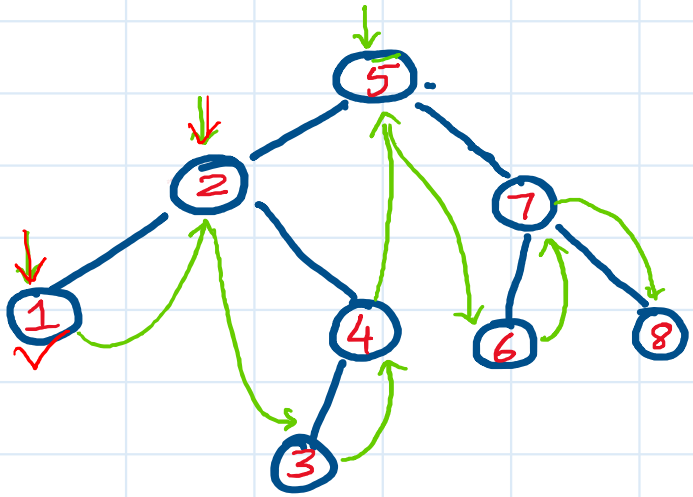
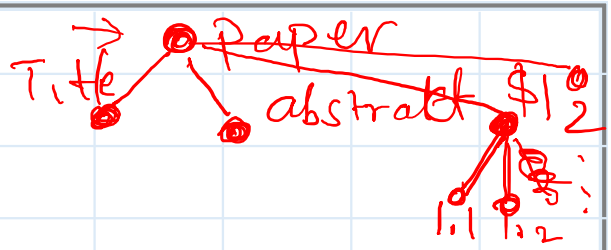
COL106 - Data Structures and Algorithms

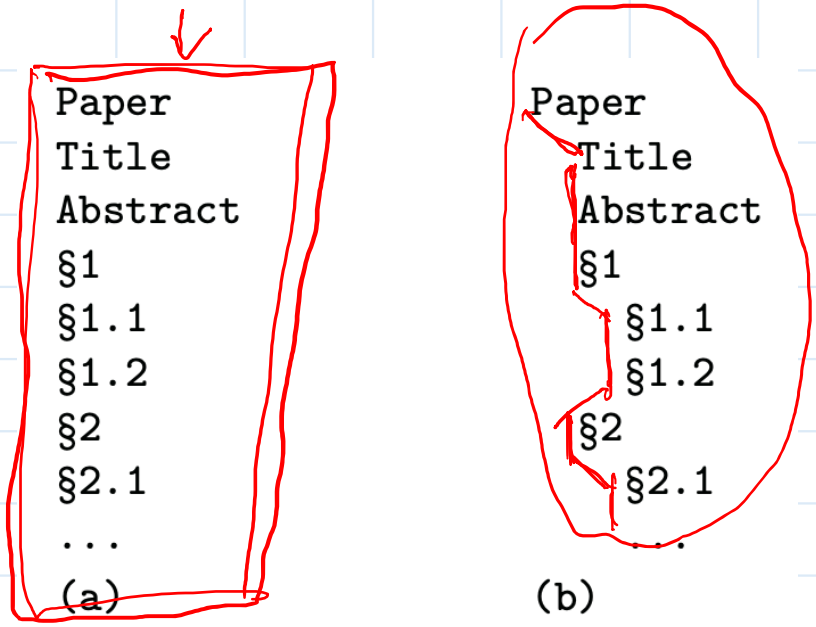
IN-ORDER TRAVERSAL OF BINARY TREE T

1. TRAVERSE INORDER THE LEFT SUBTREE OF THE ROOT OF T
2. VISIT THE ROOT NODE OF T
3. TRAVERSE INORDER THE RIGHT SUBTREE OF THE ROOT OF T



Tree Traversals for ToC generation






```
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);
     }
}
```

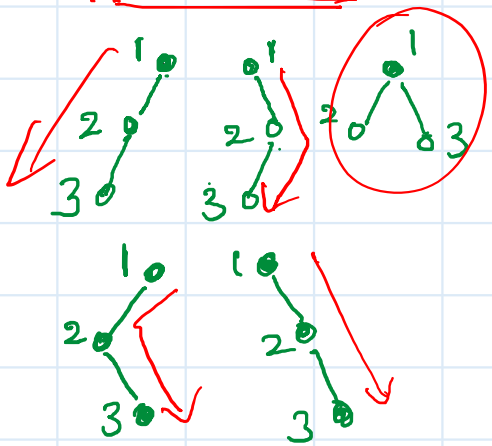
(T, T.root(), 0)

