

# COMP20007 Workshop Week 6

1 **Topic 1:** DFS & Topological Sorting

**Exercise:** Q6.1 (toposort)

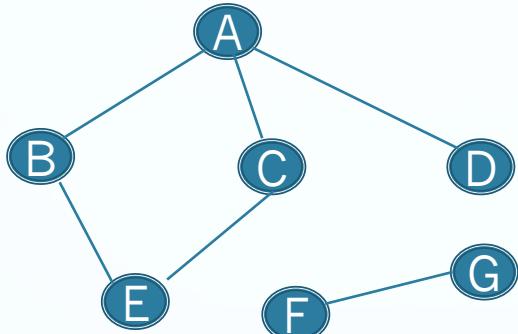
2 **Topic 2:** Binary Trees & BST

**Group/Individual Exercises:** Q 6.2, 6.4, 6.3

## Lab: **ASSIGNMENT 1**

- Finish ass1 if not yet done, or
- review workshops week 5-6 and ask questions
- on request: understanding BFS and Prim's, Dijkstra's

# DFS review: Q&A on algorithm & complexity



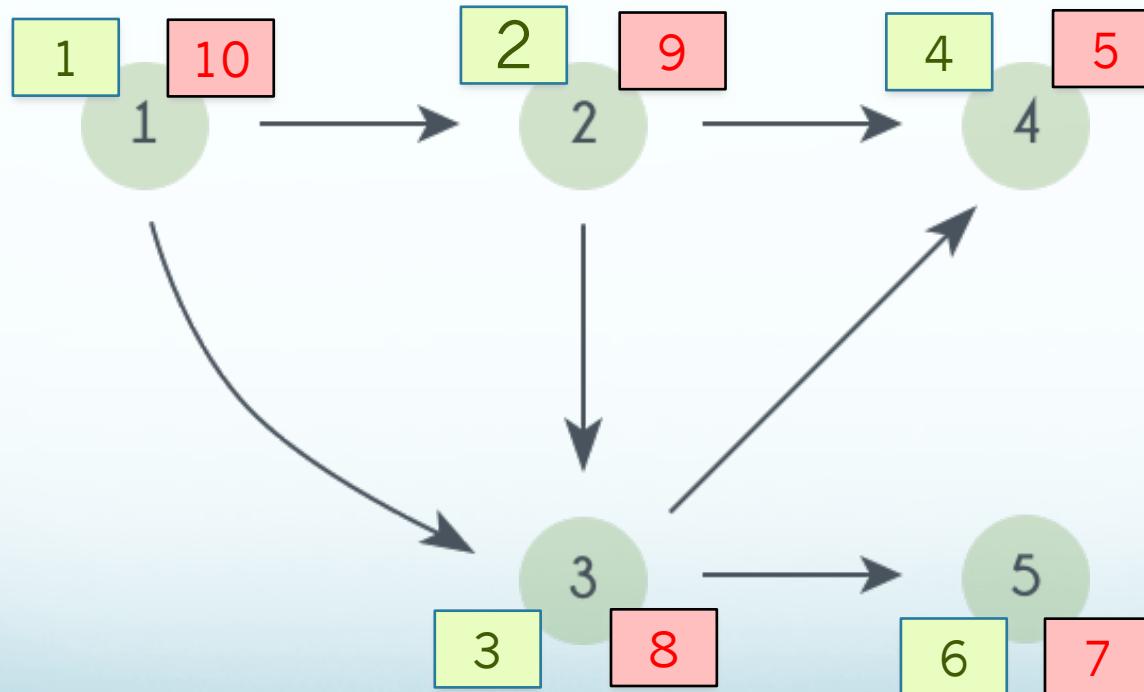
```
function DFS(G=(V,E))
    for each v in V do
        mark v with 0
    for each v in V do
        if v is marked with 0 then
            DFSEXPLORE(v)

function DFSEXPLORE(v)
    // start visiting v
    mark v with 1
    for each edge (v,w) in E do
        if w is marked with 0 then
            DFSEXPLORE(w)
    // end visiting v
```

# DFS exercise: push- and pop-order (pre- and post-order)

## Problem:

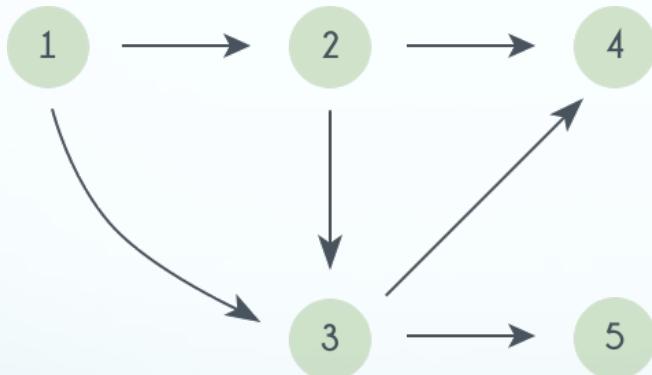
- For the graph below, write the push and pop order for DFS, starting from node 1
- Method 1:** Fill in the yellow boxes with push-orders, pink boxes with pop-orders.  
Note that you can start pop orders with 1, you can also use timestamps for both push- and pop-orders.



