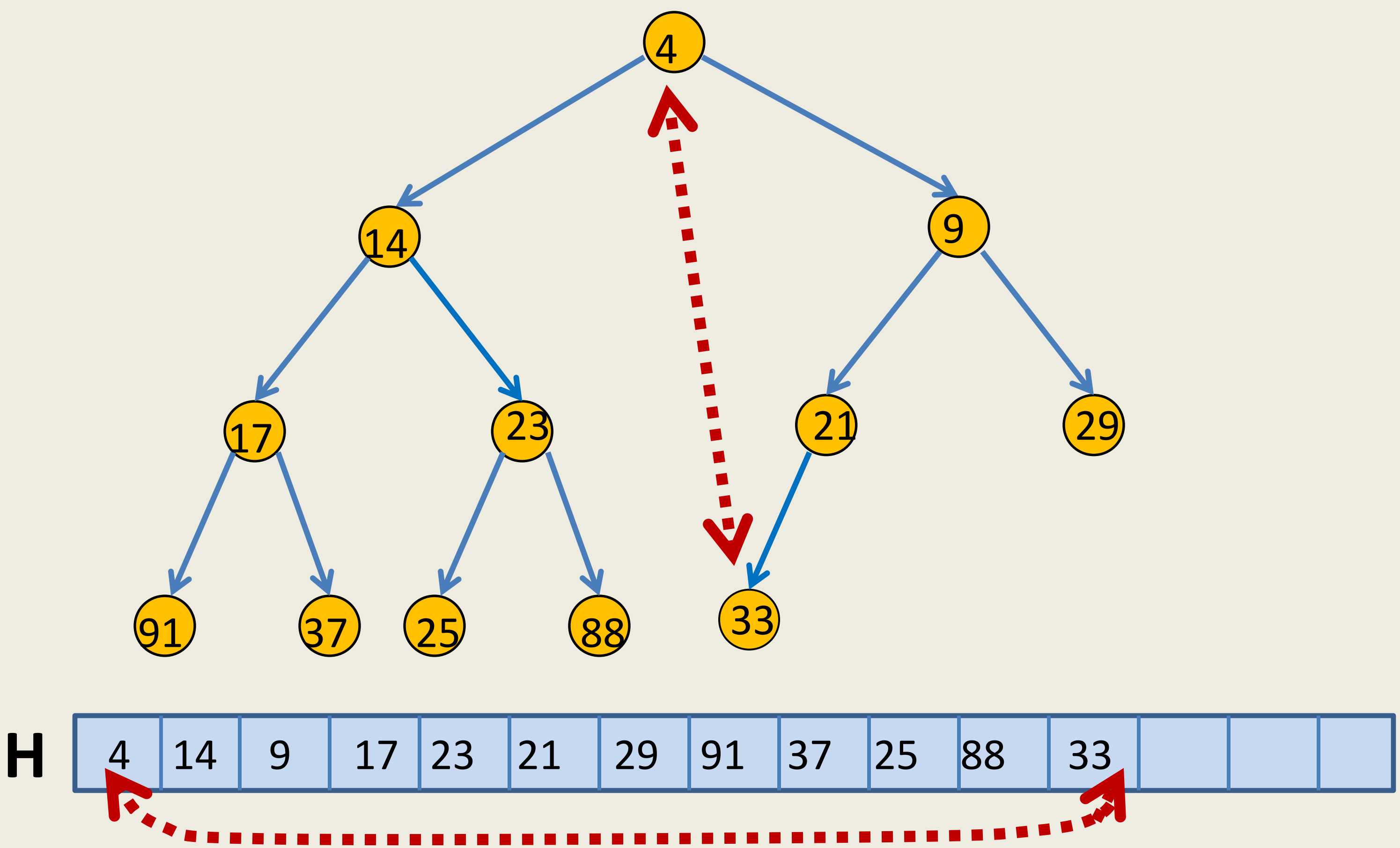
Report **H**[0].



