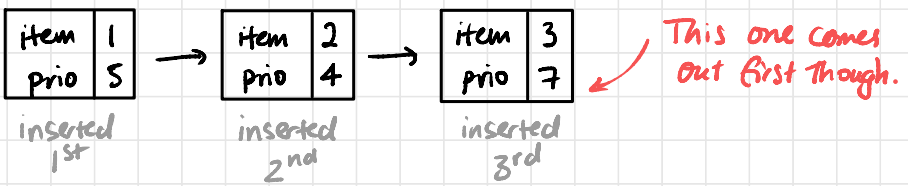
TUTORIAL 10.

HEAPS AND PRIORITY QUEUES.

Priority queue: A normal queue, except each item has a priority value associated with it. Higher priority items leave the queue before lower priority ones, even if they joined it late.



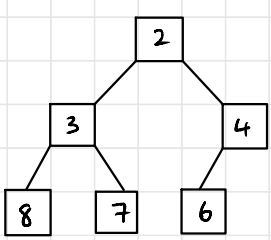
Implementing one can be done a few ways, but the best is via:

Heap: a special kind of (binary) tree which is ordered top → bottom.

There are 2 main kinds we deal with in this course:

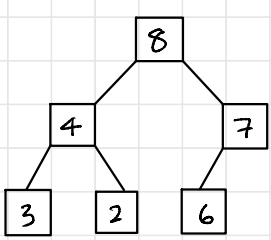
Min heap

Each node is smaller than its children



Max heap

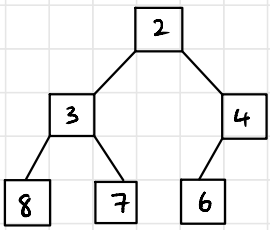
Each node is larger than its children



Note: children are not ordered horizontally!

Valid heaps must also satisfy completeness: each level of the tree must be filled except the bottom level, and this bottom row must be filled out left to right.

Implementation: Completeness ⟹ nice array representation for a heap is possible:



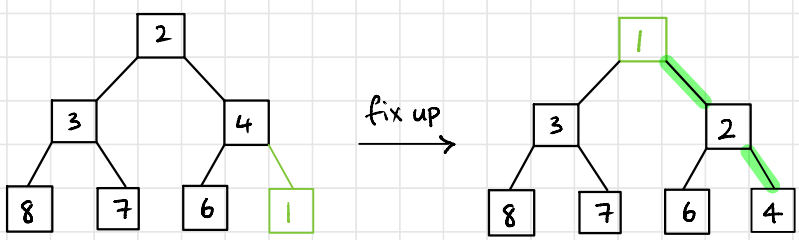
Node @ index k has left child @ 2k, right @ 2k+1 (and parent @ $\lfloor k/2 \rfloor$)

0	1	2	3	4	5	6
X	2	3	4	8	7	6

Don't use index 0

Insertion: Given a value v,

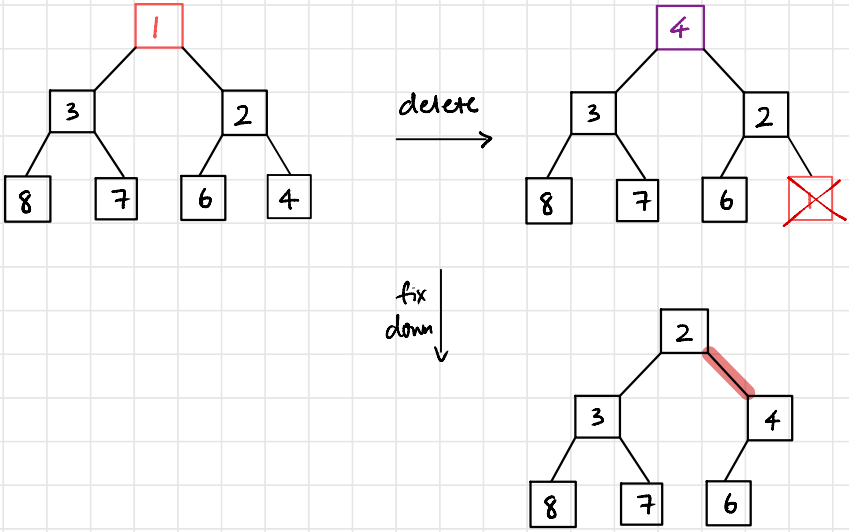
1. Insert v into the leftmost free slot in the heap
2. **Fix up:** compare v with its parent and swap if the heap property is not satisfied; repeat until you either find a parent in the right place or you hit the root.



Deletion: Always delete the root. Assuming a min heap,

1. Swap the root and lowest, rightmost value in the heap
2. Delete the root value now
3. **Fix down:** compare the new root with its smaller child, and swap them if the root is bigger. Repeat until the child is the bigger one or you hit the bottom of the heap.