# DFS exercise: push- and pop-order (pre- and post-order)

## Problem:

- For the graph below, write the push and pop order for DFS, starting from node 1
- Method 2:** Explicitly show the order of the push and pop operations, and also show the content of the stack. Fill in the yellow box.



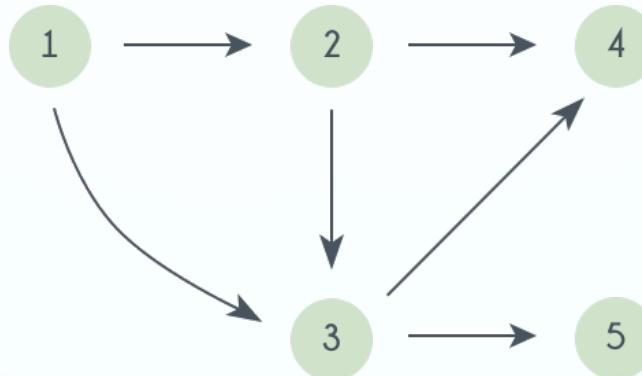
The PUSH and POP, using \$ for the bottom of stacks.

| ops    | stack content |
|--------|---------------|
| init   | \$            |
| push 1 | \$1           |
| + 2    | \$12          |
| + 3    | \$123         |
| + 4    | \$1234        |
| - 4    | \$123         |
| + 5    | \$1235        |
| - 5    | \$123         |
| - 3    | \$12          |
| - 2    | \$1           |
| - 1    | \$            |

# DFS exercise: push- and pop-order, DFS complexity

**Problem:** Modify the DFS algorithm so that it also builds the arrays `push[V]` and `pop[V]` to store the push- and the pop-order of the vertices.

Example graph:



```
// building push[V] and pop[V]
function DFS(G=(V,E))
    for each v in V do
        mark v with 0
    for each v in V do
        if v is marked with 0 then
            DFSEXPLORE(v)

function DFSEXPLORE(v)
    mark v with 1
    for each edge (v,w) in E do
        if w is marked with 0 then
            DFSEXPLORE(w)
```

**Homework:** Complexity of the algorithm:

using adjacency matrix

?

using adjacency list

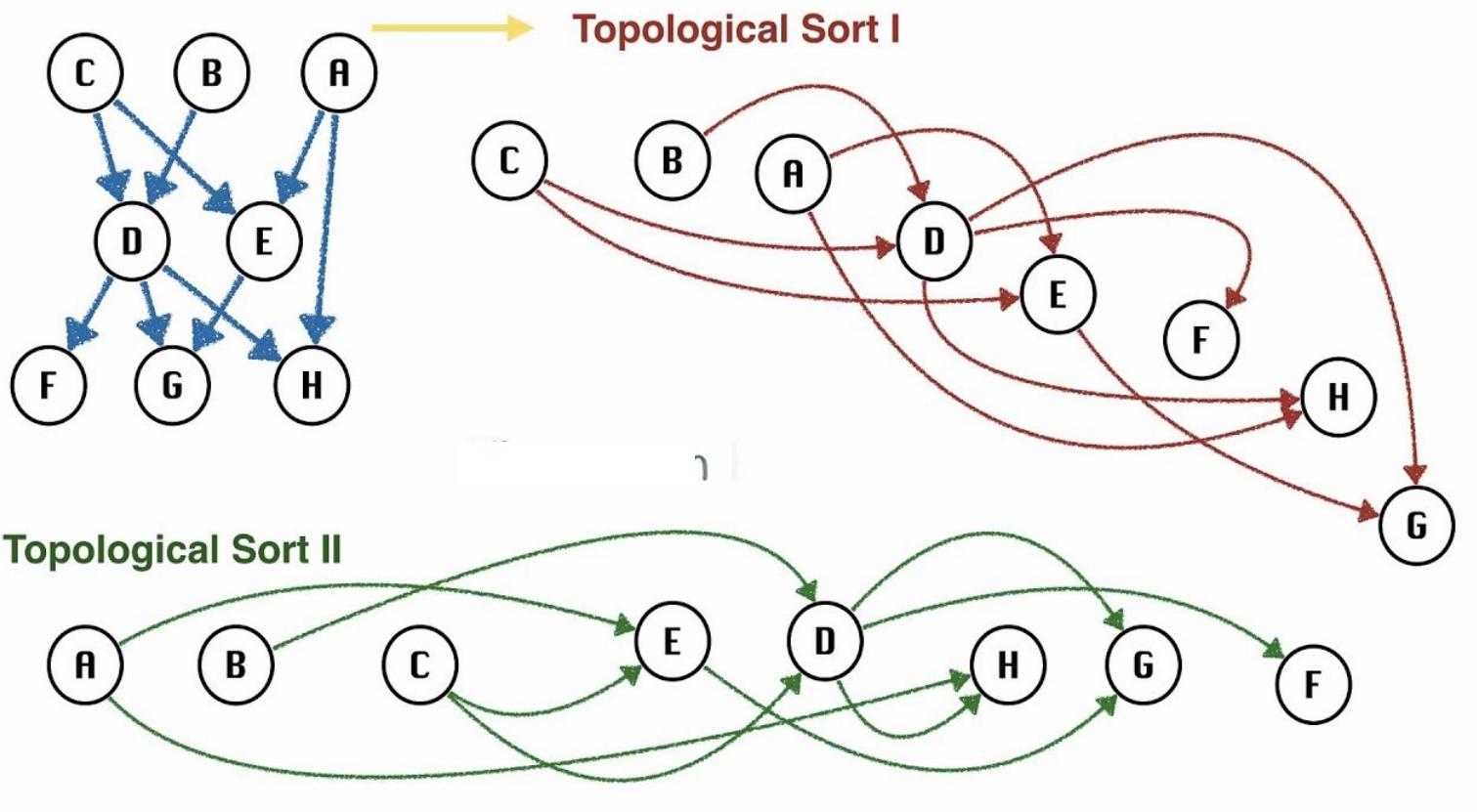
?

