Part 1: Recursion

1. [MT2 1.a] Provide the Python code for an implementation of such an array reversal function.

```
def reverseArray(A, i, j):
        # Input: An array A and nonnegative integer indices i and j
        # Output: The reversal of the elements in A starting at
index i and ending at j
        if i >= j:
                return
        # swap A[i], A[j]
        A[i], A[j] = A[j], A[i]
        # recursive call
        reverseArray(A, i + 1, j - 1)
```

2. [MT2 1.b] How many times times is `reverseAarray` called if the input array is `A == [1,2,3,4,5,6]`? List the individual calls! For each call, show the values of A, i, and j.

```
reverseArray([1,2,3,4,5,6], 0, 5)
reverseArray([6,2,3,4,5,1], 1, 4)
reverseArray([6,5,3,4,2,1], 2, 3)
reverseArray([6,5,4,3,2,1], 3, 2)
```

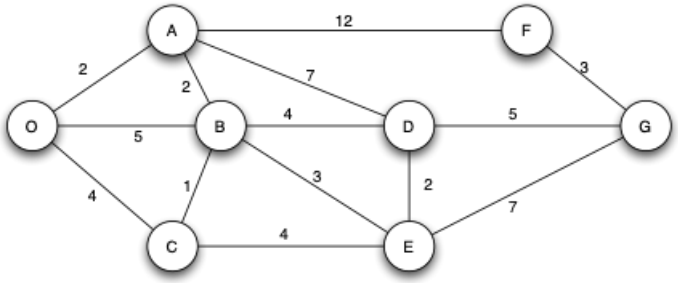
3. \[MT2 1.c\] How many times is reverse array called for the

reversal of an array of length _n_ (assuming _n_ is an even number!!)?
n/2 + 1

4. \[MT2 1.d\] Write a method that recursively calculates the factorial for n, which is defined as $n! = n * (n-1)!$ (Factorial value of 0 is 1)

```python
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n - 1)
```

Part2: Dijkstra's Algorithm



1. [MT2 2.a] Consider the network shown above. With the indicated link costs, use Djikstra's shortest-path algorithm to compute the table of shortest paths from O to all other network nodes. Show how the algorithm works by computing a table below. Use column N' for *"all visited nodes in current step"*, and each row for *"distance and predecessor of each destination node once a new node is visited"*.

> Practice on this interactive webapp: link

2. [MT2 2.b] From these results, show the least cost path from O to nodes $A \sim G$, and briefly describe (in a sentence) how you got that answer from your work in part 1). Solution from Dijkstra
3. [MT2 2.c] What are the distance vectors in nodes B and D ? _Note: you do not have to run the distance vector algorithm; you should be able to compute distance vectors by inspection.

$$D_B(O) = 4, D_B(A) = 2, \ldots$$

4. [MT1 2.c] Based on the paths from O to all other nodes you identified in 3, draw the shortest path tree/graph. Shortest Path Tree

Part 3: Binary Trees

Answer the following questions regarding binary trees. Note that these binary trees are **not** binary search trees and may contain nodes with any value at any location. The node of the tree is defined as:

```
class Node:
    def __init__(self, k, v, l=None, r=None):
        self.key   = k
        self.value = v
```

```
        self.leftChild  = l
        self.rightChild = r
```

Assume the tree is defined as the following and properly initialized:

```
class Tree:
    def __init__(self,r=None):
    self.root = r
```

1. [MT1 3.a] Write a recursive method that returns the height of a binary tree. Assume the method is called with `treeHeight(t.root)`, where `t` is the tree.

```
def treeHeight(node):
    if node.leftChild is None and node.rightChild is None:
        # leaf node
        return 0
    if node.leftChild is not None:
        left_height = treeHeight(node.leftChild)
    else:
        left_height = 0
    if node.rightChild is not None:
        right_height = treeHeight(node.rightChild)
    else:
        right_height = 0
    return max(left_height, right_height) + 1
```

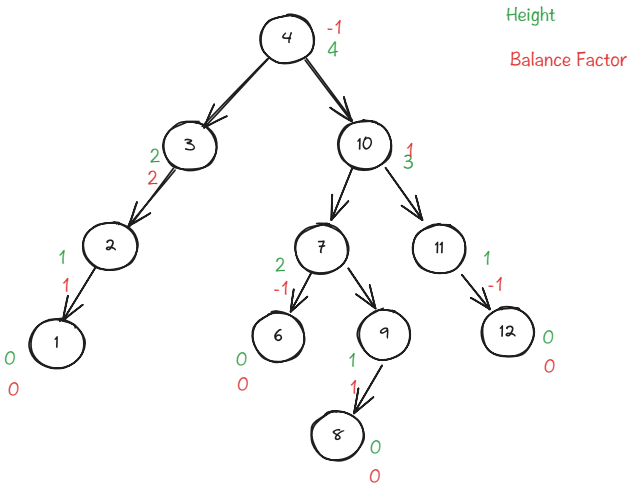
2. [MT1 3.b] Write a recursive method that returns the in-order traversal array of a binary tree. Assume the method is called with `inOrder(t.root)`, where `t` is the tree.

```
def inOrder(node):
        traversal_array = []
        if node.leftChild is not None:
                traversal_array += inOrder(node.leftChild)
        traversal_array.append(node)
        if node.rightChild is not None:
                traversal_array += inOrder(node.rightChild)
        return traversal_array
```