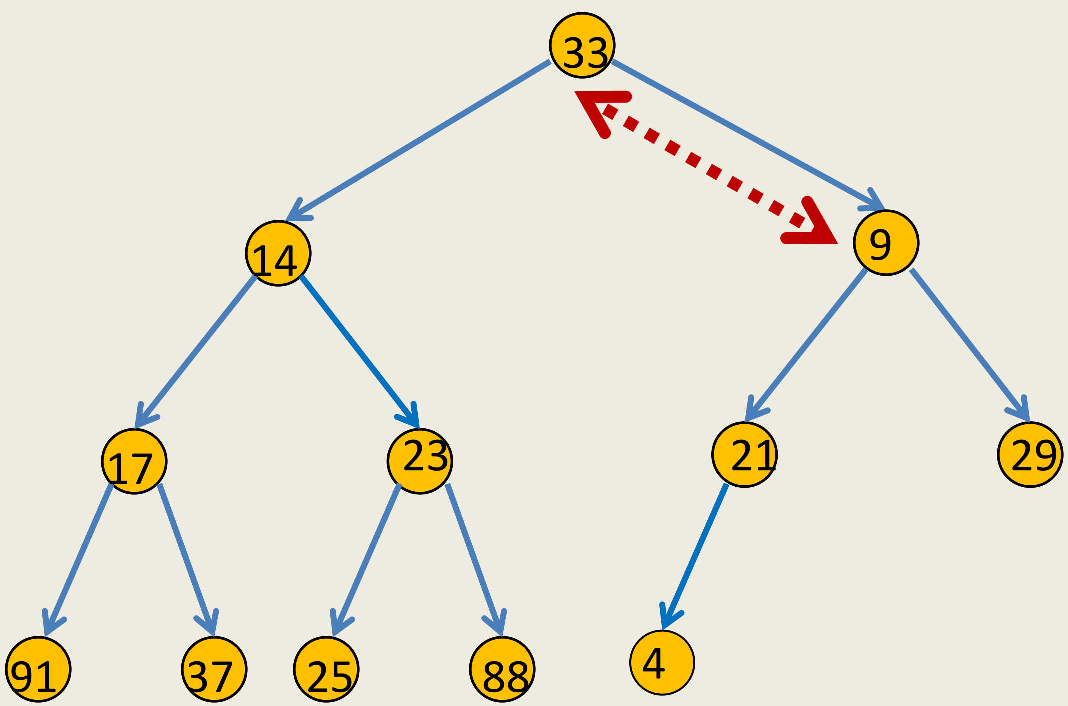
H

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

Extract_min(H)

