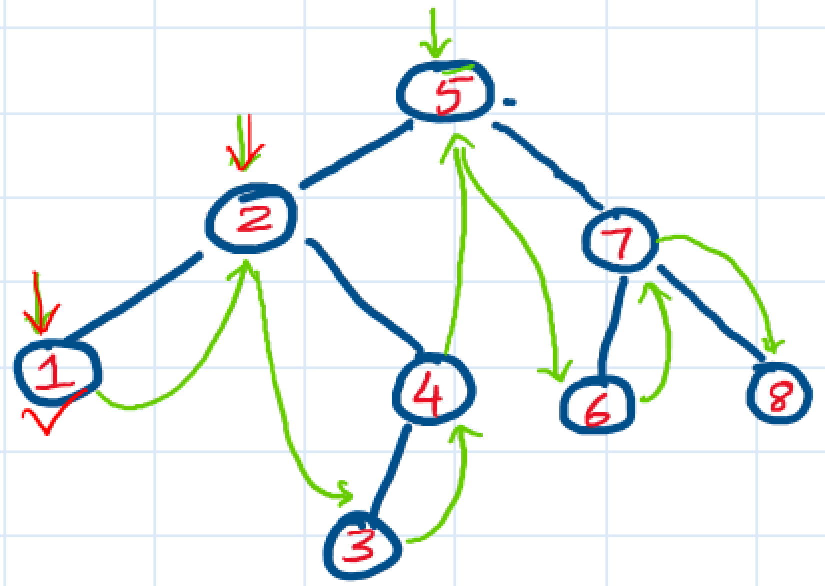
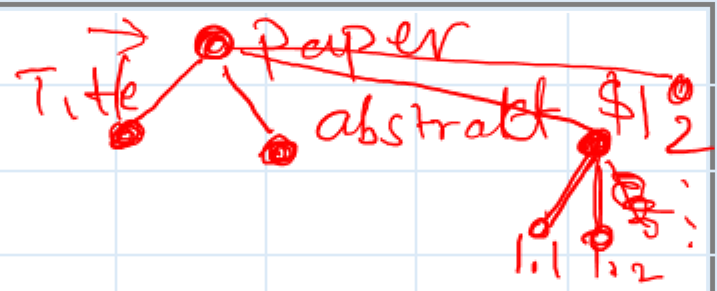
COL106 - Data Structures and Algorithms

IN-ORDER TRAVERSAL OF BINARY TREE T

* TRAVERSE INORDER THE LEFT SUBTREE OF THE ROOT OF T
* VISIT THE ROOT NODE OF T
* TRAVERSE INORDER THE RIGHT SUBTREE OF THE ROOT OF T



Tree Traversals for ToC generation




```
Paper
Title
Abstract
§1
§1.1
§1.2
§2
§2.1
...
```
(a)

```
Paper
  Title
  Abstract
  §1
    §1.1
    §1.2
  §2
    §2.1
    ...
```
(b)

```
for (TreeNode<E> p : T.preorder()) {
      System.out.println(p.getElement());
}
```

```
for (TreeNode<E> p : T.preorder()) {
      System.out.println(spaces(2 * T.depth(p)) +
                 p.getElement());
}
```

```
public static <E> void printPreorderIndent (Tree<E> T, TreeNode<E> p, int d) {
     System.out.println (spaces(2 * d)+p.getElement());
     for(TreeNode<E> c: T.children(p)){
          printPreorderIndent (T, c, d+1);
     }
}
```

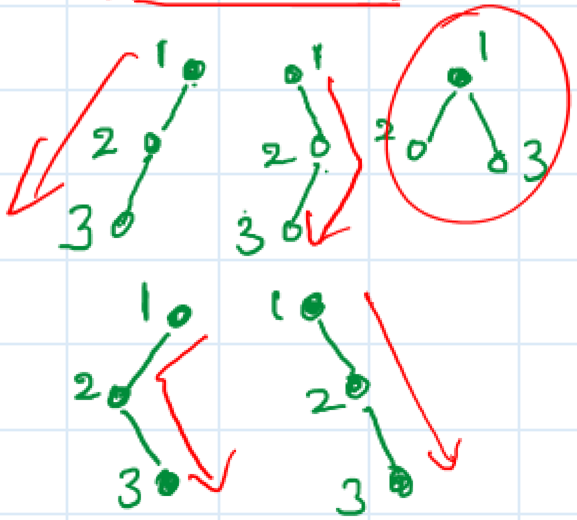
(T, T.root(), 0)