**Homework:** What's the complexity of

|            |   |
|------------|---|
| BFS        | : |
| Prim's     | : |
| Dijkstra's | : |

# Topological Sorting (for DAG only!)

A *topological ordering*: sorting the nodes of the graph such that all edges point in one direction, to nodes later in the ordering.



# Group Work: Q 6.1

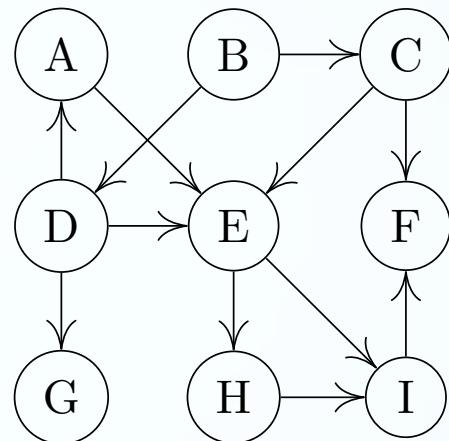
T1: Finding a topological order for the graph by running a DFS.

YOUR ANSWER:

First, run the DFS

(you can also just draw graph and write down push and pop order on the left and right of each node like we did earlier)

| operation  | content | operation | content |
|------------|---------|-----------|---------|
| init stack | \$      |           |         |
| push A     | \$A     |           |         |



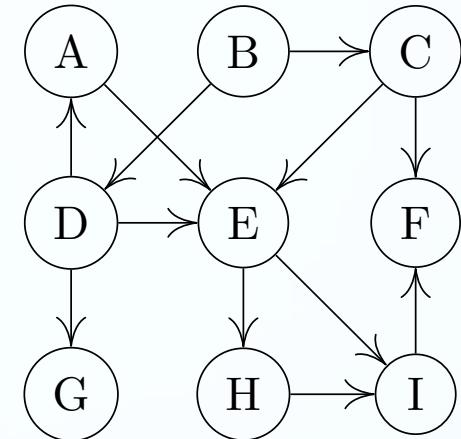
The topological order resulted from the above DFS run:

???

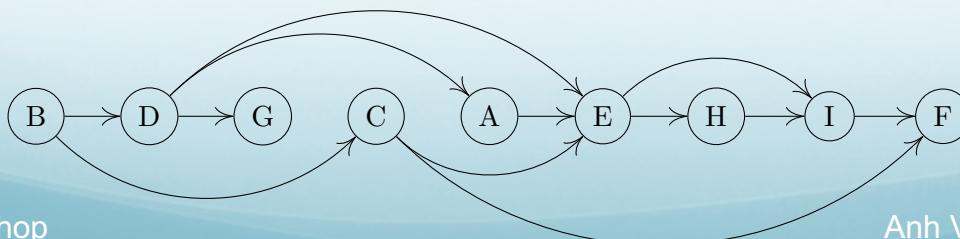
# Check: Group Work: Q6.1

T1: Finding a topological order for the graph by running a DFS.

| operation  | content | operation | content |
|------------|---------|-----------|---------|
| init stack | \$      | push B    | \$B     |
| push A     | \$A     | push C    | \$BC    |
| push E     | \$AE    | pop C     | \$B     |
| push H     | \$AEH   | push D    | \$BD    |
| push I     | \$AEHI  | push G    | \$BDG   |
| push F     | \$AEHIF | pop G     | \$BD    |
| pop F      | \$AEHI  | pop D     | \$B     |
| pop I      | \$AEH   | pop B     | \$      |
| pop H      | \$AE    |           |         |
| pop E      | \$A     |           |         |
| pop A      | \$      |           |         |