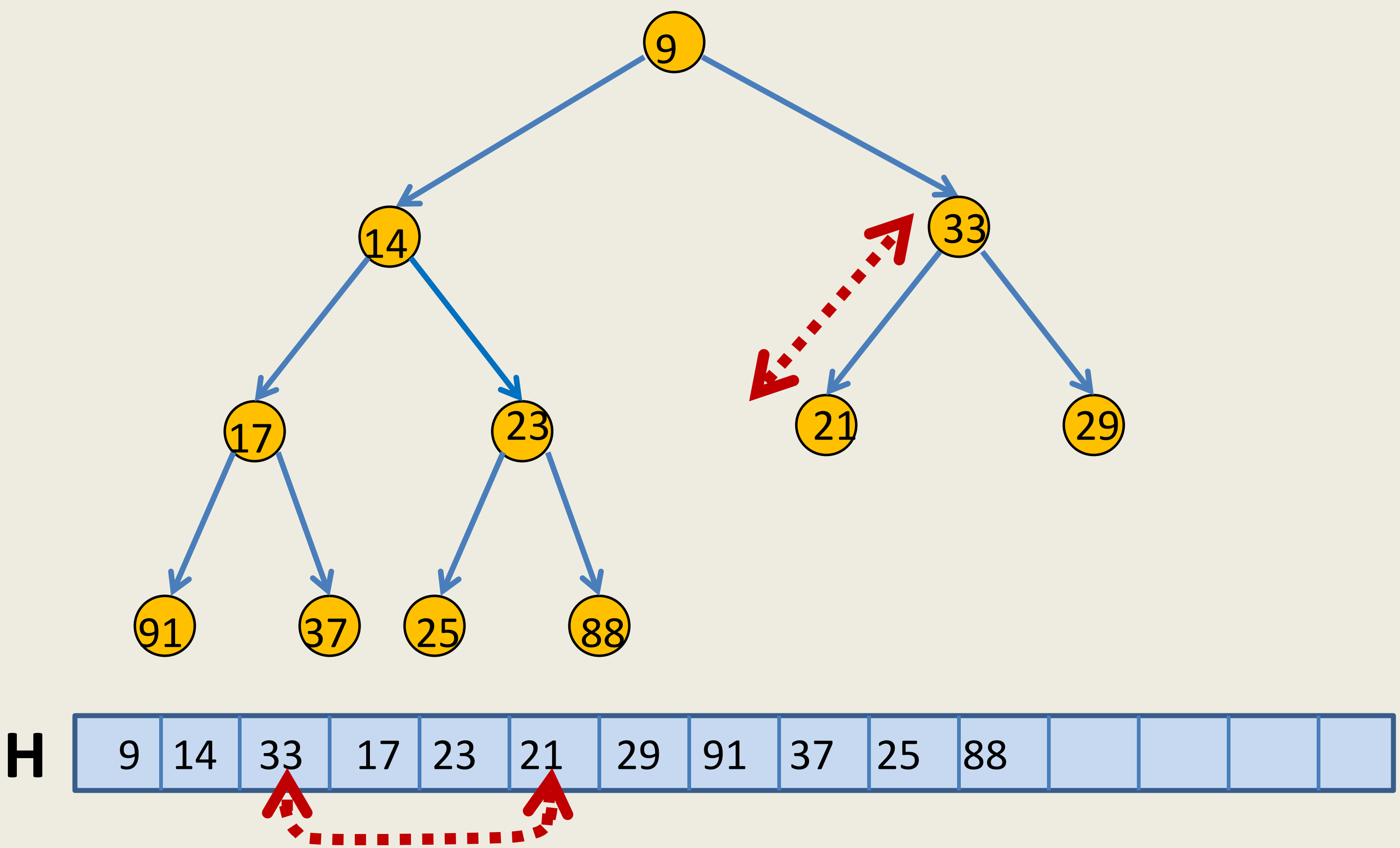


Extract_min(H)



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

Extract_min(H)

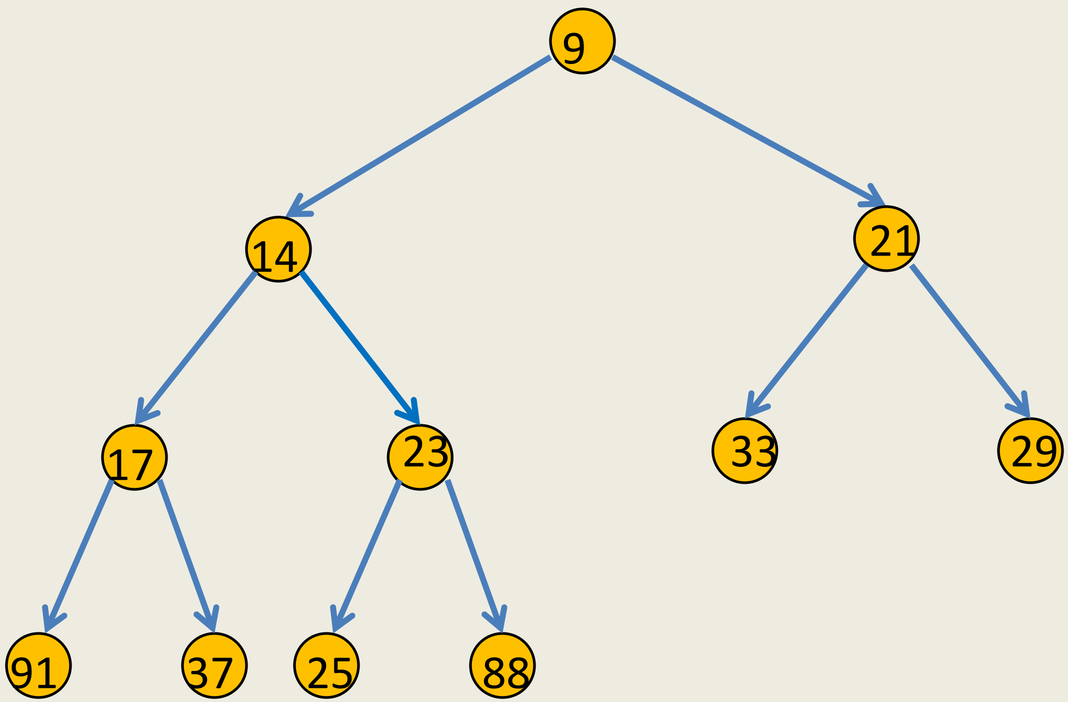


Extract_min(H)

We are done.

The no. of operations performed is of the order of no. of levels in binary heap.