[1,2,3,4,5,6]

Representing binary trees using their traversals

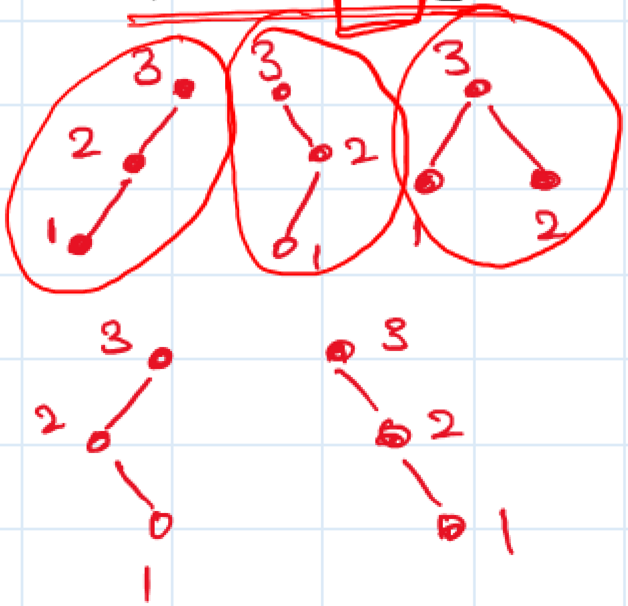
Preorder?

[1, 2, 3]



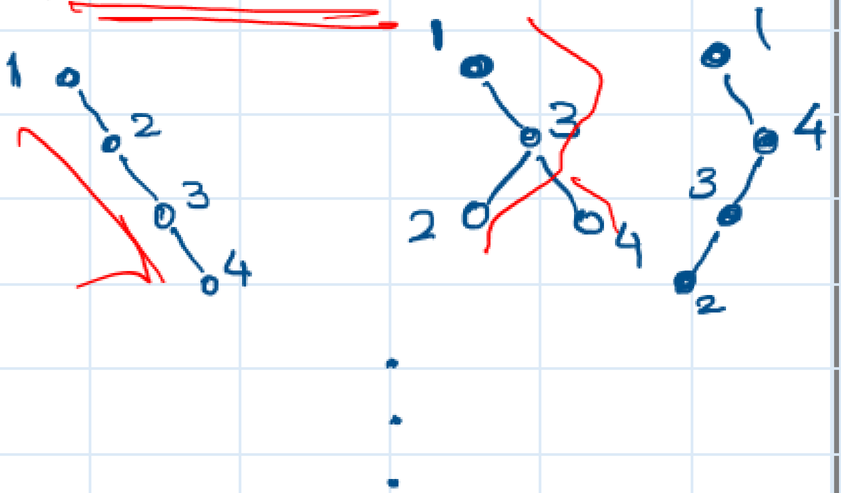
Postorder?

[1, 2, 3]



Inorder?

[1, 2, 3, 4]