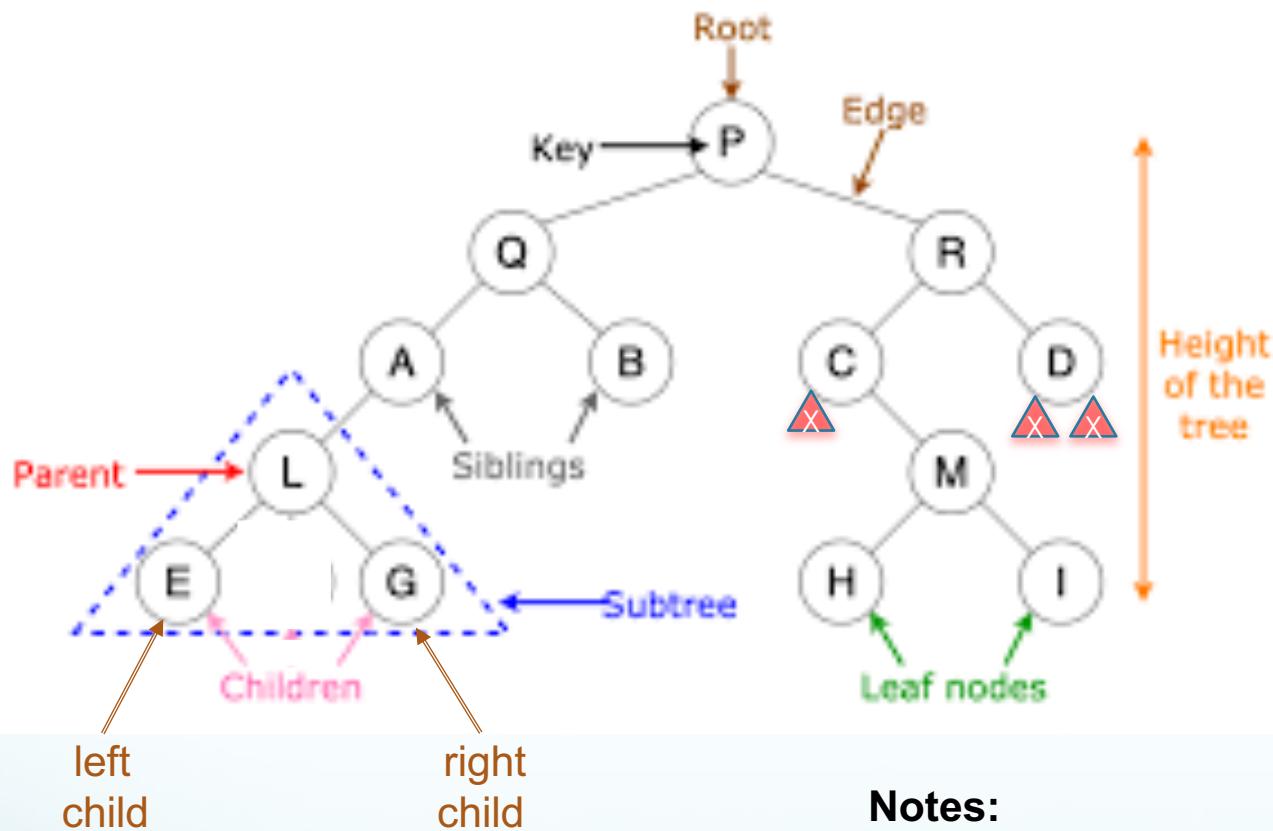


The topological order resulted from the above DFS run (= reversed pop order): B D G C A E H I F  
you can check by re-draw the graph in the above linear order.



## Topic 2

# Binary Trees as Special Graphs



### Notes:

△ denotes a NULL node, aka. *external node*, only a few of them drawn here. If the tree has  $n$  internal nodes, it has  $n+1$  external nodes

Make differences between:

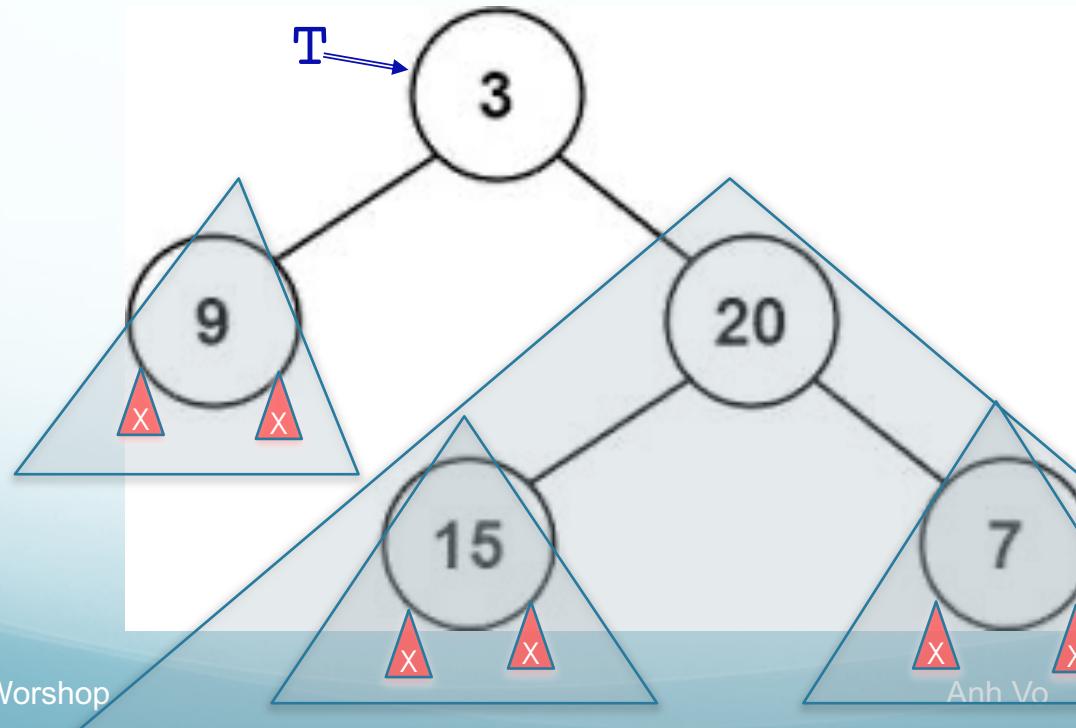
- *leaf nodes* and *external nodes*
- *none-leaf nodes* and *internal nodes*

