Big O?

```
Function(array, length)
    sum = 0
    for i = 0 to length - 1
        sum = sum + array[i]
    end for
    return sum/length - 1
end Function
```

A) $O(1)$
B) $O(n)$
C) $O(\log_2 n)$
D) $O(n + 3)$
E) $O(2^n)$

Big O?

```
Function(array, length)
    N = (length-1)
    for i from 1 to N
        for j from 0 to N - 1
                if a[j] > a[j + 1]
                    swap( a[j], a[j + 1] )
            endif
        end for
    end for
end func
```

A) $O(1)$
B) $O(n)$
C) $O(\log_2 n)$
D) $O(n^2)$
E) $O(2^n)$

What is the output if we call the function below as: func(2)

```
func(num)
    if (num == 0)
        return 0
    else
        result = (num*2)  +
                func(num-1)
        return result
```

A) 0
B) 3
C) 6
D) 2 1 0
E) 4 2 0

Big O?

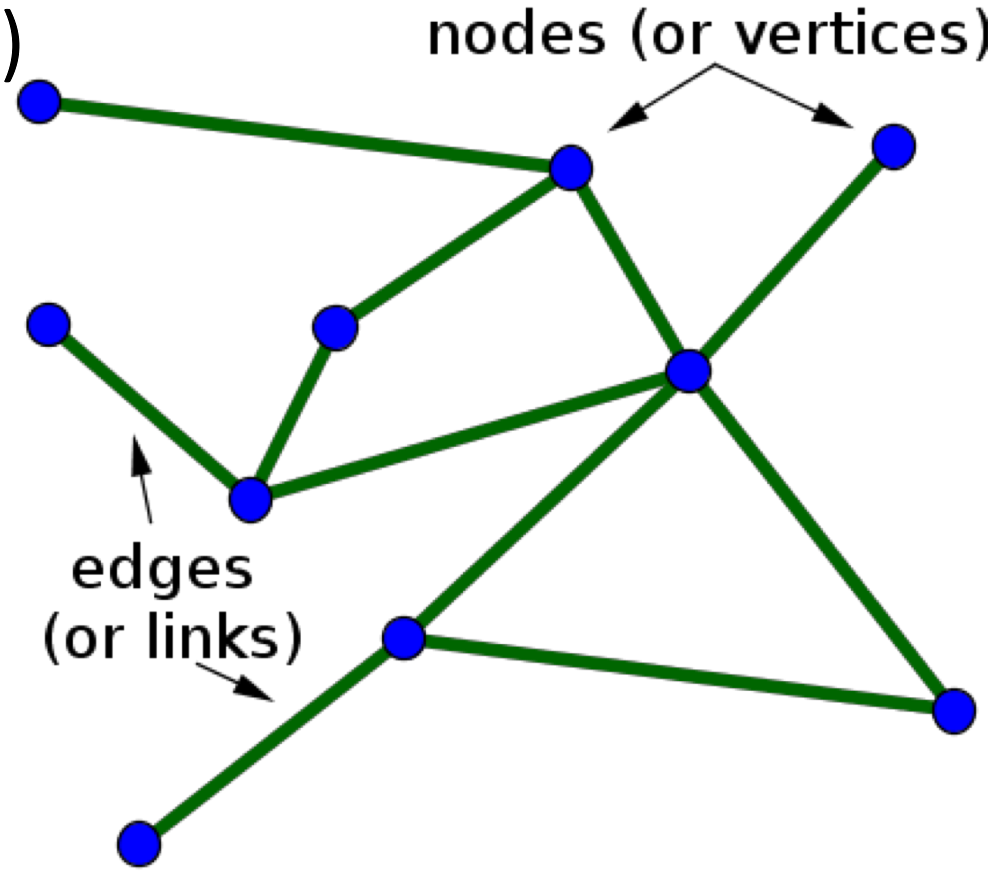
```
func(num)
    if (num == 0)
        return 0
    else
        result = (num*2)  +
                   func(num-1)
        return result
```

A) $O(1)$
B) $O(n)$
C) $O(\log_2 n)$
D) $O(n^2)$
E) $O(2^n)$

Graph Intro

Graph

- A set of vertices (nodes)
- A set of edges (links)



Graph terms…

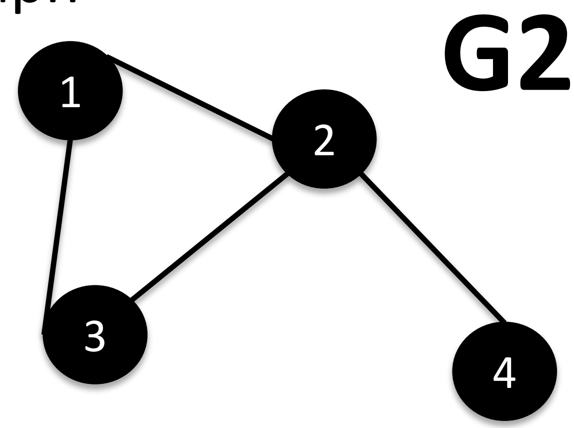
- Vertex set is the list of vertices in the graph

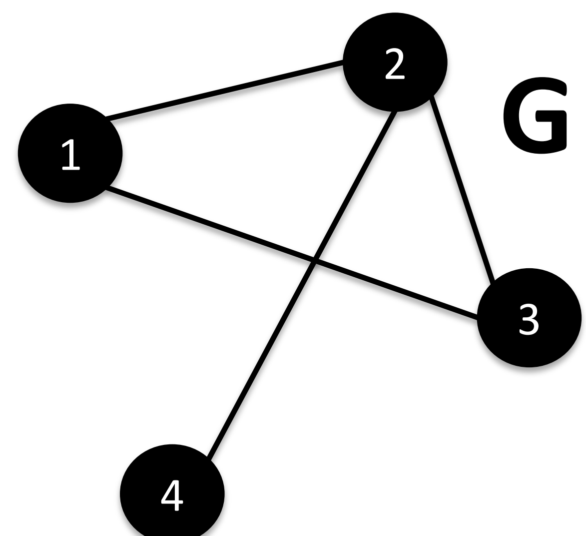
  $V_G = \{1, 2, 3, 4\}$
- Edge set is the list of connected vertices in the graph

  $E_G = \{ \{1,2\}, \{2,3\}, \{1,3\}, \{2,4\} \}$
- Cardinality is the # of vertices

  $|G| = 4$
- Degree of vertex is the # of edges coming out of it

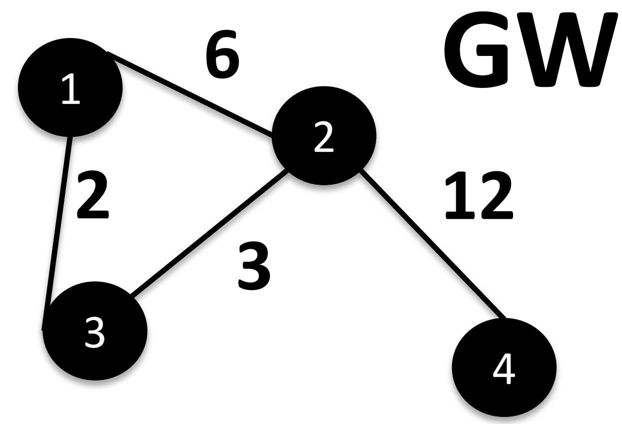
  $deg(1) = 2$
  $deg(4) = 1$
- Isomorphic graphs
  - Equivalent vertex set and edge sets
  - $V_G == V_{G2}$ and $E_G == E_{G2}$

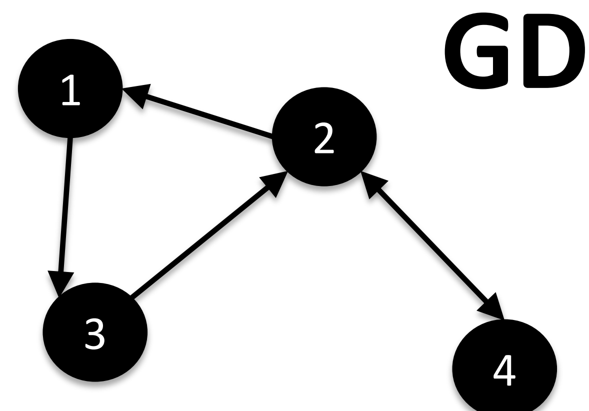




Weighted Graphs

- Weighted graph
  - associates a label (weight) with every edge in the graph. Weights are usually real numbers.
- Directed graph
  - Contains ordered pair of endvertices that can be represented graphically as an arrow drawn between the endvertices.





