THE UNIVERSITY OF MELBOURNE

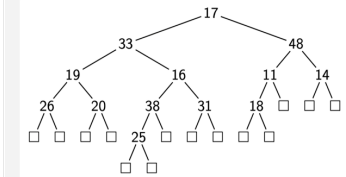
COMP90038 Algorithms and Complexity

Tutorial 7 Trees, Priority Queues, Heaps and Heapsort

Tutor: Chuang(Frank) Wang

1. Review

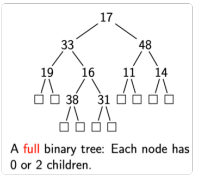
THE UNIVERSITY OF MELBOURNE



Trees

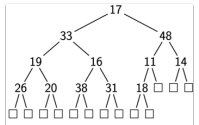
Binary Tree

1. **Def**: an ordered tree in which every vertex has no more than two children and each child is designated as either a left child or a right child of its parent; a binary tree may also be empty.
   - **Types**
     - full binary tree



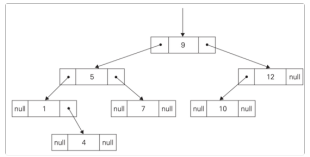
A full binary tree: Each node has 0 or 2 children.

     - complete tree



A complete tree: Each level filled left to right.

2. **Components of a node**
   - data
   - the pointer to the left subtree
   - the pointer to the right subtree



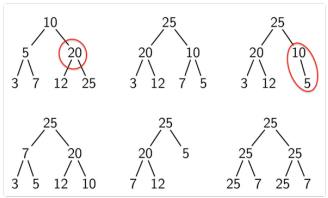
3. **The height of a tree**
   - the number of edges from the root to the deepest leaf.
   - the empty tree having height -1.
4. **Traversal:**
   - **Preorder**: Root -> Left -> Right
   - **Inorder**: Left -> Root -> Right
   - **Postorder**: Left -> Right -> Root
   - **Level-order**: level by level starting from root

BST (Binary Search Tree)

1. **Def:** a binary tree that satisfies **left < root < right** (Think recursively!)
2. **Operations**
   - Search
     - Balanced VS Unbalanced
   - Insert
   - Delete

(MAX) Heap

1. **Def:** a complete binary tree which satisfies the heap condition: **Each child has a priority (key) which is no greater than its parent's.**



2. **Height:** $\lfloor \log_2 n \rfloor$
3. **Properties:**
   - The children of node i are 2i and 2i + 1.
   - The nodes which happen to be parents are in array positions 1 to $\lfloor n/2 \rfloor$.
4. **Operations:**
   - **Injecting - Climb up**
   - **Building a Heap Bottom-Up: Sifting Down**
   - **Ejecting a Maximal Element ------>**

Maximum Key Deletion from a heap

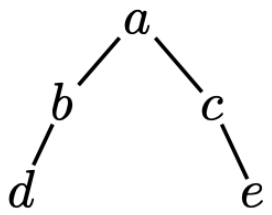
Step 1 Exchange the root's key with the last key K of the heap.

Step 2 Decrease the heap's size by 1.

Step 3 "Heapify" the smaller tree by sifting K down the tree exactly in the same way we did it in the bottom-up heap construction algorithm. That is, verify the parental dominance for K: if it holds, we are done; if not, swap K with the larger of its children and repeat this operation until the parental dominance condition holds for K in its new position.

2. Tutorial Questions

Question 51

51. Traverse 

in preorder, inorder, and postorder.

Preorder: Root -> Left -> Right
Inorder: Left -> Root -> Right
Postorder: Left -> Right -> Root
Level-order: level by level starting from root

Preorder: a, b, d, c, e

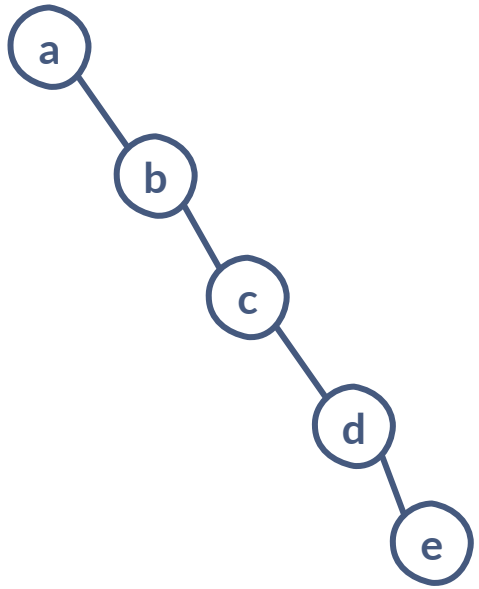
Inorder: d, b, a, c, e

Postorder: d, b, e, c, a

Question 52

52. A certain binary tree yields a, b, c, d, e when traversed preorder, and it yields the same sequence when traversed inorder. What does the binary tree look like?