# Binary Tree: Recursive Definition

A binary tree is:

- NULL, or
- a node, called the tree's *root node*, that contains:
  - some data, normally including a key,
  - a link to another binary tree called the root's *left child*, and
  - a link to another binary tree called the root's *right child*

**Side note:** a non-empty tree is fully defined by its root. For simplicity we use: tree  $T$  = its root node = pointer to its root node.



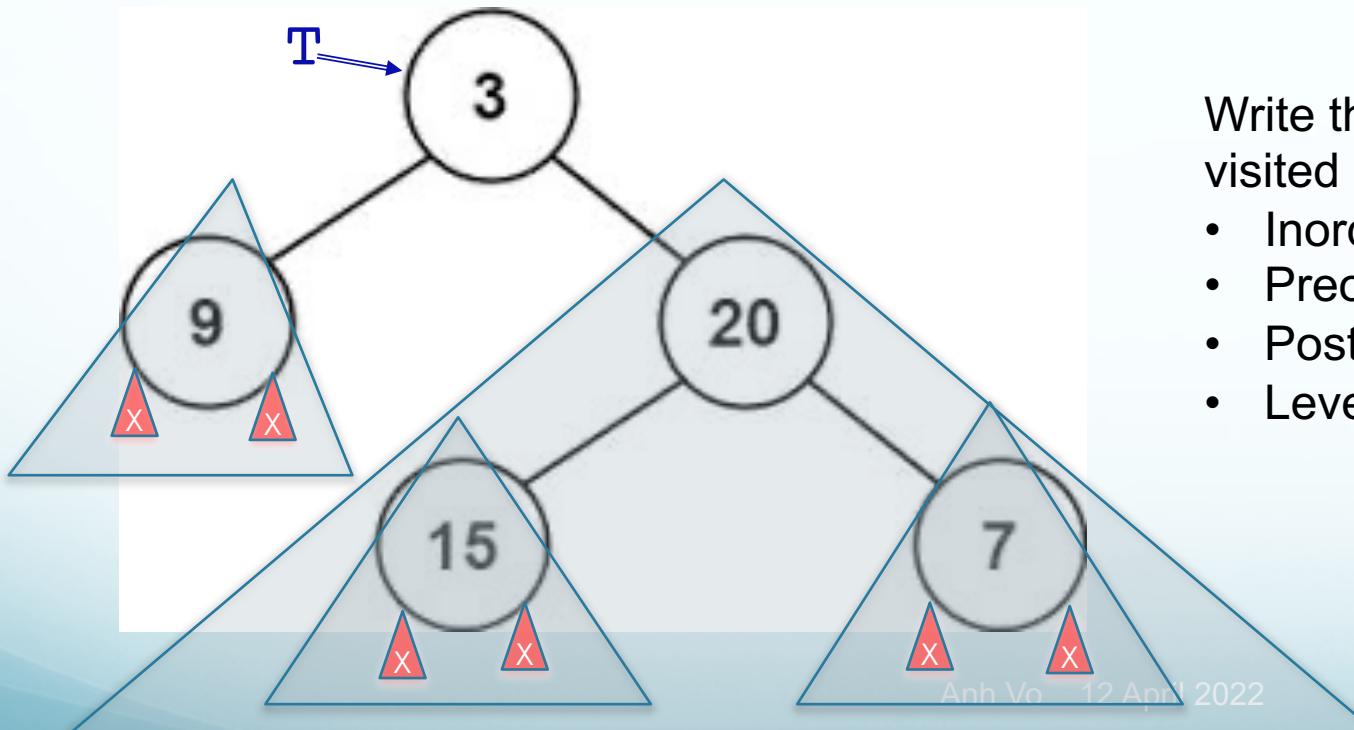
# Binary tree traversal

# What is tree traversal?

For a tree T, what job we need to do with Traversal(T)?

What is *inorder*, *preorder*, *postorder* traversal? Are they BFS or DFS?

What is *level-order* traversal? Is it BFS or DFS?