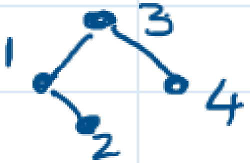


Complete the rest

⇒ No single traversal can uniquely determine binary tree

A little bit of embellishments can easily fix

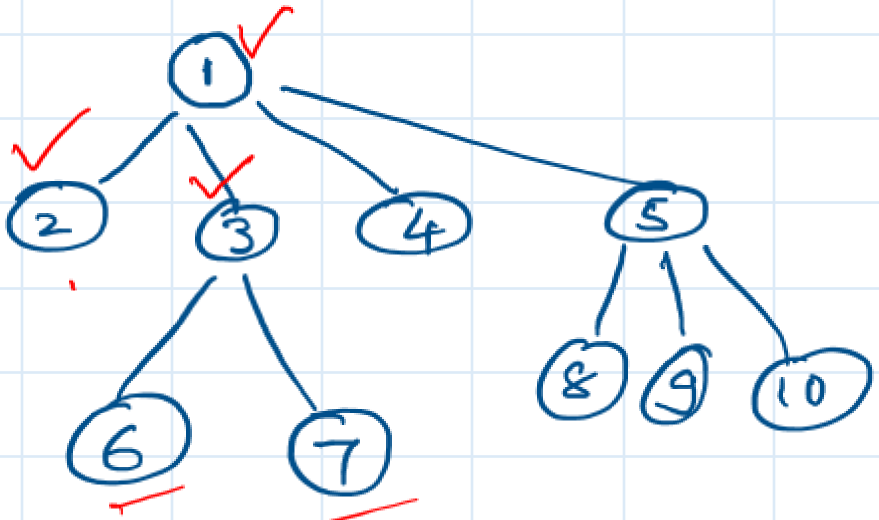
e.g. [[1,[2]],3,[4]] ⇒



PARANTHETIC REPRESENTATION OF A TREE

$$P(T) = p.getElement() + \text{"("} + P(T_1) + \text{","} + \ldots \ldots + P(T_k) + \text{")"}$$

Recursively generate a preorder paranthetic representation.



$\Rightarrow$ 1 (2(), 3 (6(), 7()), 4(), 5 (8(), 9(), 10()))

For In order ? (in Binary Trees)

(P(L)), P, (P(R))

PRIORITY QUEUES

Unlike regular FIFO queues often one needs to process elements according to some "priority".

Priority queue is a collection which allows

(a) arbitrary element insertion
(b) removal of the element that has first priority.

priority is assigned to an element as its $\boxed{\text{key}}$

Key is often represented as a number but any object which has a way to compare any two instances of the object to establish a natural ordering.

PRIORITY QUEUE ADT

insert(k, v)
Creates an entry with key k and value v in the priority queue.

min()
returns the entry (k, v) with minimal key

removeMin()
removes and returns the entry (k, v) with minimal key. null if empty priority queue.

size() and **isEmpty()**

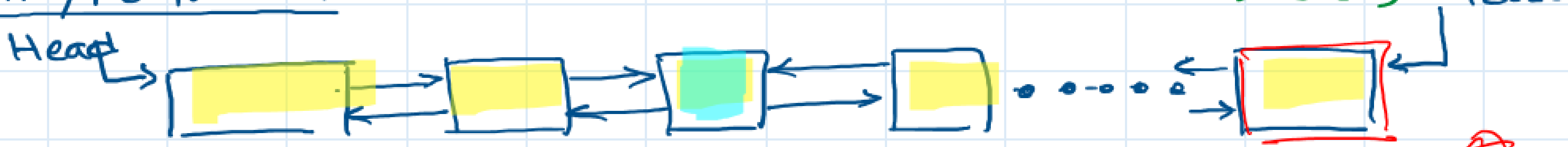
```
interface Entry<K,V> {
    K getKey();
    V getValue();
}
```

IMPLEMENTING A PRIORITY QUEUE (LINKED LIST)

⇒ Store Entry objects in a doubly linked list.

insert(k, v) : create Entry and insert $\longrightarrow O(1)$

min / removeMin : $\longrightarrow O(n)$



We need to go through the list to find the entry with smallest key value. $O(n)$ n $\frac{n}{2}$

⇒ Can we keep the list sorted?

min / removeMin : can be done in $O(1)$

What about insert(k, V) now!!? $O(n)$

HEAP DATASTRUCTURE

HEAP is a binary tree which stores Entry objects in its nodes.

PROPERTY 1 (RELATIONAL PROPERTY)