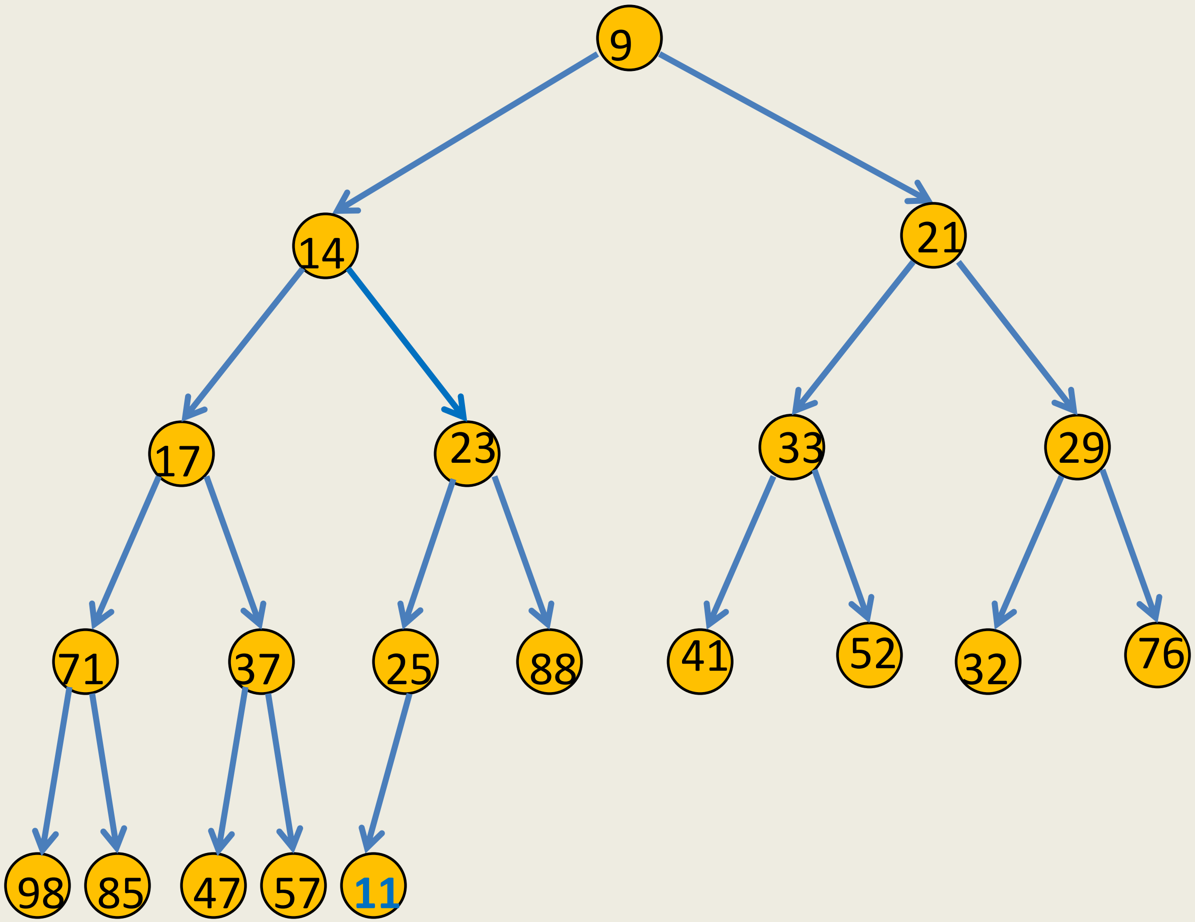
= **O**(log n)   …show it as an exercise.