Write the nodes in order visited by:

- Inorder : 9 3 15 20 7
  - Preorder : 3 9 20 15 7
  - Postorder : 9 15 7 20 3
  - Levelorder:

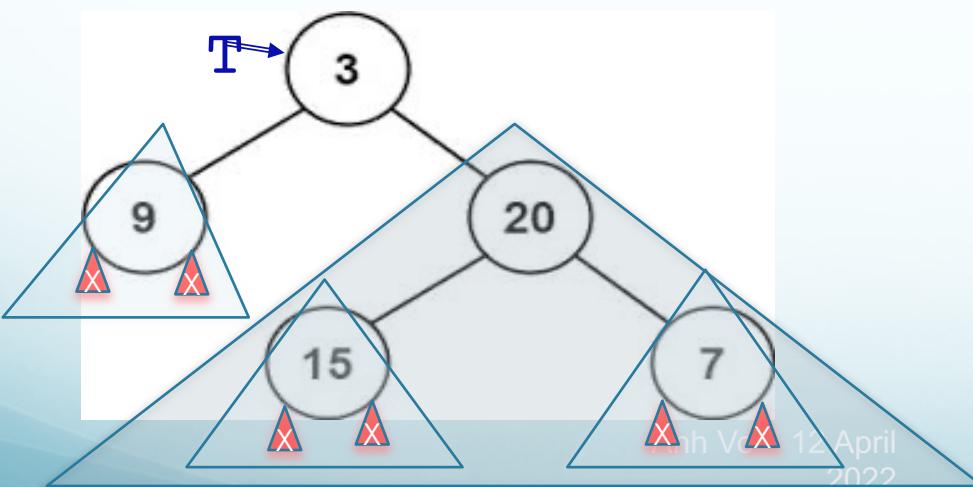
# Binary tree traversal

**What is tree traversal?** visiting all nodes of a tree

**For a tree T, what job we need to do with Traversal(T)?** 3 jobs: visit the root, traverse the left child, traverse the right child

**What is *inorder*, *preorder*, *postorder* traversal?** just depends on when we visit the root: in-between, before, or after traversing the children. All of them are DFS.

**What is *level-order* traversal?** Think of the tree as a graph, and start traversal from the root. In level-order, we visit level by level, left to right, starting from the root. This is BFS.



Write the nodes in order visited by:

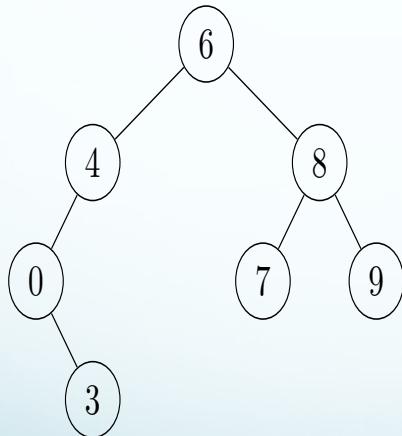
- Inorder : 9 3 15 20 7
- Preorder : 3 9 20 15 7
- Postorder : 9 15 7 20 3
- Levelorder: 3 9 20 15 7

## Q 6.4: Binary Tree Sum

Write an algorithm to calculate the sum of a binary tree where each node contains a number.

YOUR ANSWER: The pseudocode:

```
function Sum( T )
    if T is empty then
        return 0
    return T.key + Sum(T.left) +
T(T.right)
```

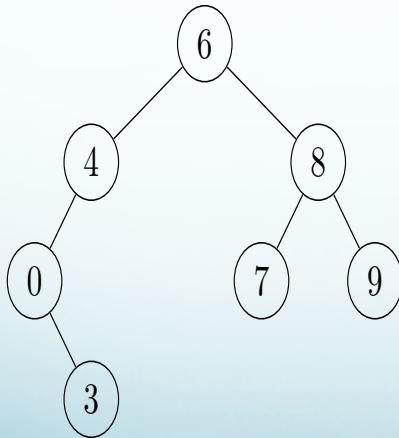


## Q 6.4: Binary Tree Sum

Write an algorithm to calculate the sum of a binary tree where each node contains a number.

YOUR ANSWER: The pseudocode:

```
function Sum( T )
    if T is empty then
        return 0
    return T.key + Sum(T.left) + Sum(T.right)
// actually inorder and postorder also OK
```



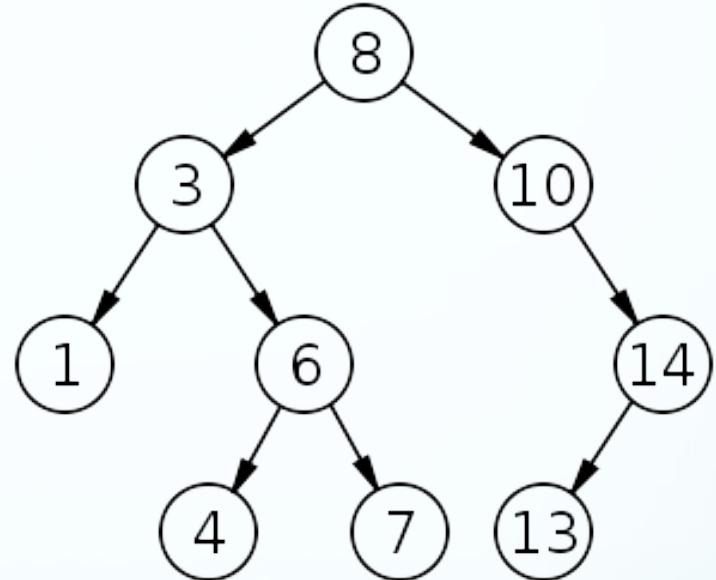
# Binary (Search) Tree

How to:

1. print the keys in increasing order?
2. print in decreasing order?
3. copy the tree?
4. free the tree?

