



Design and Analysis
of Algorithms I

Data Structures

Heaps: Some
Implementation Details

Heap: Supported Operations

- A container for objects that have keys
- Employer records, network edges, events, etc.

Insert: add a new object to a heap.

Running time : $O(\log(n))$

Equally well,
EXTRACT MAX



Extract-Min: remove an object in heap with a minimum key value. [ties broken arbitrarily]

Running time : $O(\log n)$ [n = # of objects in heap]

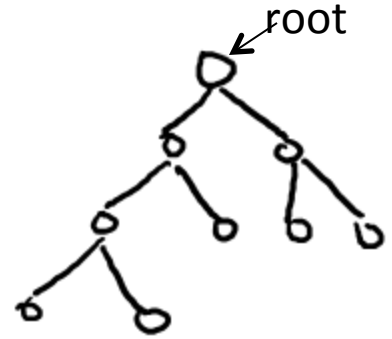
Also : **HEAPIFY** ($\begin{matrix} n \text{ batched Inserts} \\ \text{in } O(n) \text{ time} \end{matrix}$), **DELETE** ($O(\log(n))$ time)

The Heap Property

Conceptually : think of a heap as a tree.
-rooted, binary, as complete as possible

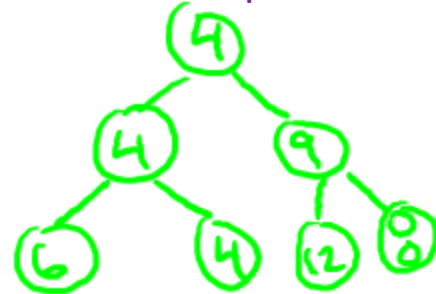
Heap Property: at every node x ,
 $\text{Key}[x] \leq \text{all keys of } x\text{'s children}$

Consequence : object at root must
have minimum key value

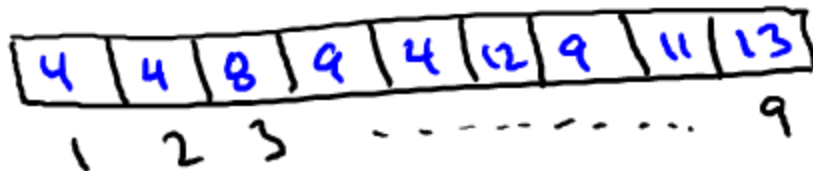


A heap

alternatively



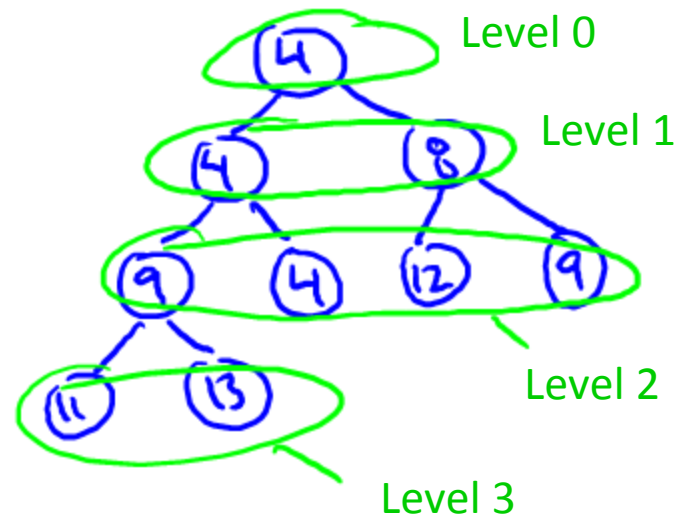
Array Implementation



Note : $\text{parent}(i) = i/2$ if i even
 $= \lfloor i/2 \rfloor$ if i odd

i.e., round down

and children of i are $2i, 2i+1$

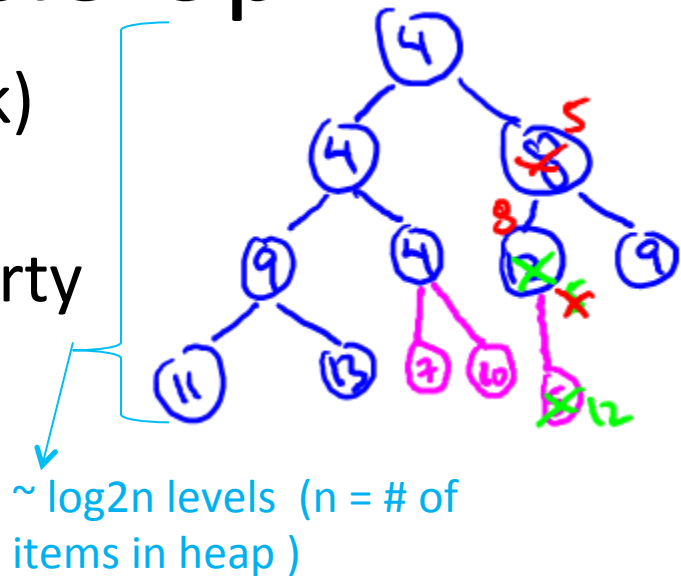


Insert and Bubble-Up

Implementation of Insert (given key k)

Step 1: stick k at end of last level.

Step 2 : Bubble-Up k until heap property is restored (i.e., key of k 's parent is $\leq k$)



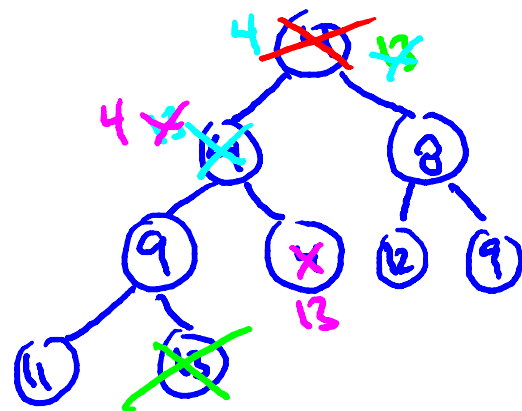
Check : 1.) bubbling up process must stop, with heap property restored
2.) runtime = $O(\log(n))$

Extract-Min and Bubble-Down

Implementation of Extract-Min

1. Delete root
2. Move last leaf to be new root.
3. Iteratively Bubble-Down until heap property has been restored

[always swap with smaller child!]



Check : 1.) only Bubble-Down once per level, halt with a heap
2.) run time = $O(\log(n))$