H

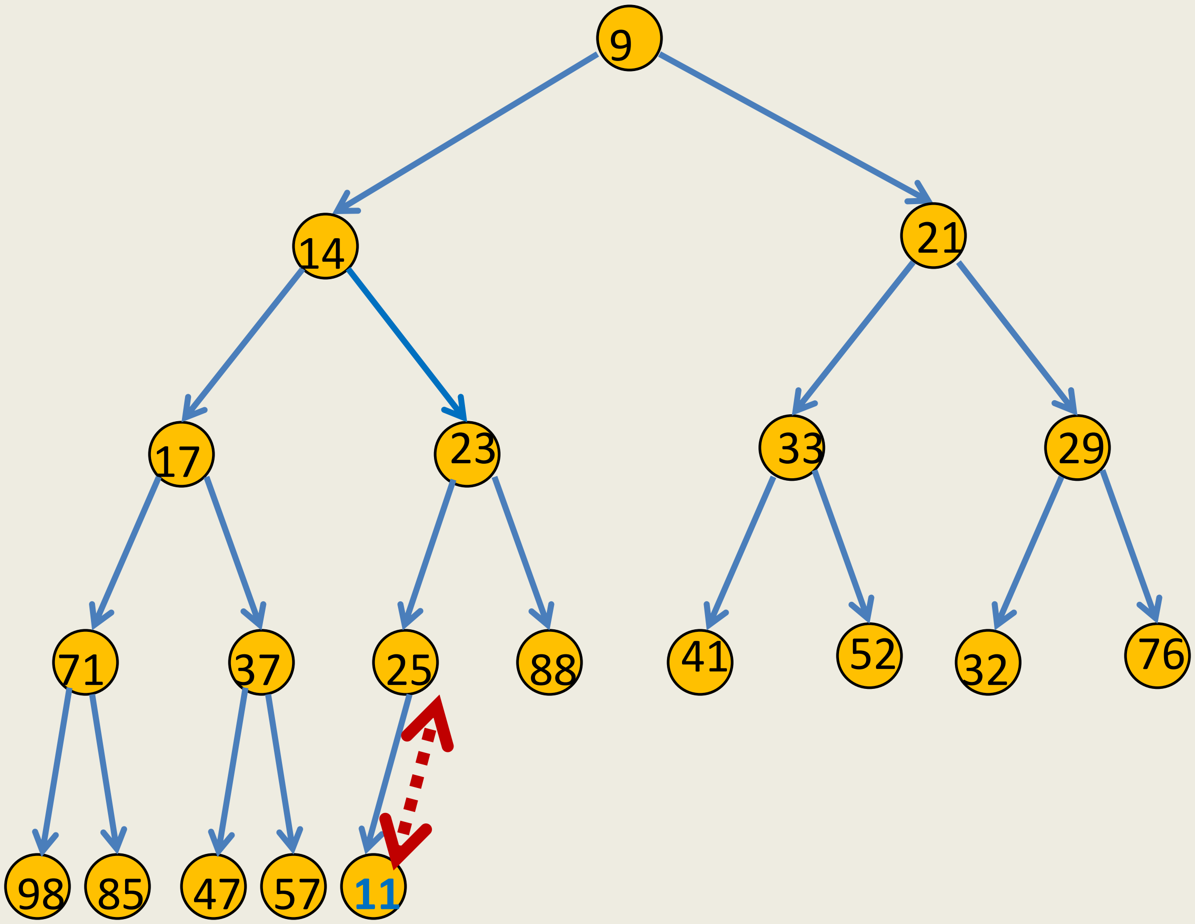
9	14	21	17	23	33	29	91	37	25	88				

Insert(x,H)