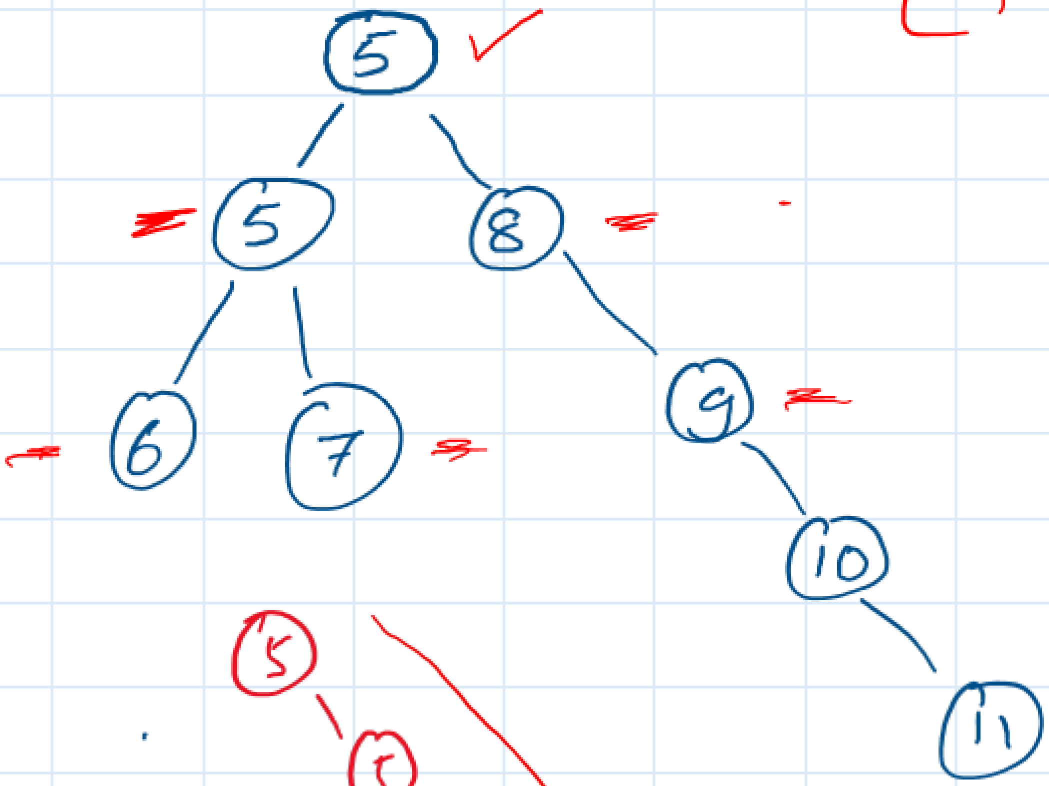
IN A HEAP T, FOR EVERY NODE P, THE KEY STORED AT P IS GREATER THAN OR EQUAL TO THE KEY STORED AT ITS PARENT (EXCEPT ROOT)

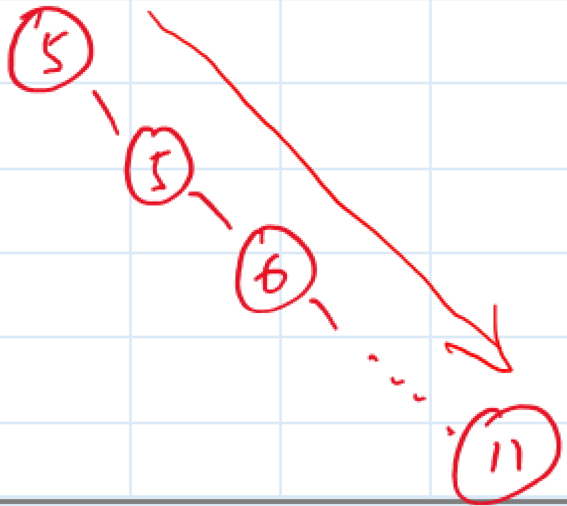
⇒ when we traverse any root – leaf path the keys encountered are strictly non-decreasing