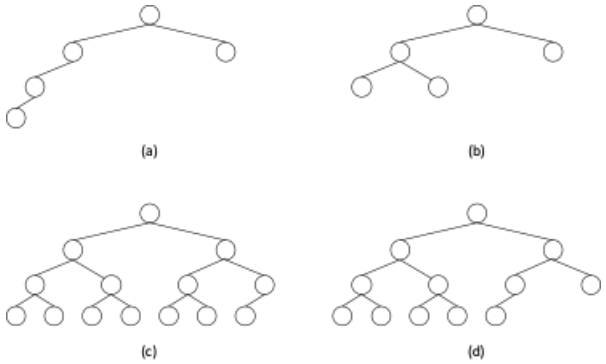
3. [MT1 3.c] Now consider a binary tree that is a BST, what is the property of the array return from an inOrder traversal of the BST?
   Sorted array

Part4: AVL trees

Answer the following questions regarding trees. Given the following AVL Tree



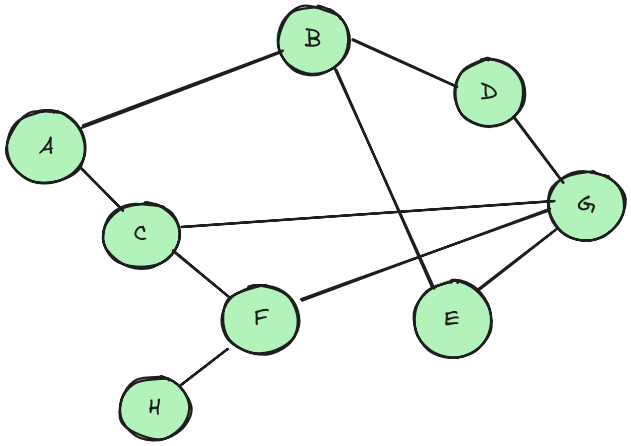
1. [MT2 4.a] Draw the resulting BST after 5 is removed, but before any rebalancing takes place. Label each node in the resulting tree with its balance factor. Replace a node with both children using an appropriate value from the node's left child. Tree
2. [MT2 4.b] What operation has to be performed (explain in words no code necessary) to find the vertex with which the top vertex has to be replaced?
3. [MT2 4.c] After removing the top node, is the tree balanced?
4. [MT2 4.d] Now rebalance the tree that results from (1). Draw a new tree for each rotation that occurs when rebalancing the AVL Tree (you only need to draw one tree that results from an RL or LR rotation). You do not need to label these trees with balance factors. Rebalance
5. [MT2 4.e] Which of the trees shown below is balanced and which is not?



6. [MT1 4.d] In a balanced tree, can there be a non-leaf node that is the only children (no sibling) for a node? Explain your answer! Non Sibling

Part 5: Graphs

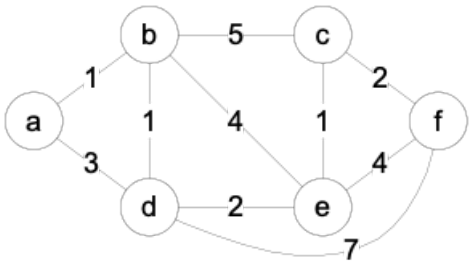
Answer the following question regarding graphs.



1. [MT2 5.a] List the order of nodes that are visited in the following graph when performing the state traversal. Assume that neighboring nodes are visited in alphabetical order. (8 points)
    1. Depth-first traversal starting at node A
    2. Depth-first traversal starting at node G
    3. Breadth-first traversal starting at node F

4. Breadth-first traversal starting at node B

2. [MT2 5.b] For the graph shown in the figure below, show the adjacency list and adjacency matrix representation of that graph. Using a linked list for the neighbors in the adjacency list representation.



Adjacency list:

Node ID	Neighbor list
a	(b,1) (d, 3)
b	(a, 1) (1, d) (4, e), (5, c)
c	
d	
e	
f	

Adjacency Matrix:

	a	b	c	d	e	f
a	0	1	∞	3	∞	∞
b						
c						
d						

	a	b	c	d	e	f
e						
f						

3. [MT2 5.c] Name two applications that would benefit from Prim's Spanning Tree Algorithm. Also, briefly explain the benefits of a tree generated by that algorithm.
4. [MT1 5.c] Now assume all links in the graph are bi-directional. Identify four loops that exist in the graph by listing the nodes that compose that loop.