Clicker

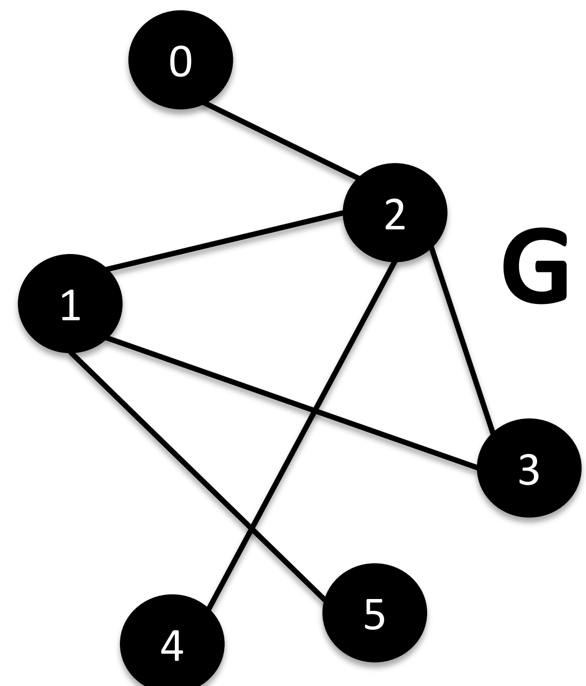
- What is the cardinality of G? |G|

A. 0

B. 3

C. 5

D. 6



Clicker

- What is the degree of vertex 2 in G? degree(2)

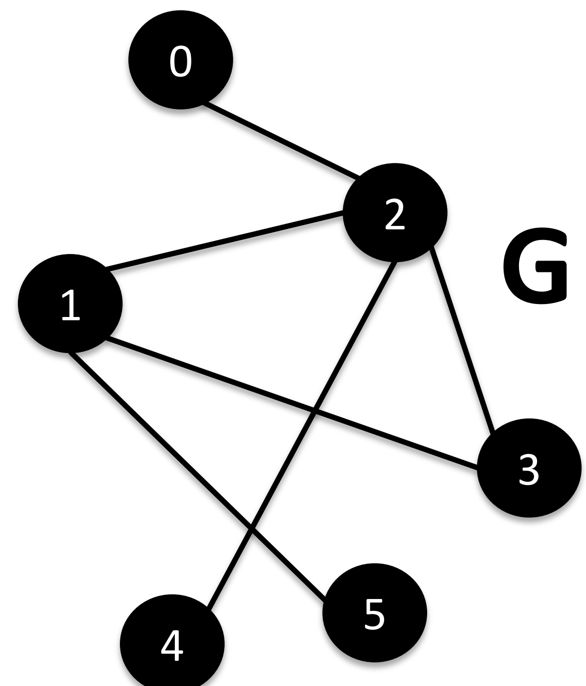
A. 0

B. 1

C. 0 1 3 4

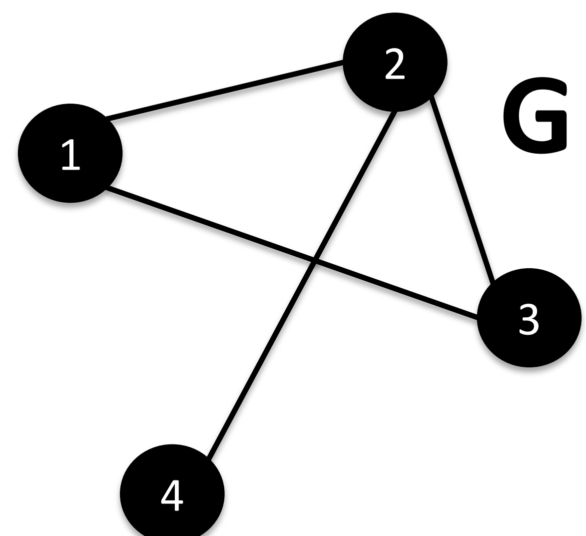
D. 4

E. 5



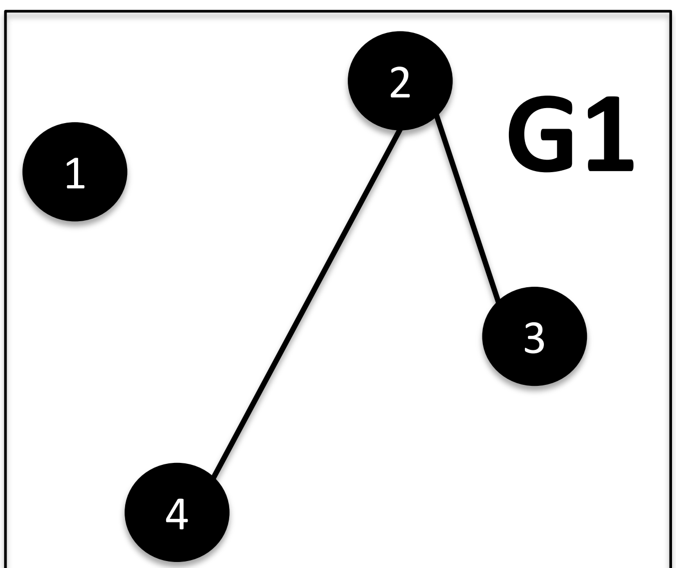
Cyclic graphs

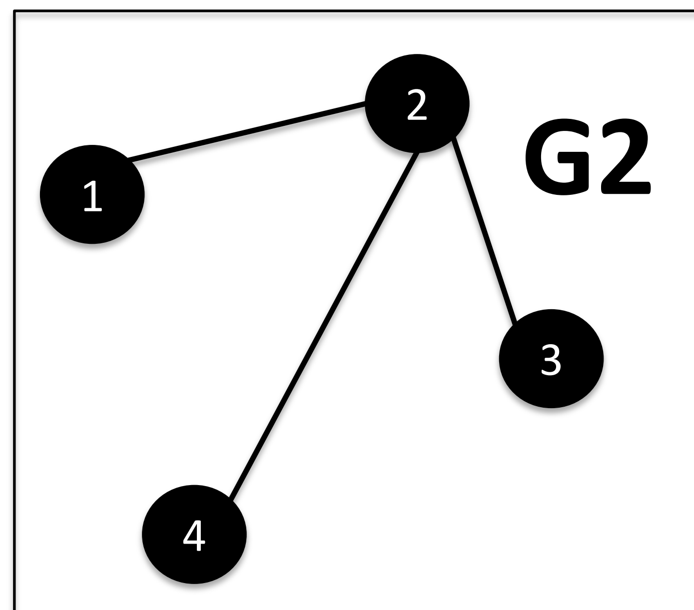
- A graph that contains at least one cycle
- A cycle is some number of vertices connected in a closed chain.