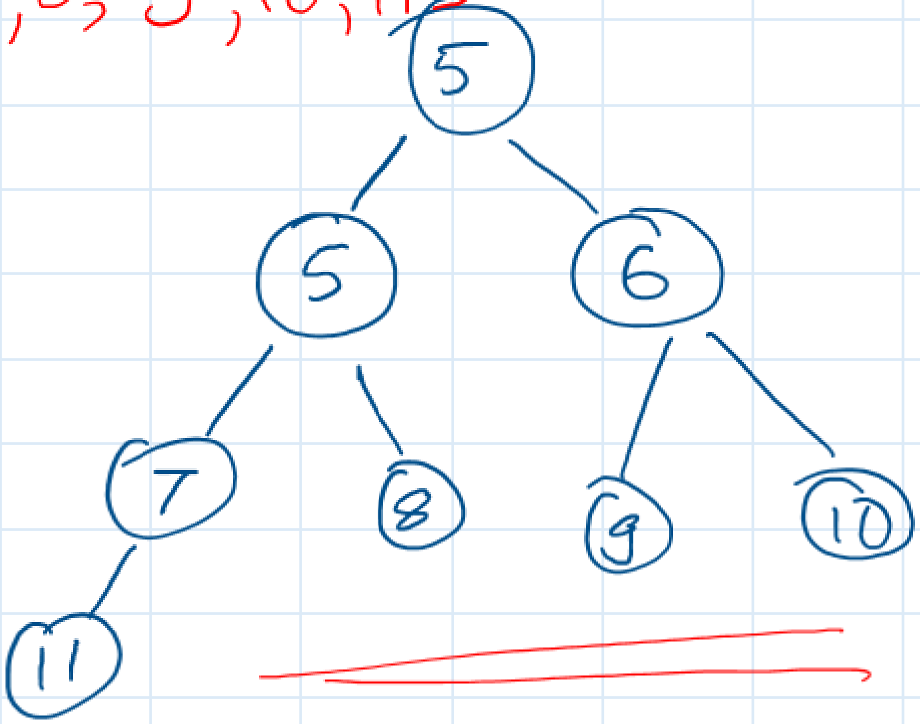
⇒ the root contains the minimal key.

(for simplicity we only show keys stored in each node of the tree)

[5, 5, 6, 7, 8, 9, 10, 11]







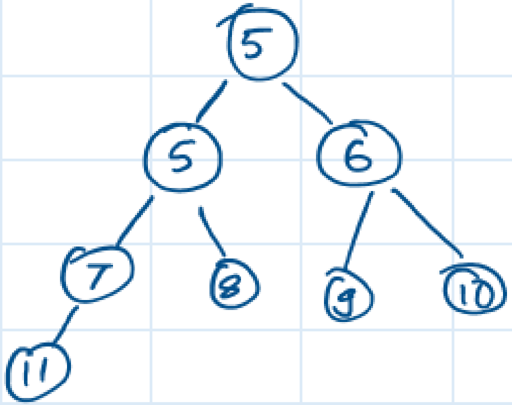
WE SHOULD AVOID BAD TREE CONFIGURATIONS !

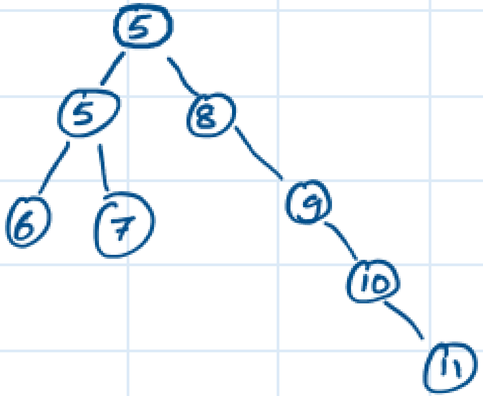
PROPERTY 2 (STRUCTURAL PROPERTY)

A heap T with height h should satisfy **complete** binary tree property.

(i) if levels 0, 1, ..., h-1 of T have maximal number of nodes possible

(ii) remaining nodes at level h reside in the leftmost possible positions at level h.





A Heap with n entries has height $h = \lfloor \log_2 n \rfloor$

T is a complete binary tree upto $h-1$ levels.

$$\Rightarrow \# \text{ of entries } = 1+2+4+\cdots\cdots+2^{h-1}$$

$$= 2^h - 1$$

Number of entries in level h is at least 1. and at most 2^h

$$\Rightarrow \quad n \geq 2^h - 1 + 1 = 2^h \qquad\qquad n \leq 2^h - 1 + 2^h = 2^{h+1} - 1$$

$$\Rightarrow \quad h \leq \log_2 n \qquad\qquad h \geq \log(n+1) - 1$$

$$\Rightarrow \quad h = \lfloor \log n \rfloor$$

IF WE CAN IMPLEMENT PRIORITY QUEUE USING HEAPS S.T. ALL OPERATIONS ARE IN TIME PROPORTIONAL TO THE HEIGHT OF HEAP, THEN THEY RUN IN $O(\log n)$ TIME