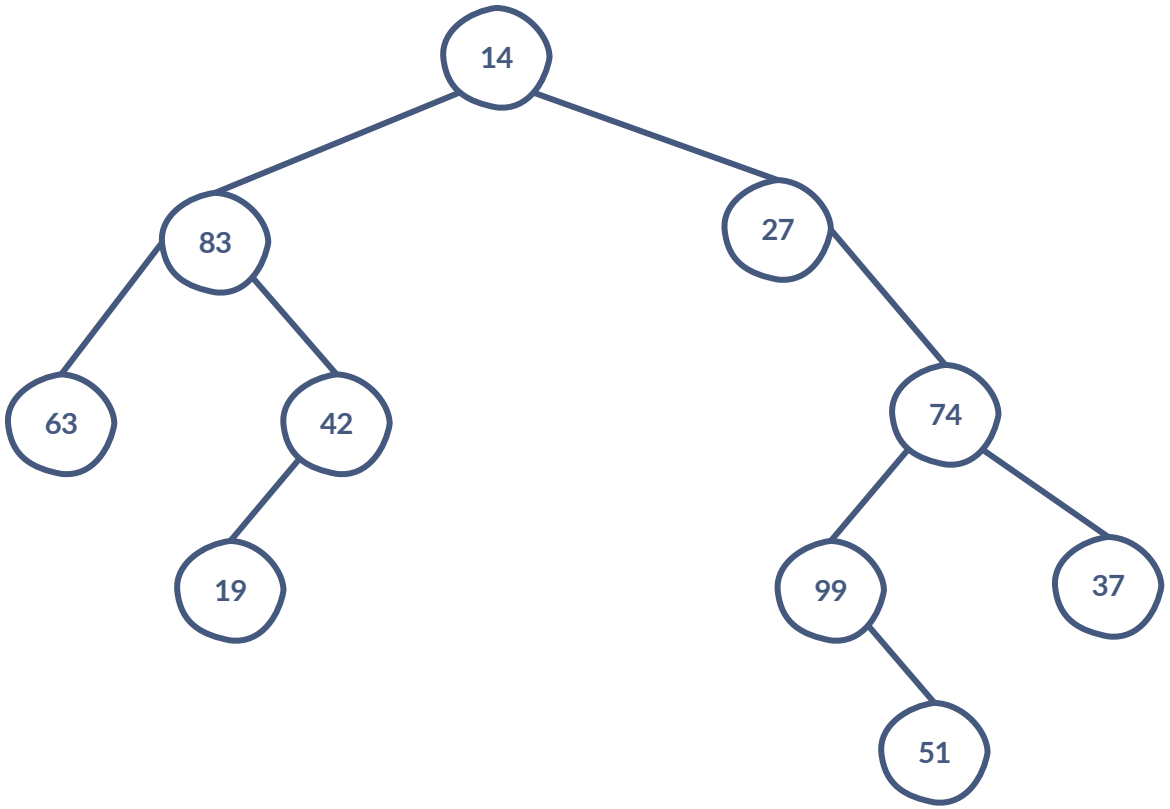
Preorder: Root -> Left -> Right
Inorder: Left -> Root -> Right
Postorder: Left -> Right -> Root
Level-order: level by level starting from root



Question 53

53. A certain binary tree yields 14, 83, 63, 42, 19, 27, 74, 99, 51, 37 when traversed preorder and 63, 83, 19, 42, 14, 27, 99, 51, 74, 37 when traversed inorder. Which sequence does it yield when traversed postorder?

Preorder: Root -> Left -> Right
Inorder: Left -> Root -> Right
Postorder: Left -> Right -> Root
Level-order: level by level starting from root



63, 19, 42, 83, 51, 99, 37, 74, 27, 14.