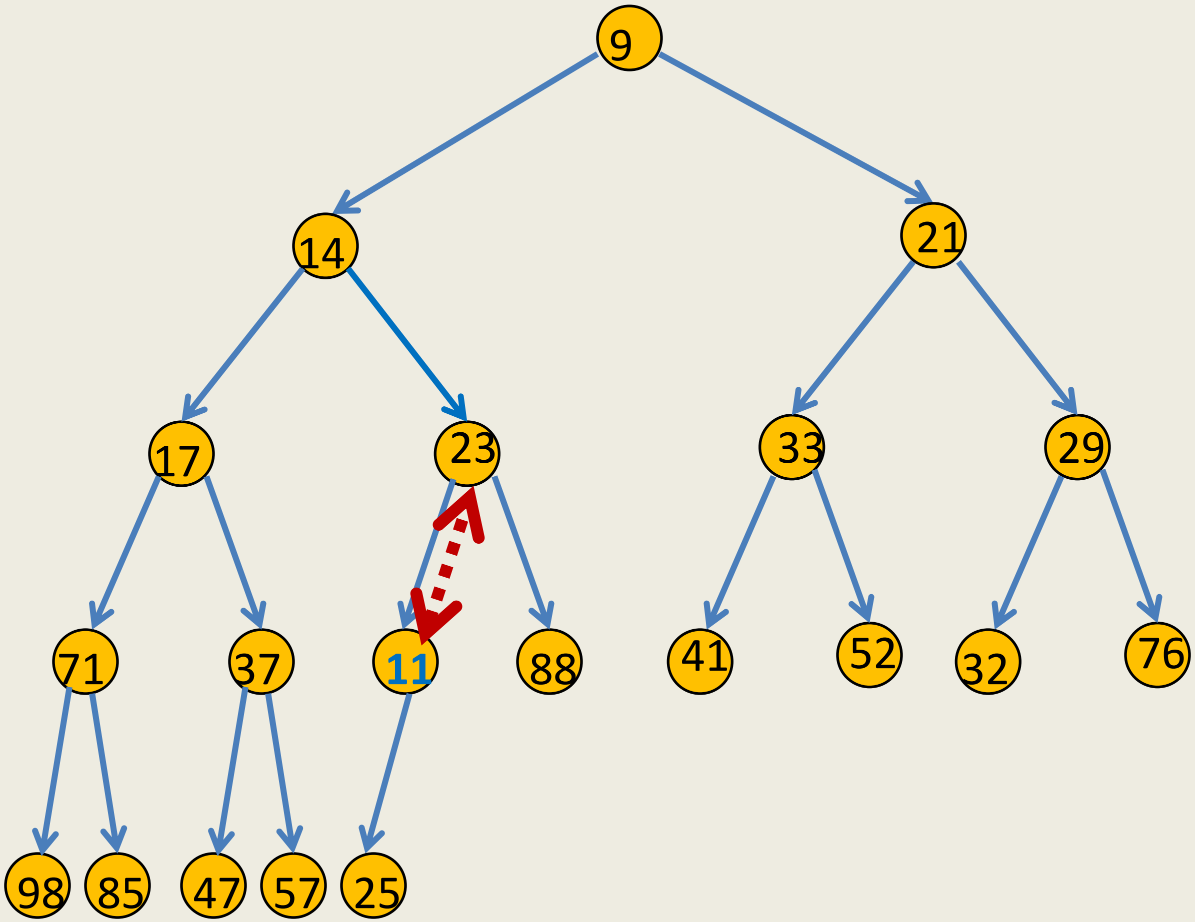
H	9	14	21	17	23	33	29	71	37	25	88	41	52	32	76	98	85	47	57	11	

Insert(x,H)



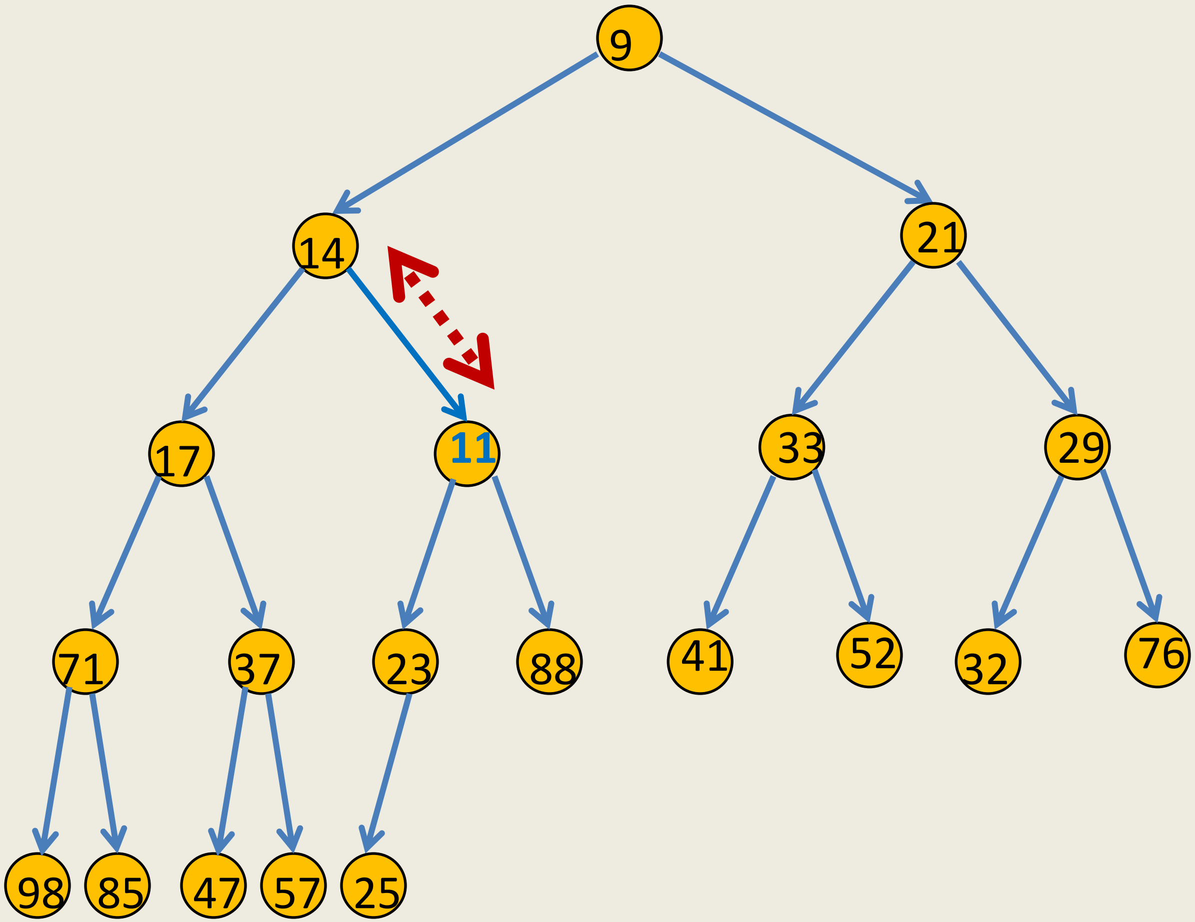
H																				
	9	14	21	17	23	33	29	71	37	25	88	41	52	32	76	98	85	47	57	11