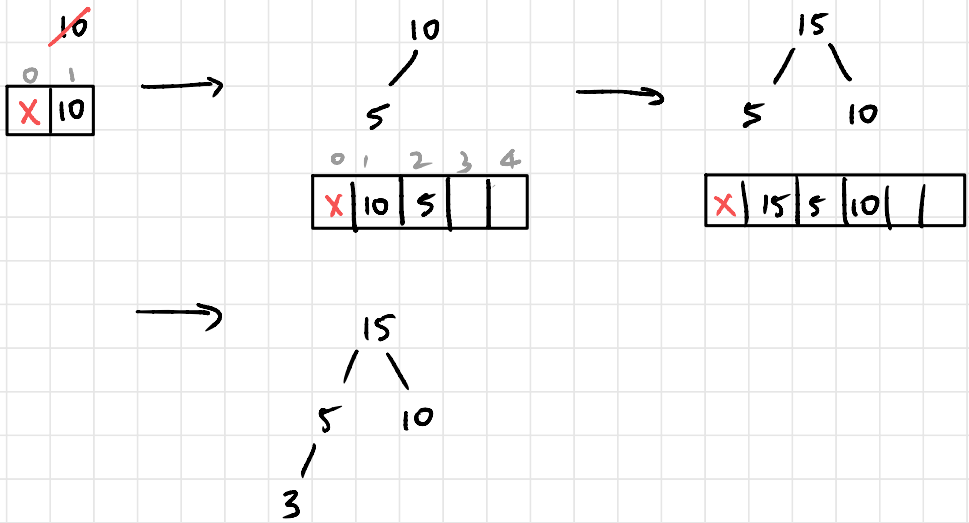
(For a max heap: interchange bigger ⟷ smaller)



Complexity: Complete trees have O(log n) height ⇒ both operations are O(log n)!

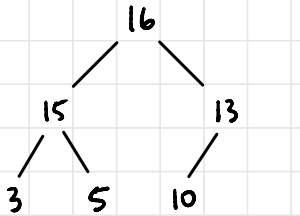
2. (Reduced)

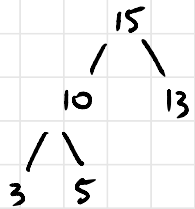
Insert ~~10~~ 5 15 3 16



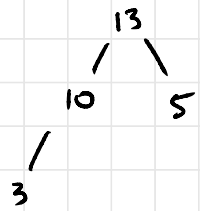
Now delete twice:

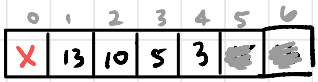
1st





2nd





TRIES

Like assignment 1!

Motivation: Text problems are very common. We have seen many data structures which could be used for good time complexity, but there are other considerations too: space is one of them.

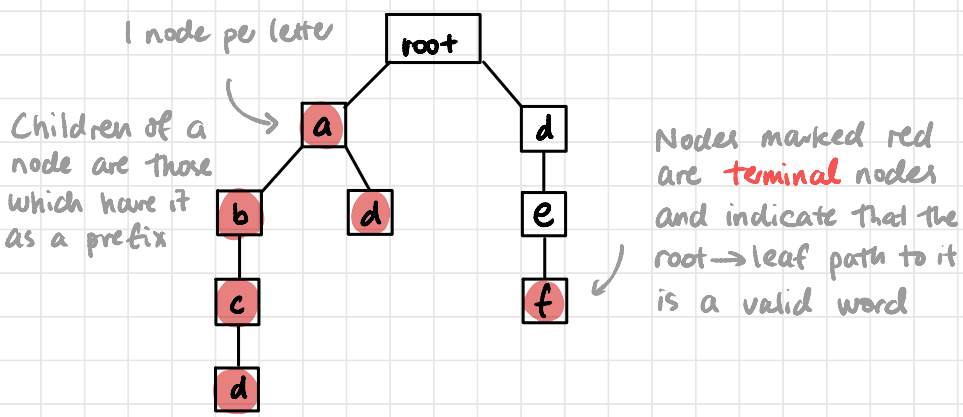
Example: What if we wanted to store the words

a, ab, abc, abcd, ad, def

For all data structures so far, need $1+2+3+4+2+3 = 15$ chars to store them + any other bits of data (e.g. pointers) in our data structure of choice.

However: The first 4 strings all share a common prefix! In principle, can store the first 4 strings with just 4 chars.

How? Use a variation on a tree to efficiently encode prefixes:



Now we only need 8 nodes! ⇒ Trie data structure.

Complexity of insertion, deletion and search are all O(len(string)).

4. Two deletion cases:

①

②