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: $O(\log n)$

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]

(equally well, EXTRACT MAX)

Also: HEAPIFY (n batched Inserts in $O(n)$ time), DELETE ($O(\log n)$ time)

The Heap Property

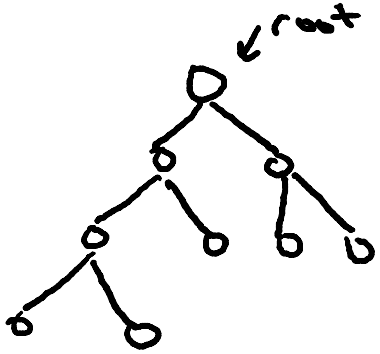
Conceptually: think of a heap as a tree.

- rooted, binary, as complete as possible

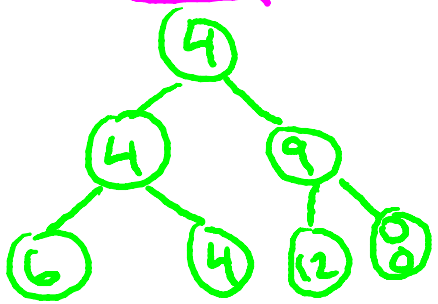
Heap property: at every node x,

$Key[x] \leq$ all keys of x's children

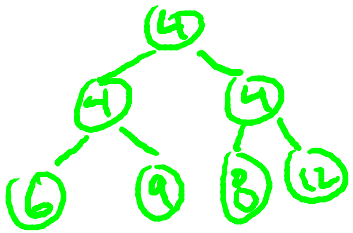
Consequence: object at root must have minimum key value.



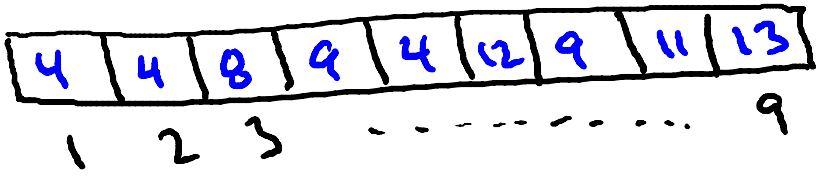
a heap

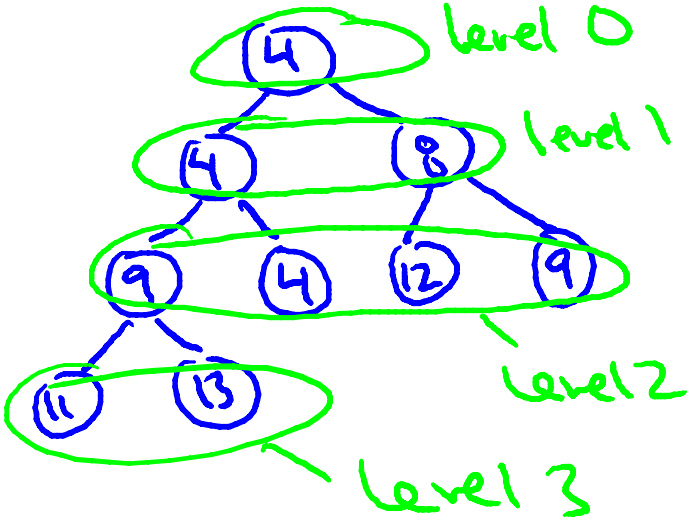


alternatively



Array Implementation





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

↖ i.e., round down

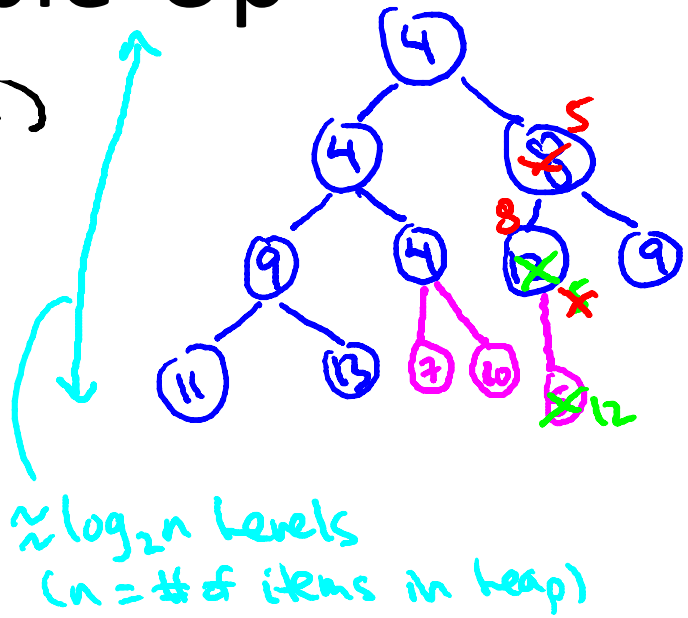
and children of i are $2i, 2i+1$

Insert and Bubble-Up

Implementation of Insert (given key k)

Step1: stick k at end of last level.

Step2: Bubble-Up k until heap property is restored (i.e., key of k's parent is $\leq k$).

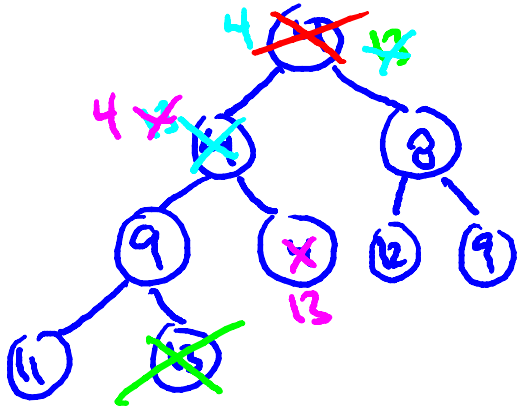


Check: (1) bubbling up process must stop, with heap property restored.

(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 = $O(\log n)$