Insert(x,H)



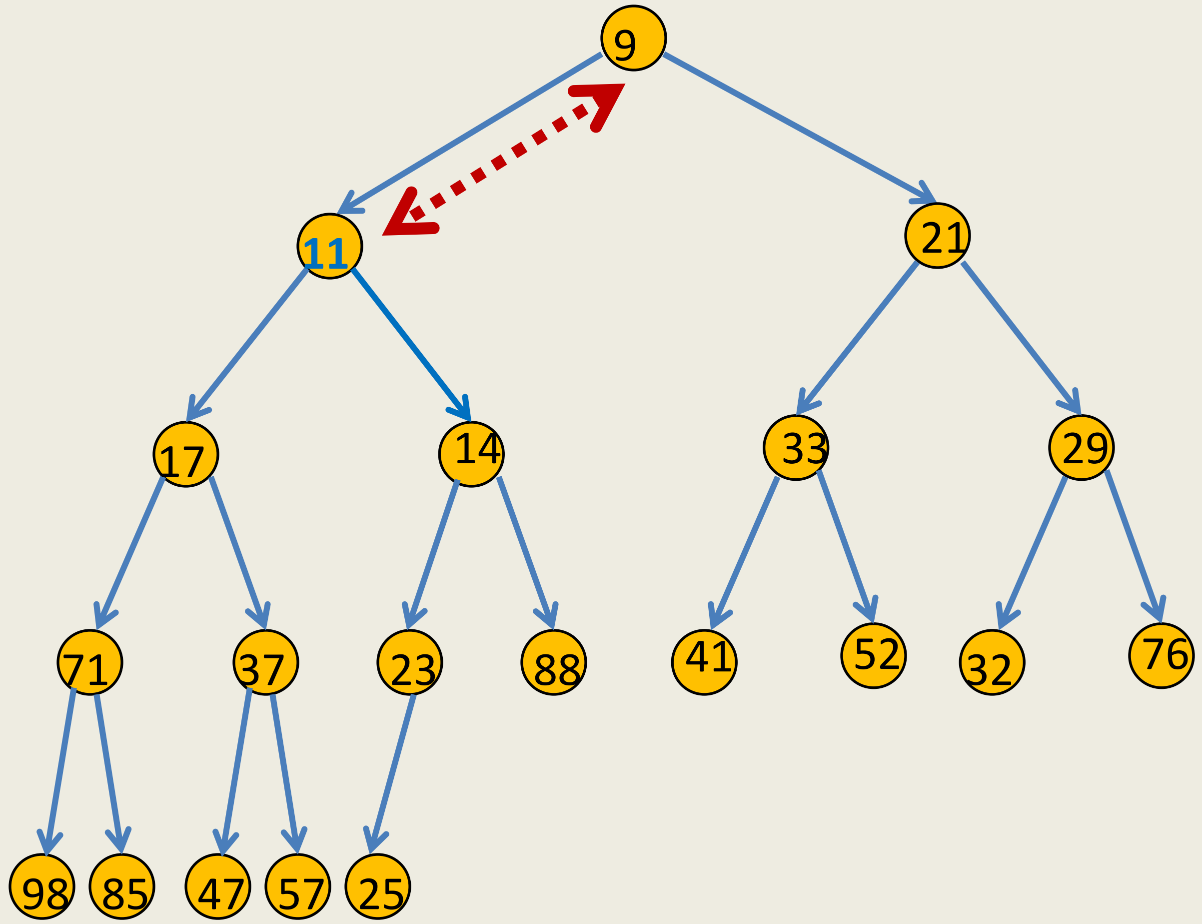
H																				
9	14	21	17	23	33	29	71	37	11	88	41	52	32	76	98	85	47	57	25	

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)



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(i) ← 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(H1,H2)	**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 $\boldsymbol{n}$ elements $\{x_0, \ldots, x_{n-1}\}$, build a binary **heap H** storing them.