Representing binary trees using their traversals

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

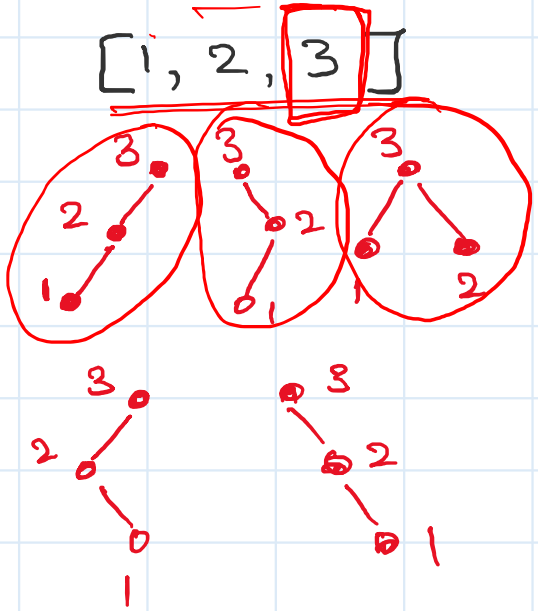
Preorder?

[1, 2, 3]



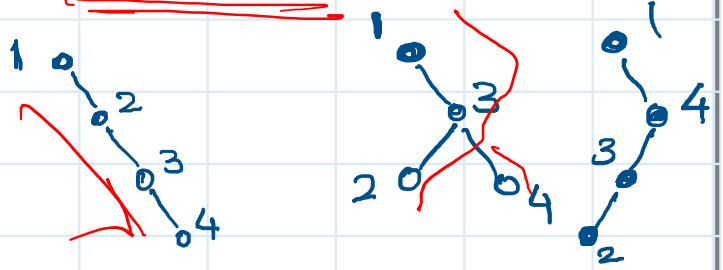
Postorder?

[1, 2, 3]



Inorder?

[1, 2, 3, 4]

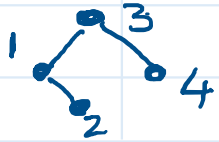


Complete the rest

⇒ No single traversal can uniquely determine binary tree

A little bit of embellishments can easily fix

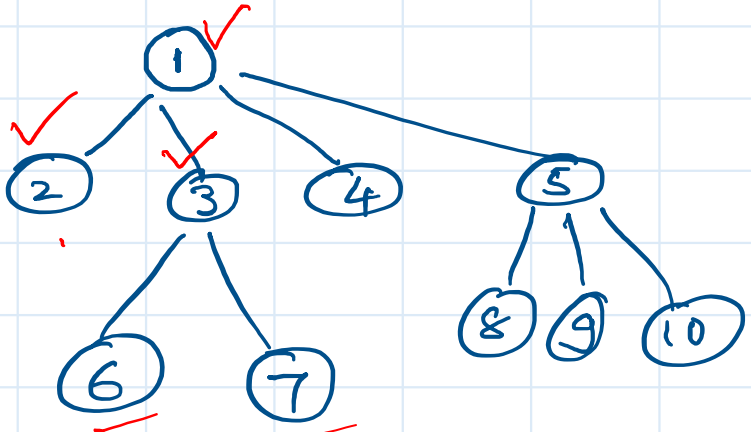
e.g. [[1,[2]],3,[4]] ⇒



PARANTHETIC REPRESENTATION OF A TREE

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

Recursively generate a preorder paranthetic representation.



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

For Inorder ? (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 **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

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

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.

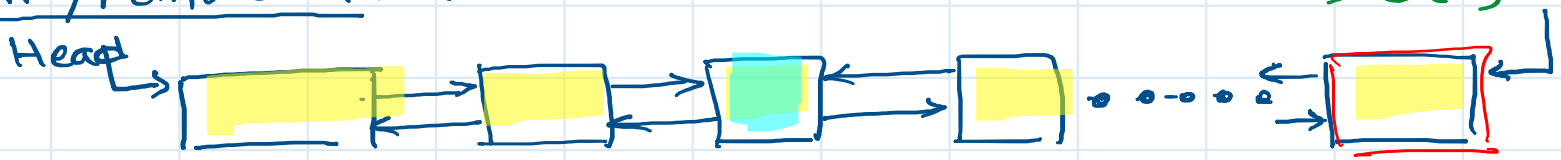
size() and **isEmpty()**

Implementing A Priority Queue (Linked List)

⇒ Store Entry objects in a doubly linked list.

insert(k, v) : create Entry and insert → O(1)

→ min / removeMin : → O(n)



We need to go through the list to find the entry with smallest keyvalue. 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.