Your answers:

- 1.
- 2.
- 3.
- 4.



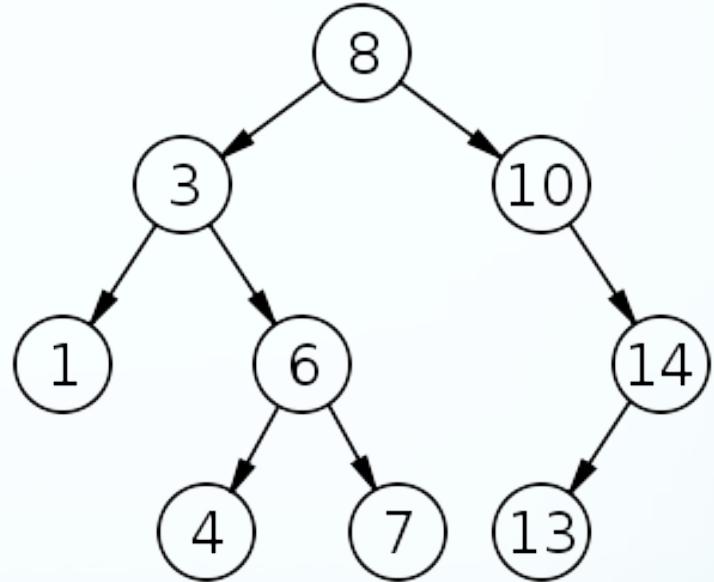
# Binary (Search) Tree

How to:

1. print the keys in increasing order?
2. print in decreasing order?
3. copy the tree?
4. free the tree?

Your answers:

1. do inorder left-root-right
2. do inorder right-root-left
3. do preorder
4. do postorder

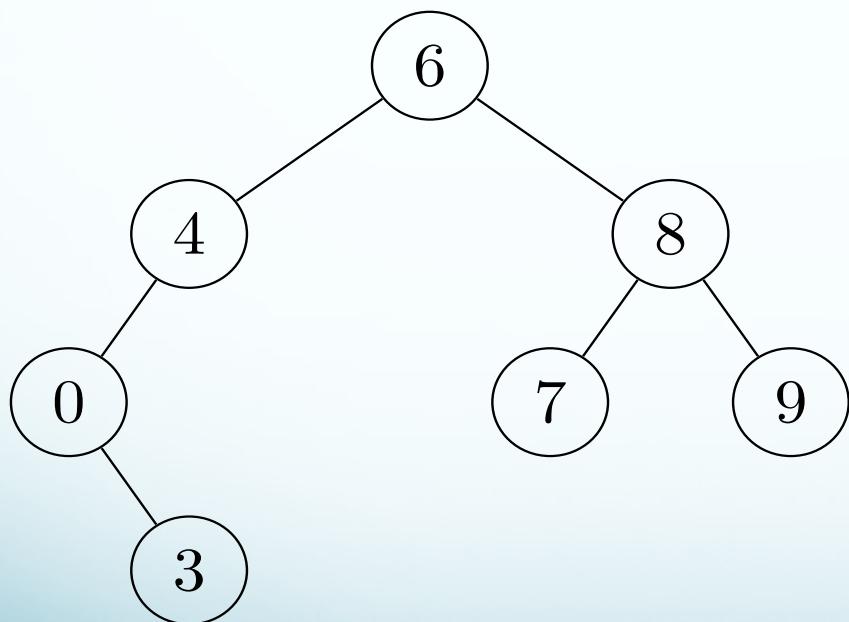


# Group/Individual work

Questions 6.2, 6.3.

## Q 6.2: conventional traversal

Write the *inorder*, *preorder* and *postorder* traversals of the following binary tree:



**YOUR ANSWER:**

In-order:

?