Trivial solution:

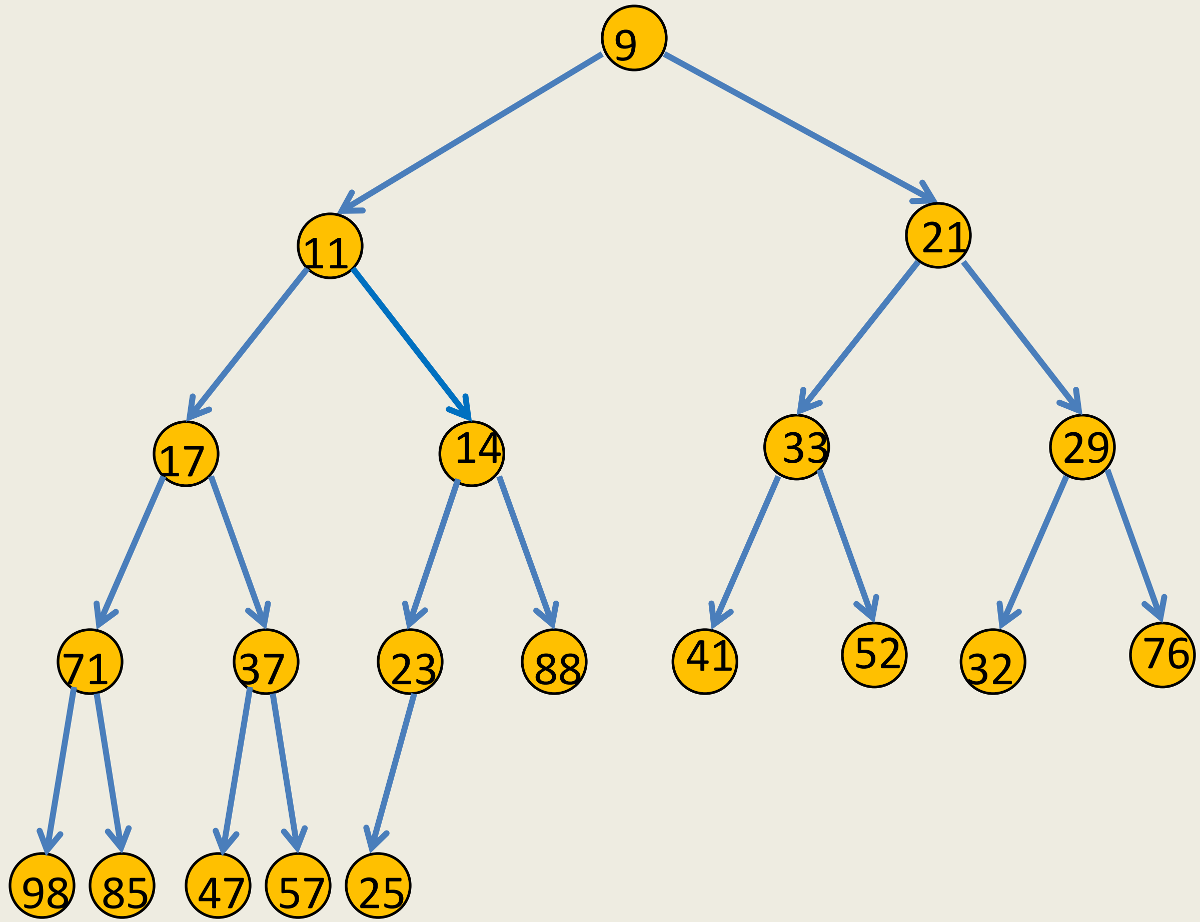
(Building the Binary heap incrementally)

CreateHeap(**H**);

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

Insert(x_i**,H);**

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 of incrementally building a Binary heap

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

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

Those who love algorithm should ponder over it☺