Question 54

54. The following algorithm was designed to compute the number of leaves in a binary tree T. We denote the empty tree by *null*.

```
function LeafCount(T)
    if T = null then
        return 0
    else
        return LeafCount(T.left) + LeafCount(T.right)
```

Fix the error in this algorithm.

(a)

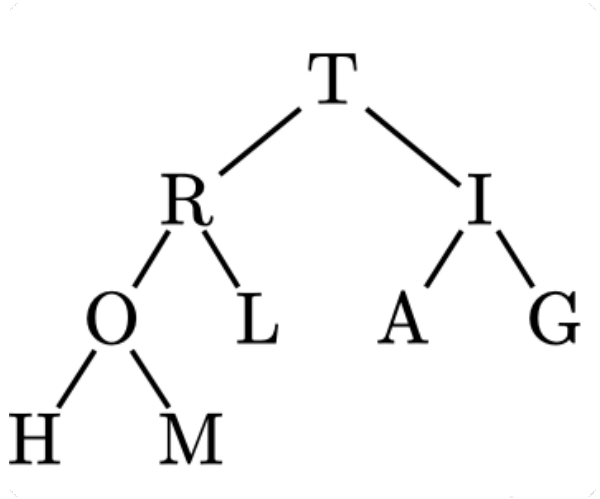
```
function LeafCount(T)
    if T = null then
        return 0
    else
        if T.left = null and T.right = null then
            return 1
        else
            return LeafCount(T.left) + LeafCount(T.right)
```