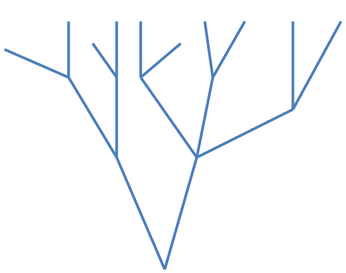
acyclic graphs

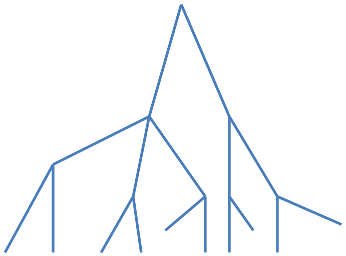
- A graph that contains no cycles (G1 and G2)
- If fully connected – called a tree (G2)
- If no cycles but not fully connected – called a forest (G1)





Trees



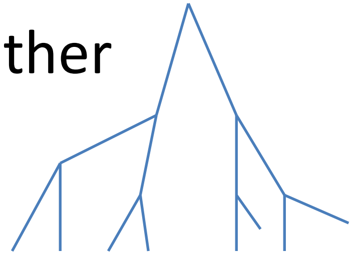


General tree

- A general tree T is a set of one or more nodes such that T is partitioned into disjoint subsets:
  - A single node r, the root
  - Sets that are general trees, called subtrees of r

Binary tree

- A binary tree is a set T of nodes such that either
  - T is empty, or
  - T is partitioned into three disjoint subsets:
    - A single node r, the root
    - Two possibly empty sets that are binary trees, called left and right subtrees of r



Tree Terminology

Level of a node, n, in aTree, T

- If n is the root of T, it is at level 1
- If n is not the root of T, its level is 1 greater than the level of its parent

Height of a tree, T

- If T is empty, its height is 0
- If T is not empty, its height is equal to the maximum level of its nodes

Terminology

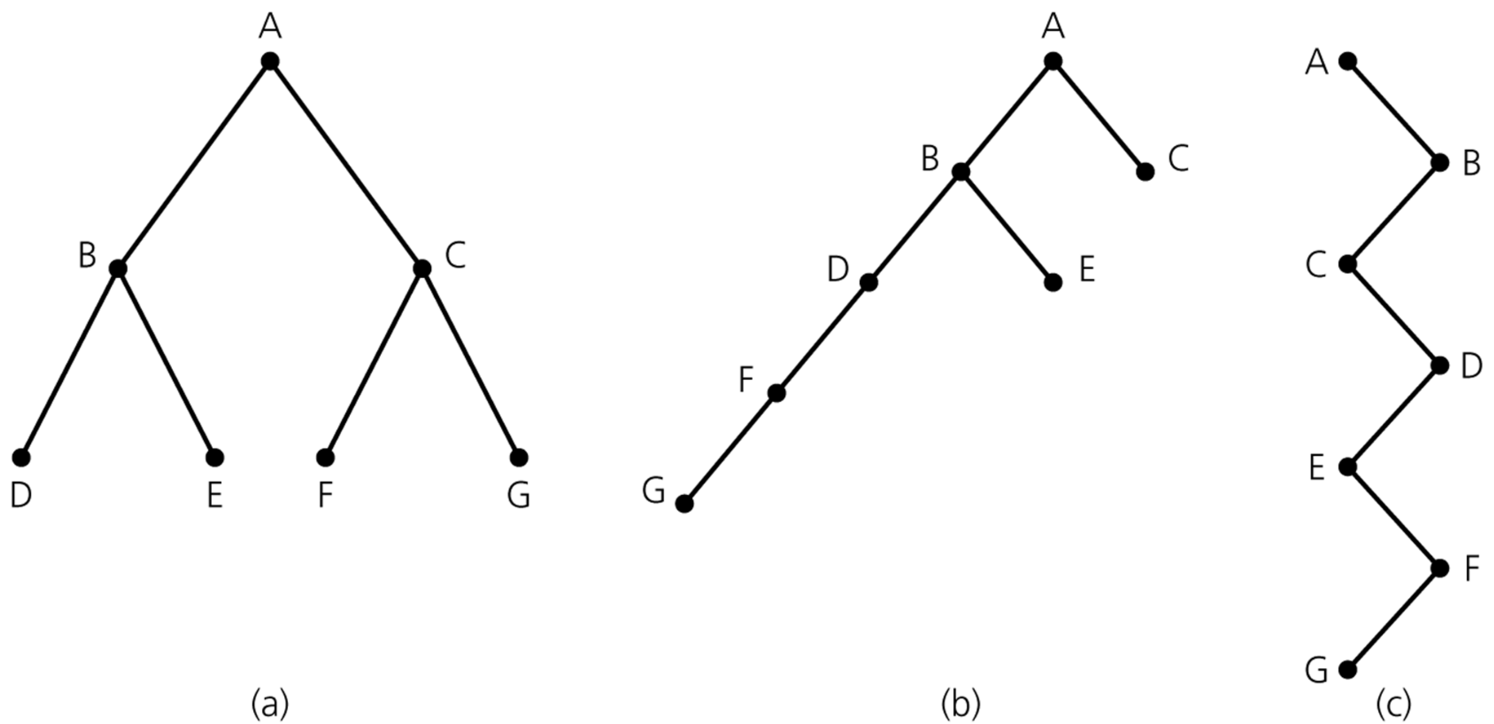


Figure 11-6

Binary trees with the same nodes but different heights

Definitions: Height

Height Binary Trees

- If T is empty, the height of T is 0
- If T is not empty, the height of T is
  1 + the maximum height of T's left and right subtrees

Example:
A binary tree, height 3

Observe: This is a *Recursive Definition!!*

Clicker

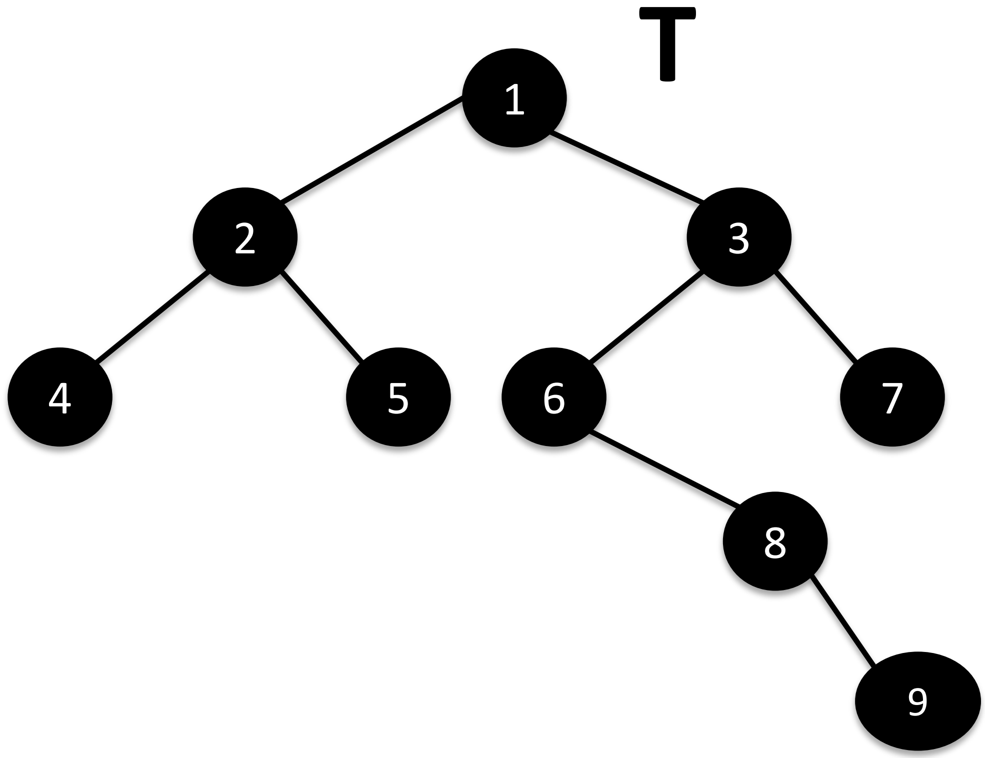
- What is the height of graph T:

A. 0

B. 3

C. 5

D. 9



Clicker

- What is the height of graph T:

A. 0

B. 3

C. 5

D. 9

T

Clicker

- What level is node 6 on in graph T?:

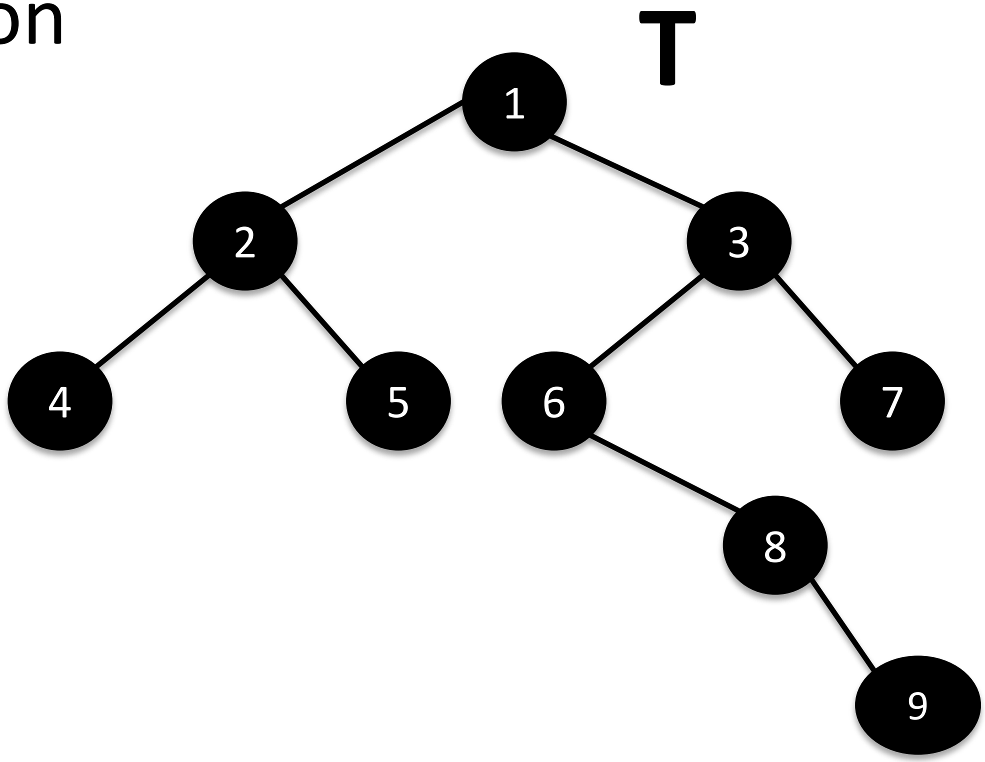
A. 0

B. 2

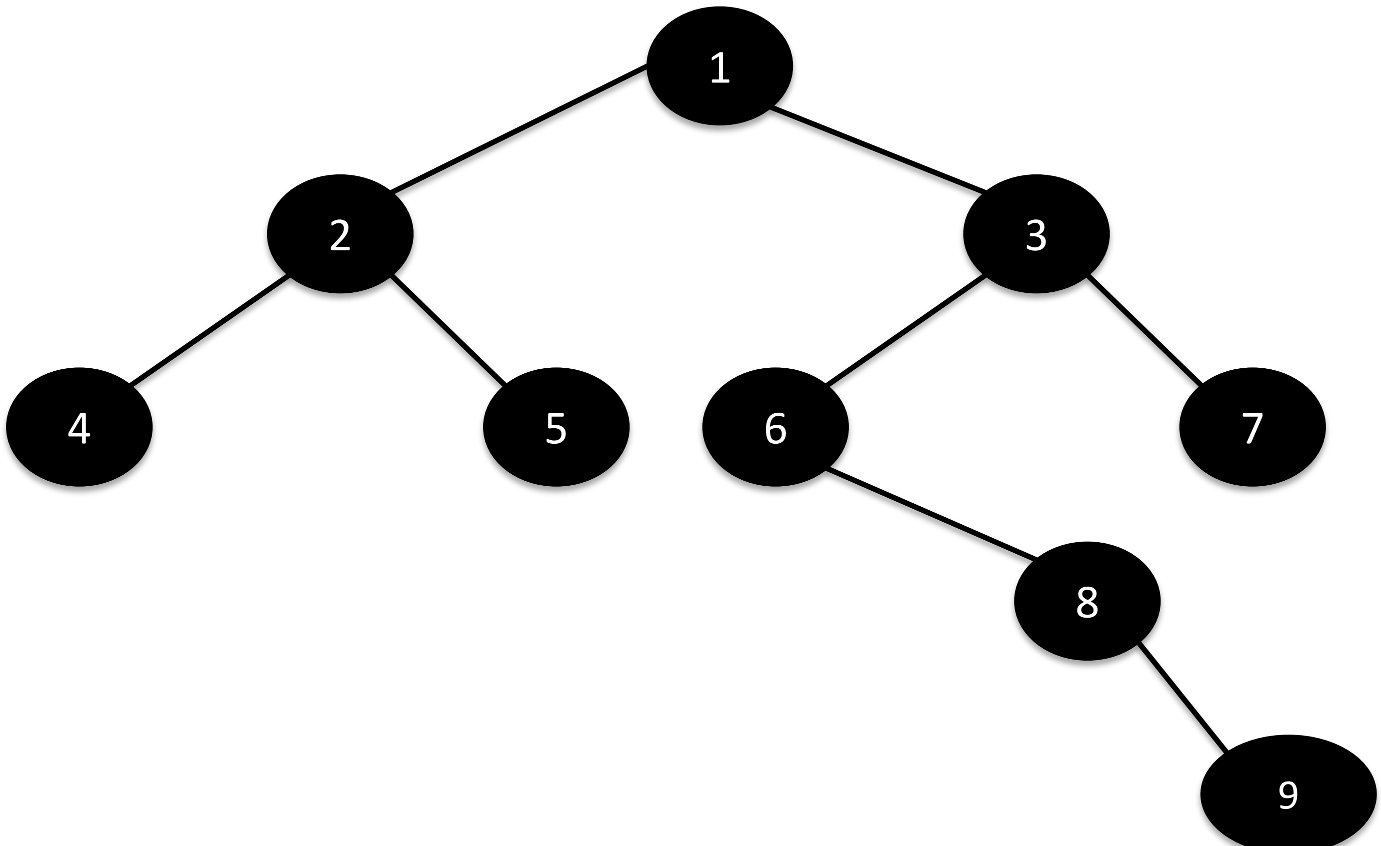
C. 3

D. 5

E. 6



T



Exercise

- Write the pseudocode for a recursive function that will take a tree and will return the height of that tree