Pre-order:

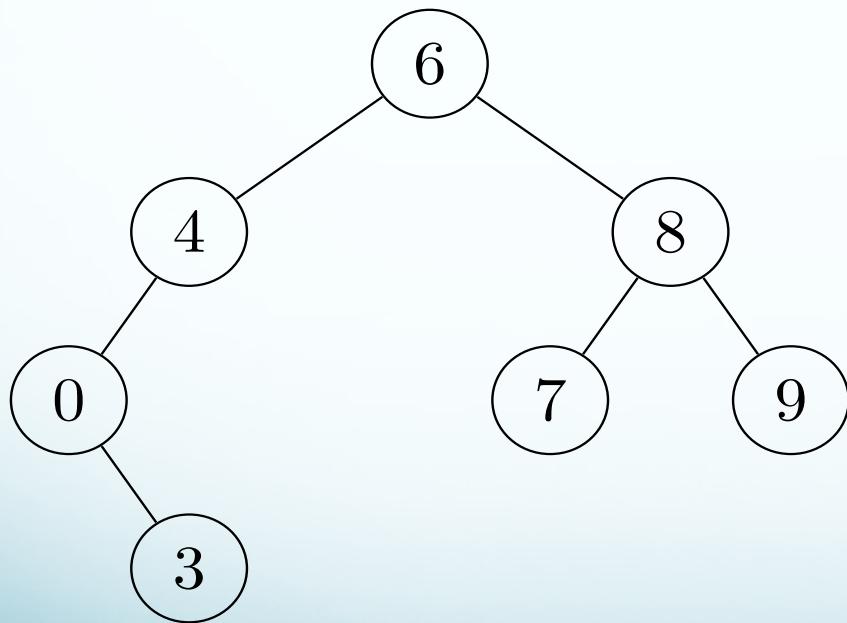
?

Post-order:

?

## Q 6.2: conventional traversal

Write the *inorder*, *preorder* and *postorder* traversals of the following binary tree:



**YOUR ANSWER:**

In-order:

? 0 3 4 6 7 8 9

Pre-order:

? 6 4 0 3 8 7 9

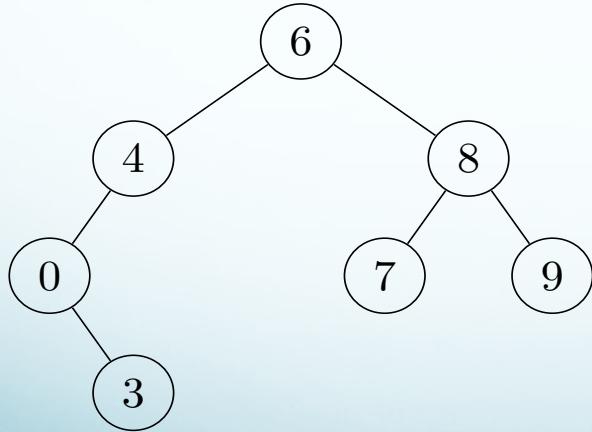
Post-order:

? 3 0 4 7 9 8 6

## Q 6.3: level-order traversal

Level-order: visit level-by-level, left-to-right, starting from the root (which is in 0-th level).

- a) For the tree below, what's the visited order?
- b) Write the level-order pseudo-code.



## YOUR ANSWER:

- a) Level-order: ???

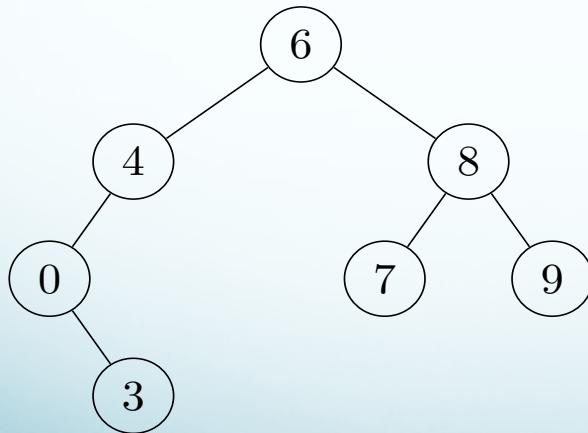
b)

```
// level order traversal for binary tree T
// supposing a non-empty tree has 3 components
//   T.root, T.left, T.right
function LEVELORDER(T)
    Q := empty queue
    INJECT(Q, T)      // =ENQUEUE
    while Q ≠ ∅ do
        t := EJECT(Q) // =DEQUEUE
        // visit root t (t.key)
        if t.left not empty then
            INJECT(Q, t.left) // =ENQUEUE
        if t.right not empty then
            INJECT(Q, t.right) // =ENQUEUE
```

## Q 6.3: level-order traversal

Level-order: visit level-by-level, left-to-right, starting from the root (which is in 0-th level).

- a) For the tree below, what's the visited order?
- b) Write the level-order pseudo-code.



## YOUR ANSWER:

- a) Level-order: 6 4 8 0 7 9 3

b)

```
// level order traversal for binary tree T
// supposing a non-empty tree has 3 components
//     T.root, T.left, T.right

function LEVELORDER(T)
    Q := empty queue
    INJECT(Q, T)                  // = ENQUEUE
    while Q ≠ ∅ do
        t := EJECT(Q)             // = DEQUEUE
        // visit root node t, ie. processing t.data
        if t.left is not NULL then
            INJECT(Q, t.left)
        if t.right is not NULL then
            INJECT(Q, t.right)
```

# LAB

Questions on Dijkstra's and BFS?

Assignment 1:

- do assignment 1 if not yet done
- make sure that you can finish on time
- note that you can discuss general problems with your friends  
but please do not reveal or show your solution and code

OR:

- do not-yet-done problems (if any) of previous workshops/lab/lectures