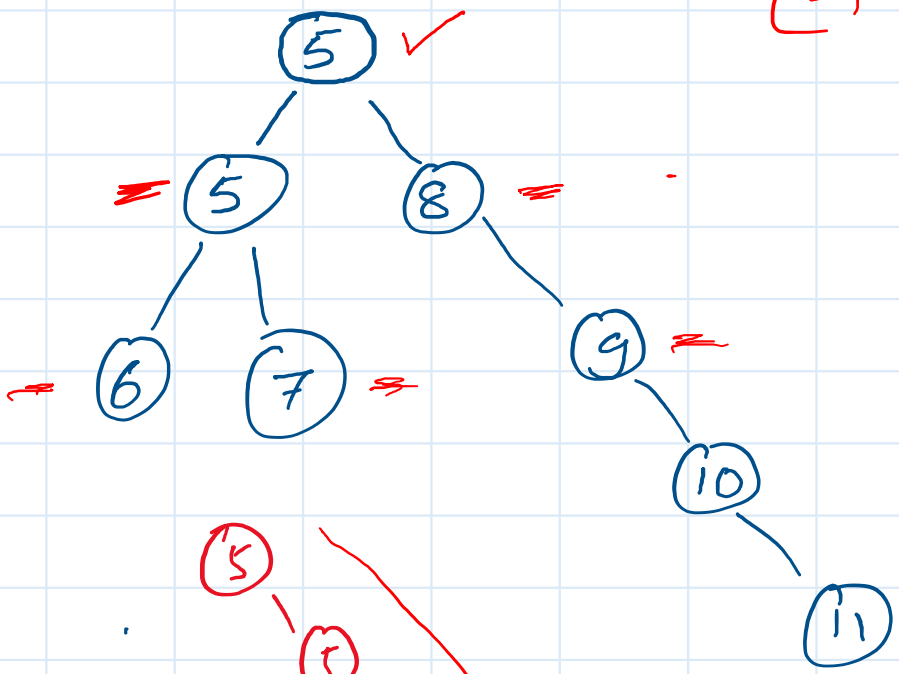
PROPERTY 1 (RELATIONAL PROPERTY)

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)

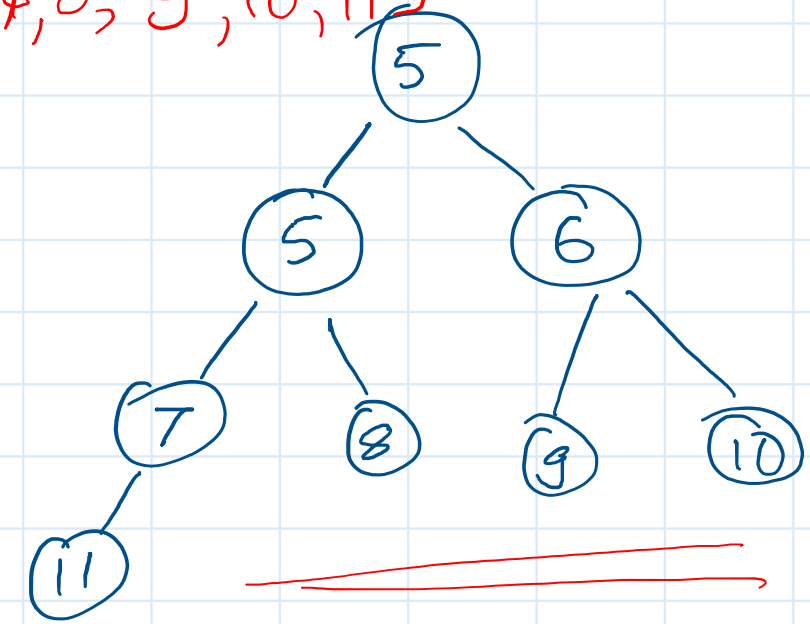
⇒ when we traverse any root - leaf path the keys encountered are strictly non-decreasing

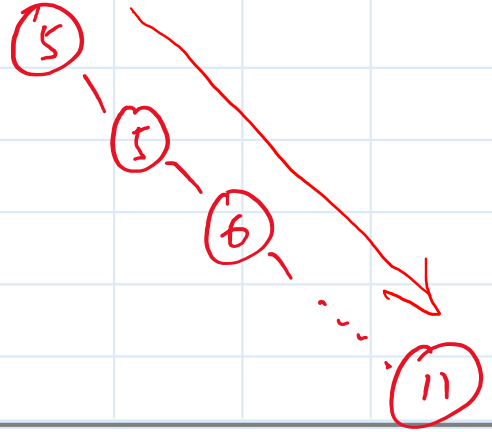
⇒ the root contains the minimal key.

(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 !