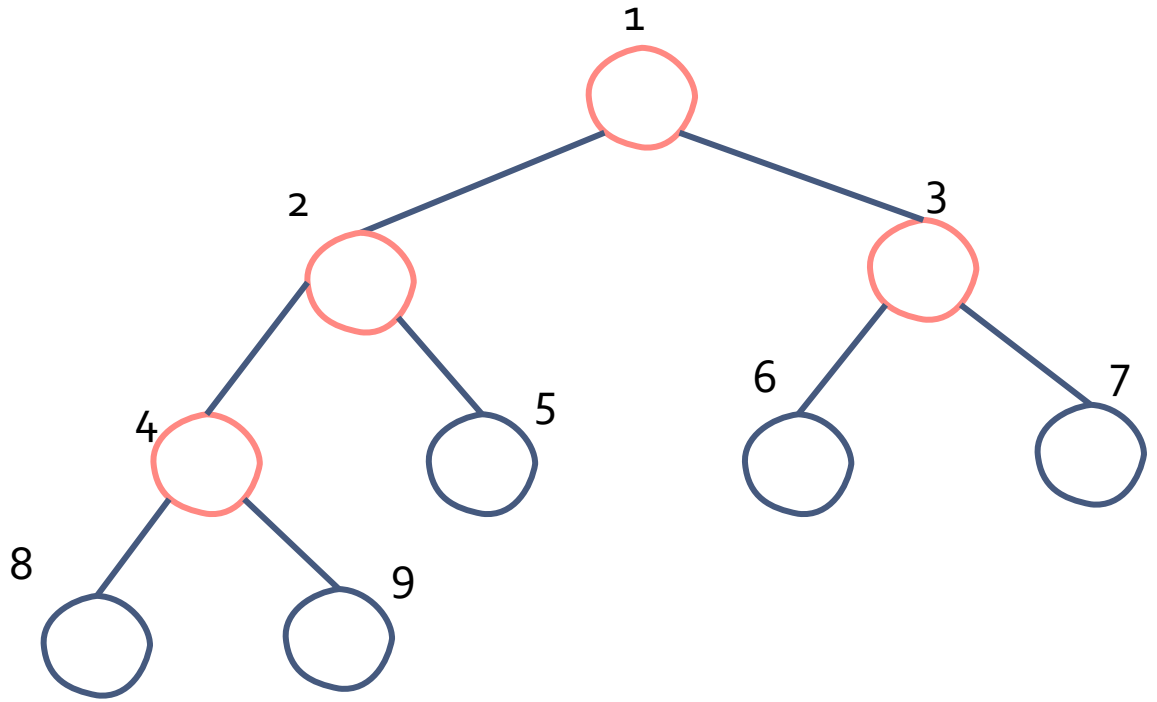
Question 56

56. Make a max-heap out of the keys A, L, G, O, R, I, T, H, M, using the bottom-up algorithm.

1, 10, 7, 15, 18, 9, 20, 8, 13

from last parent to the root parent:
max-heapify



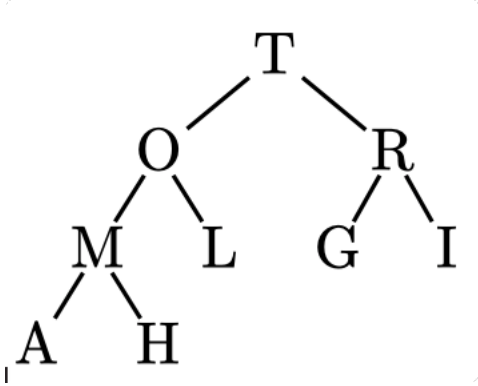


Question 57

57. Construct a max-heap from the empty heap by inserting the keys A, L, G, O, R, I, T, H, M, one by one, in that order. Is the result the same as the heap from the previous question?

1, 10, 7, 15, 18, 9, 20, 8, 13

- ✓ Place the new item at the end;
- ✓ then let it "climb up", repeatedly swapping with parents that are smaller



Question 58

58. Give an algorithm for deciding whether an array $A[1]..A[n]$ is a heap.

Properties of the Heap

The root of the tree $H[1]$ holds a maximal item; the cost of EJECT is $O(1)$ plus time to restore the heap.

The height of the heap is $\lfloor \log_2 n \rfloor$.

Each subtree is also a heap.

The children of node i are $2i$ and $2i+1$.

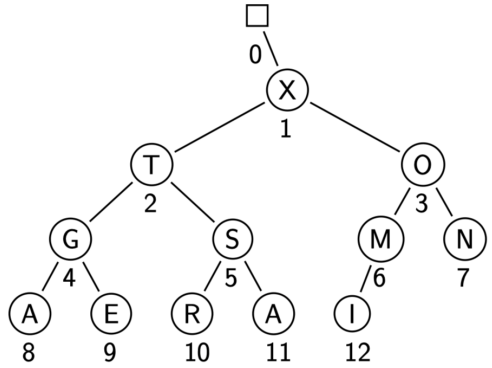
The nodes which happen to be parents are in array positions 1 to $\lfloor n/2 \rfloor$.

It is easier to understand the heap operations if we think of the heap as a tree.

Heaps as Arrays

This way, the heap condition is very simple:

For all $i \in \{0, 1, \ldots, n\}$, we must have $H[i] \leq H[i/2]$.



H:		X	T	O	G	S	M	N	A	E	R	A	I
	0	1	2	3	4	5	6	7	8	9	10	11	12

Answer: Easy:

```
function IsHeap(A[·], n)
    for i ← 2 to n do
        if A[i] > A[i/2] then
            return False
    return True
```

Question 59

59. Apply the heapsort algorithm to A, L, G, O, R, I, T, H, M.

Heapsort

Heapsort is a $\Theta(n \log n)$ sorting algorithm, based on the idea from this exercise.

Given an unsorted array $H[1] \ldots H[n]$:

Step 1 Turn H into a heap.

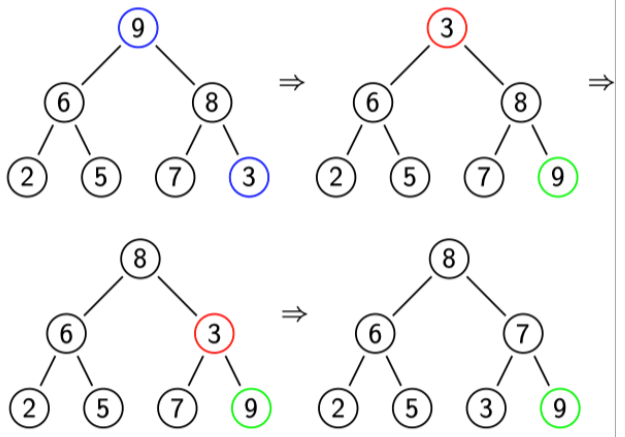
Step 2 Apply the eject operation $n - 1$ times.

Ejecting a Maximal Element from a Heap

Here the idea is to swap the root with the last item z in the heap, and then let z "sift down" to its proper place.

After this, the last element (here shown in green) is no longer considered part of the heap, that is, n is decremented.

Clearly ejection is $O(\log n)$.



THE UNIVERSITY OF MELBOURNE

Next Week:

AVL trees and 2–3 trees.

Thank you !