Recall our process:

- Input/output?
- Examples:
  - Simplest example of calling the function
  - A more complex example
- Edit the template
  - Rename function & data
  - Edit the basecase
  - Deal with one data piece
  - Update to smaller problem for recursive call

HINT: - if you have more than one smaller problem, you can make a recursive call on each of them

```
function(data)
    if (smallestPossibleProblem? data)
        the simple answer
        return ...
    else
        first part of data ...
        function(smallerProblem(data))
        return ...
```

Solution

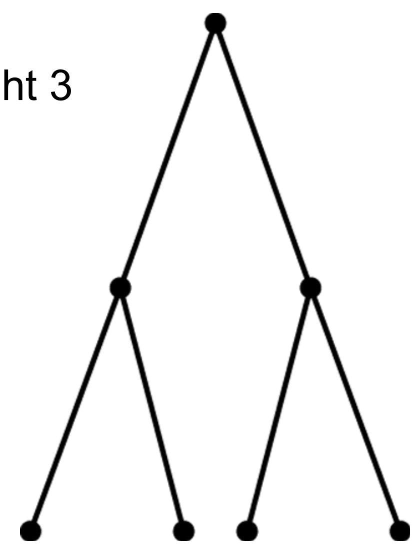
```
height(tree)
    if(tree is empty)
        return 0
    else
        heightLeft = height(tree's left subtree)
        heightRight = height(tree's right subtree)
        height = 1 + biggest (heightLeft, heightRight)
        return height
```

Definitions: Full, Complete, Balanced

Full Binary Trees

- If T is empty, T is a *full* binary tree of height 0
- If T is not empty and has height h > 0, T is a full binary tree if its root's subtrees are both full binary trees of height h – 1

Example:
A full binary tree, height 3



Exercise

- Write the pseudocode for a recursive function that will take a tree and will return true if the tree is full and false otherwise

Recall our process:

- Input/output?
- Examples:
  - Simplest example of calling the function
  - A more complex example
- Edit the template
  - Rename function & data
  - Edit the basecase
  - Deal with one data piece
  - Update to smaller problem for recursive call

HINT: - if you have more than one smaller problem, you can make a recursive call on each of them

```
function(data)
    if (smallestPossibleProblem? data)
        the simple answer
        return ...
    else
        first part of data ...
        function(smallerProblem(data))
        return ...
```

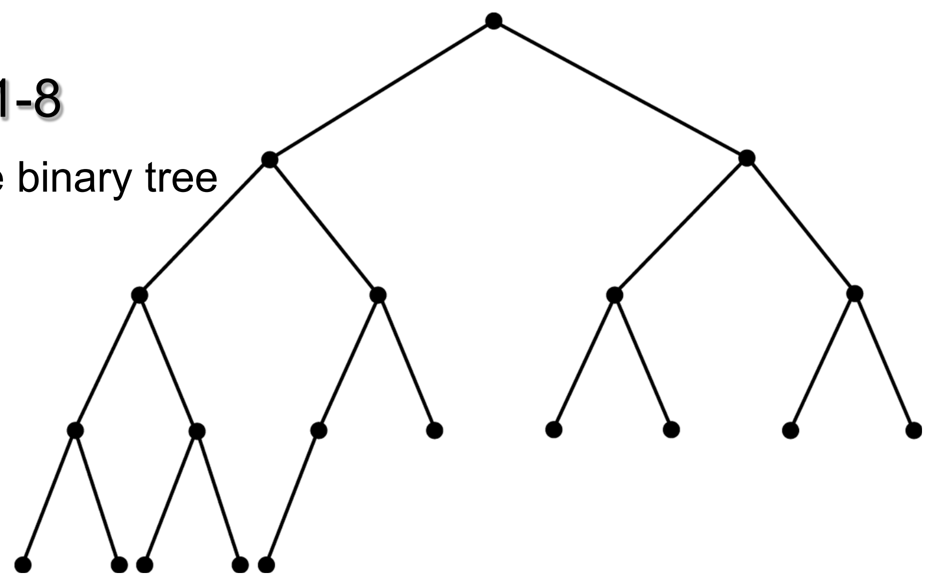
Definitions: Full, Complete, Balanced

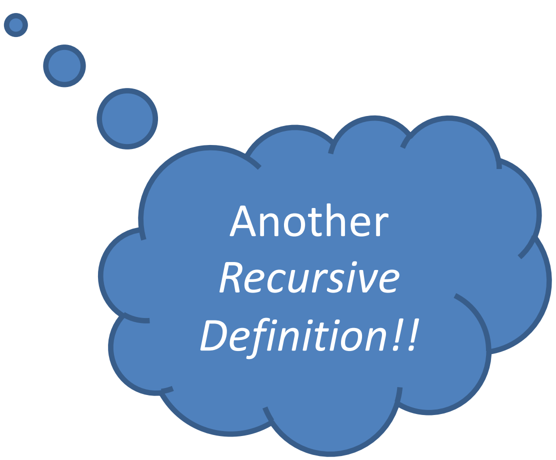
Complete binary trees

- A binary tree T of height h is *complete* if
  - All nodes at level h – 2 and above have two children each, and
  - When a node at level h – 1 has children, all nodes to its left at the same level have two children each, and
  - When a node at level h – 1 has one child, it is a left child

Figure 11-8

A complete binary tree





Definitions: Full, Complete, Balanced

Balanced binary trees

- A binary tree is *balanced* if the height of any node's right subtree differs from the height of the node's left subtree by no more than 1

Full binary trees are complete

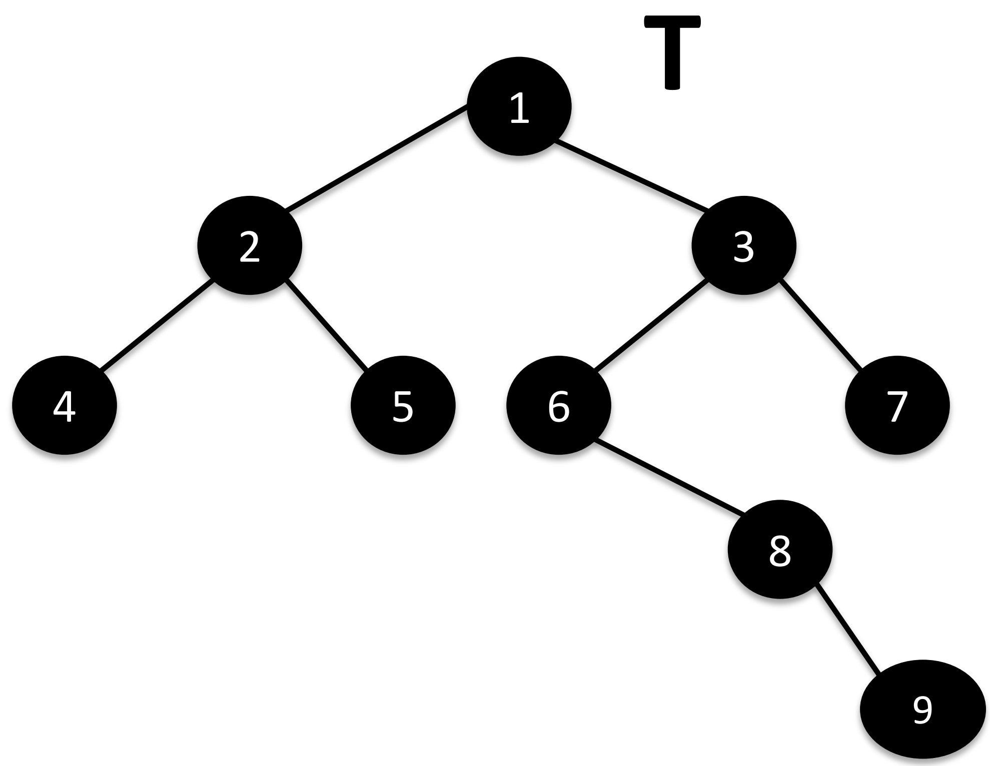
Complete binary trees are balanced

Clicker

- Is this tree full?:

A. Yes

B. No

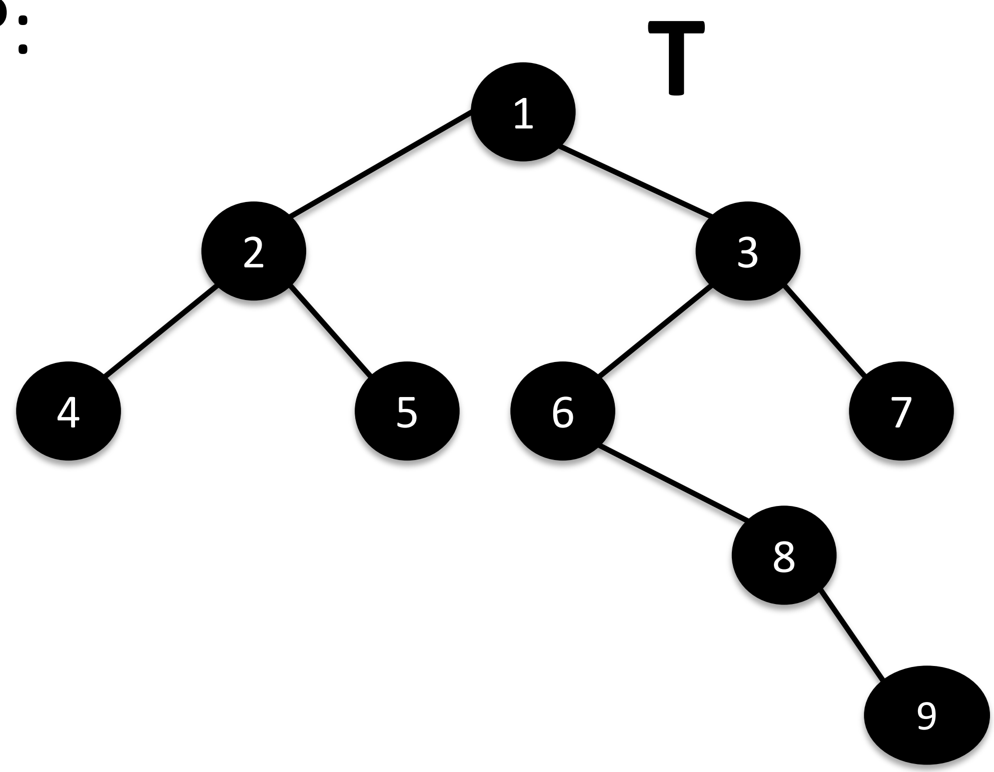


Clicker

- Is this tree complete?:

A. Yes

B. No

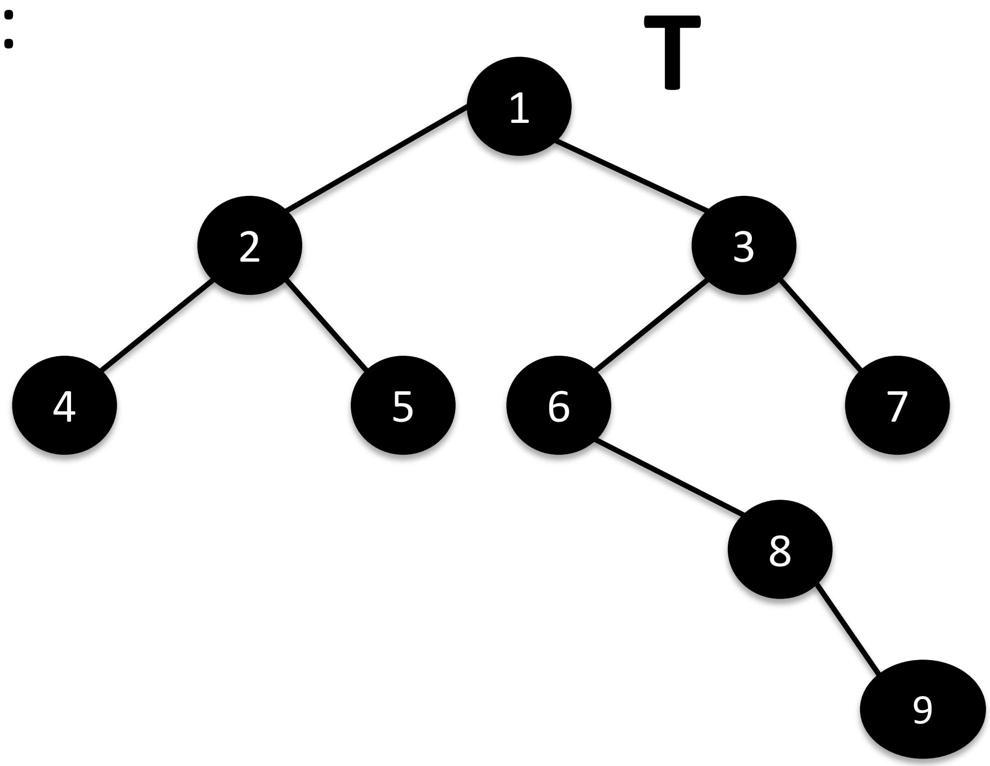


Clicker

- Is this tree balanced?:

A. Yes

B. No

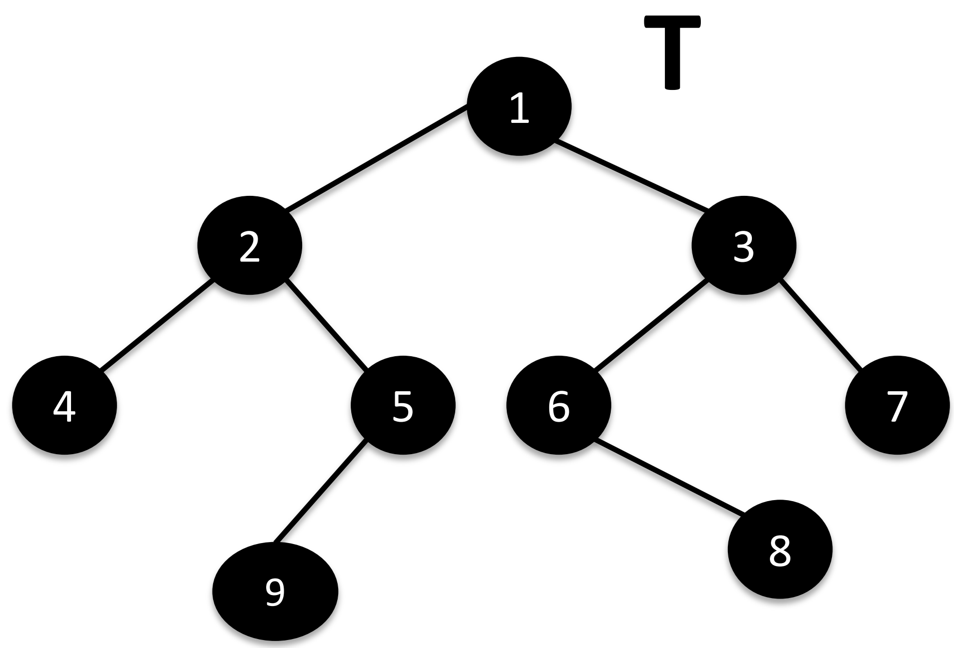


Clicker

- Is this tree full?:

A. Yes

B. No

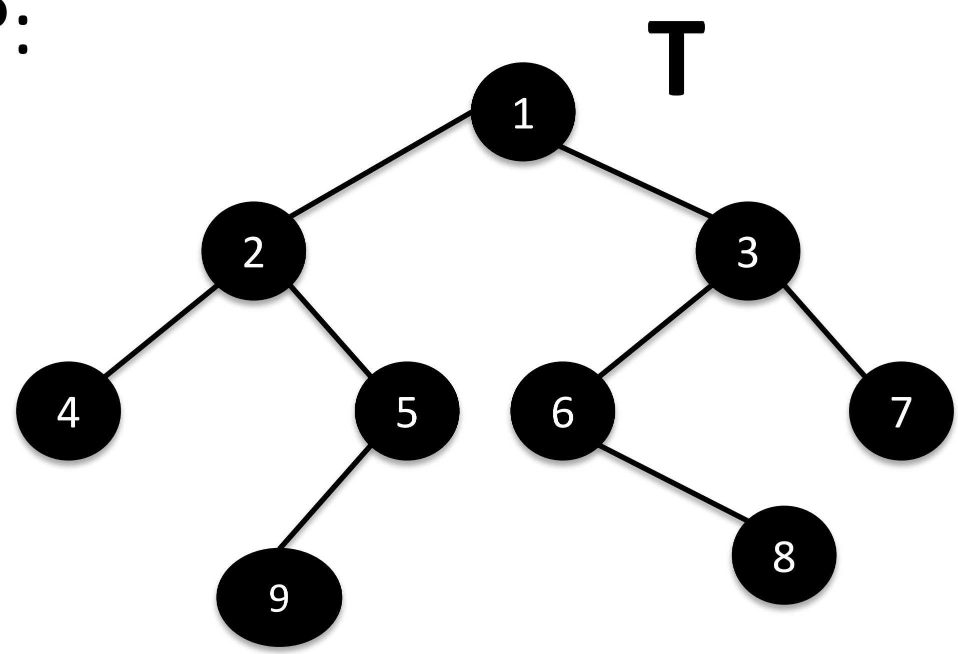


Clicker

- Is this tree complete?:

A. Yes

B. No

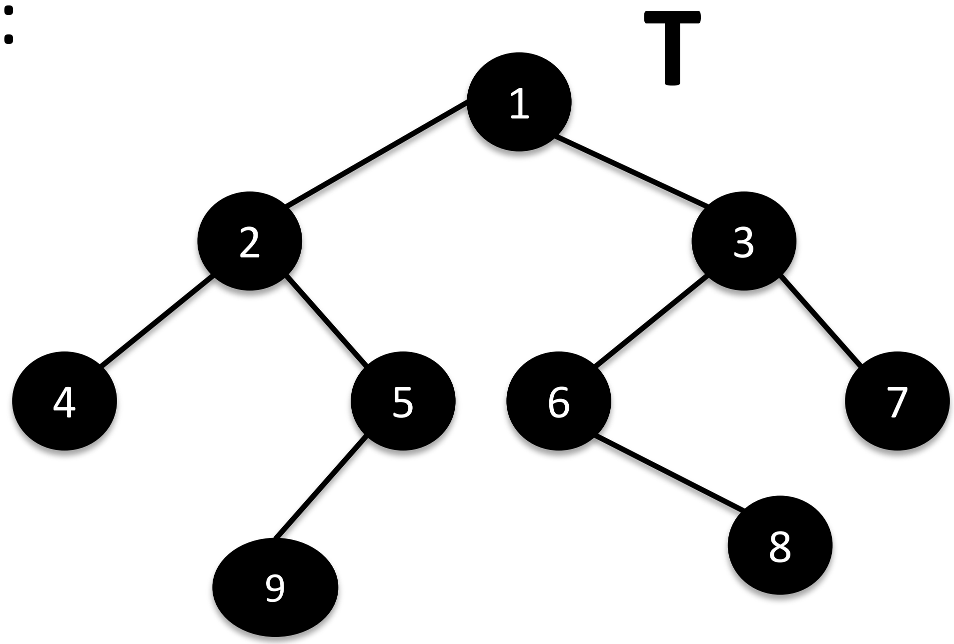


Clicker

- Is this tree balanced?:

A. Yes

B. No

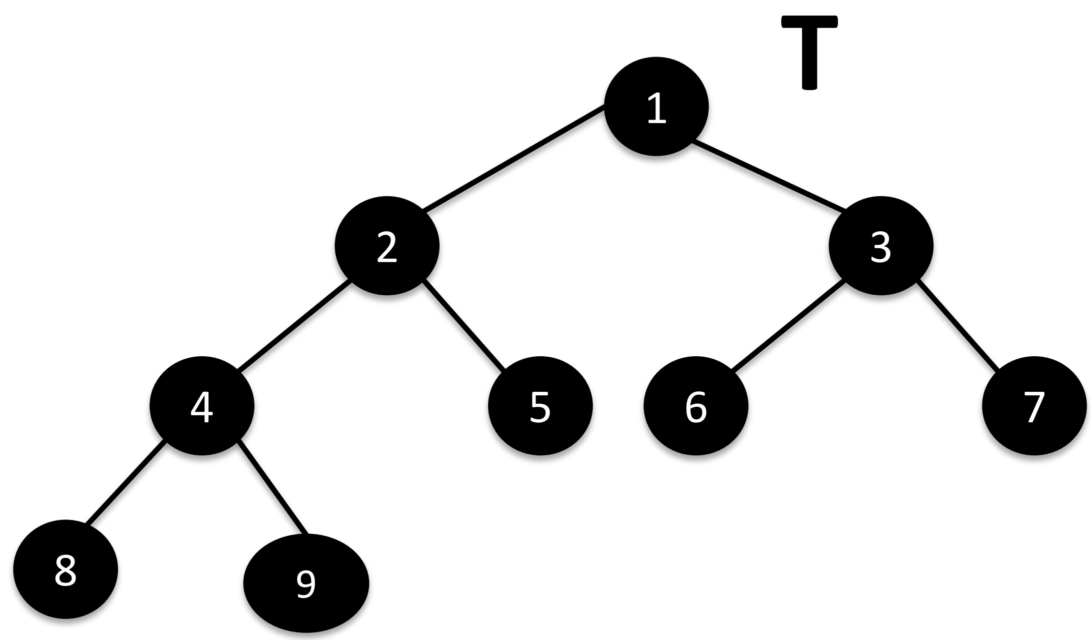


Clicker

- Is this tree full?:

A. Yes

B. No

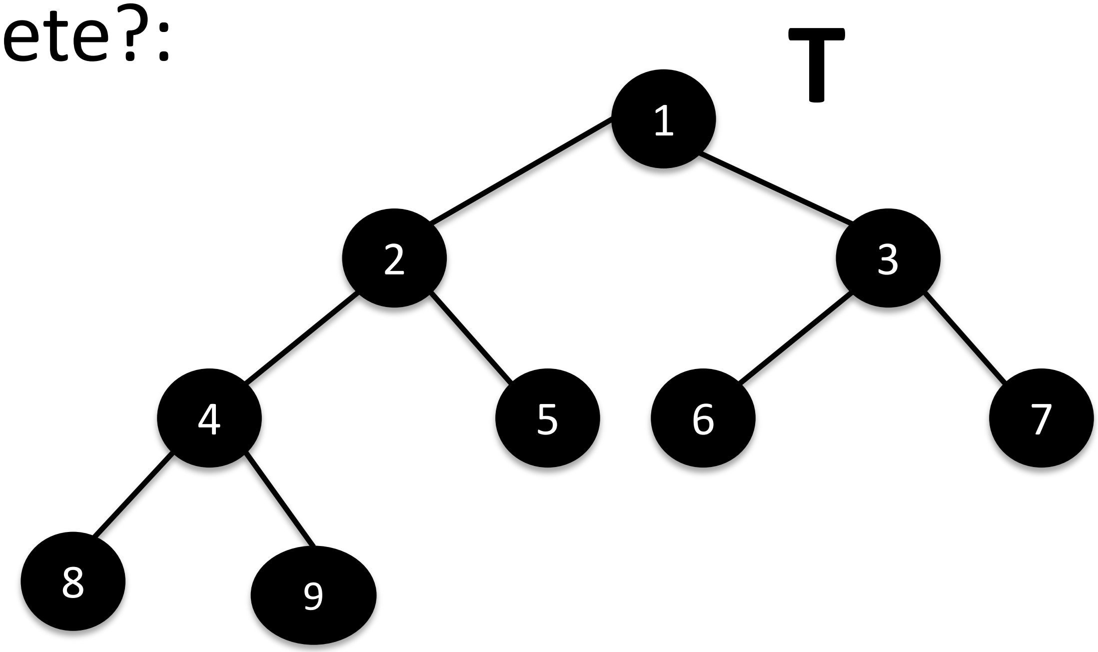


Clicker

- Is this tree complete?:

A. Yes

B. No

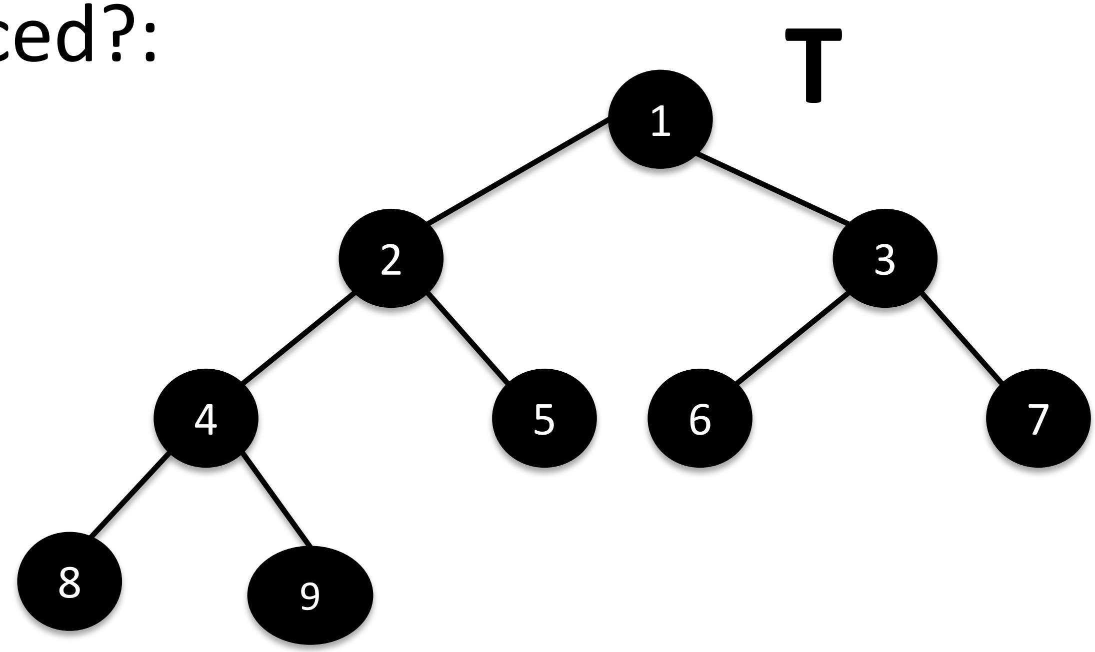


Clicker

- Is this tree balanced?:

A. Yes

B. No

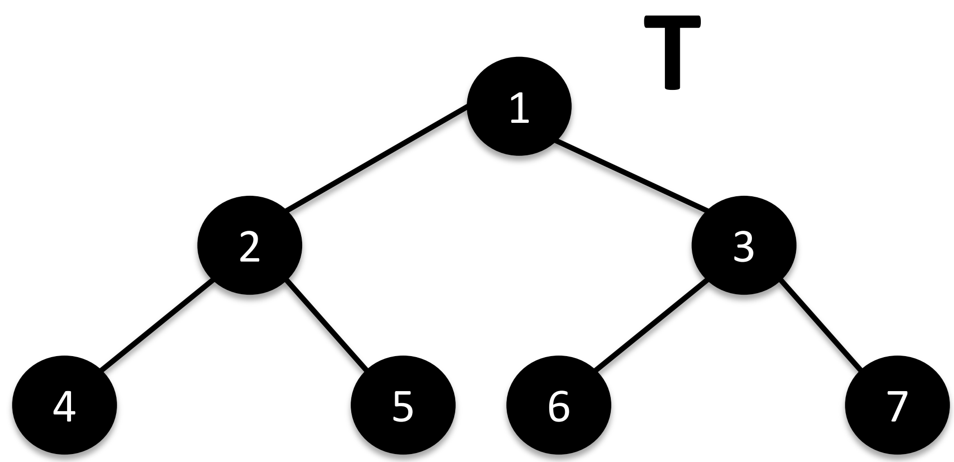


Clicker

- Is this tree full?:

A. Yes

B. No

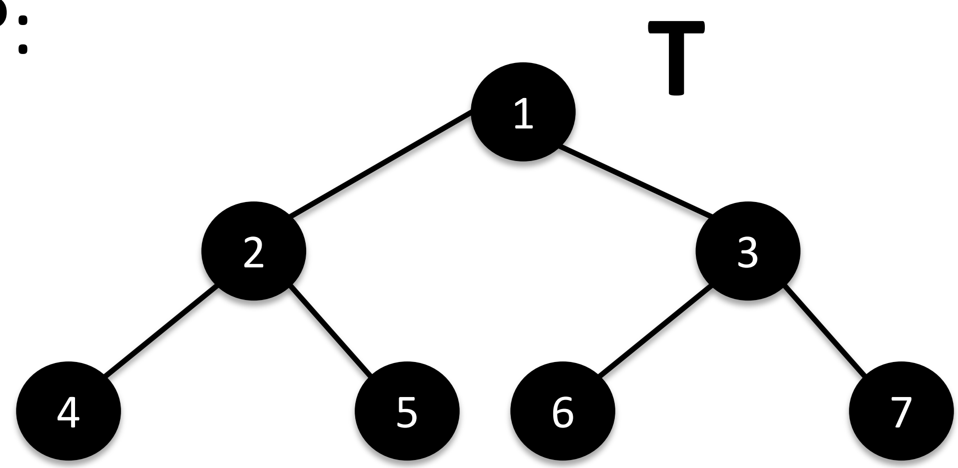


Clicker

- Is this tree complete?:

A. Yes

B. No



Clicker

- Is this tree balanced?:

A. Yes

B. No