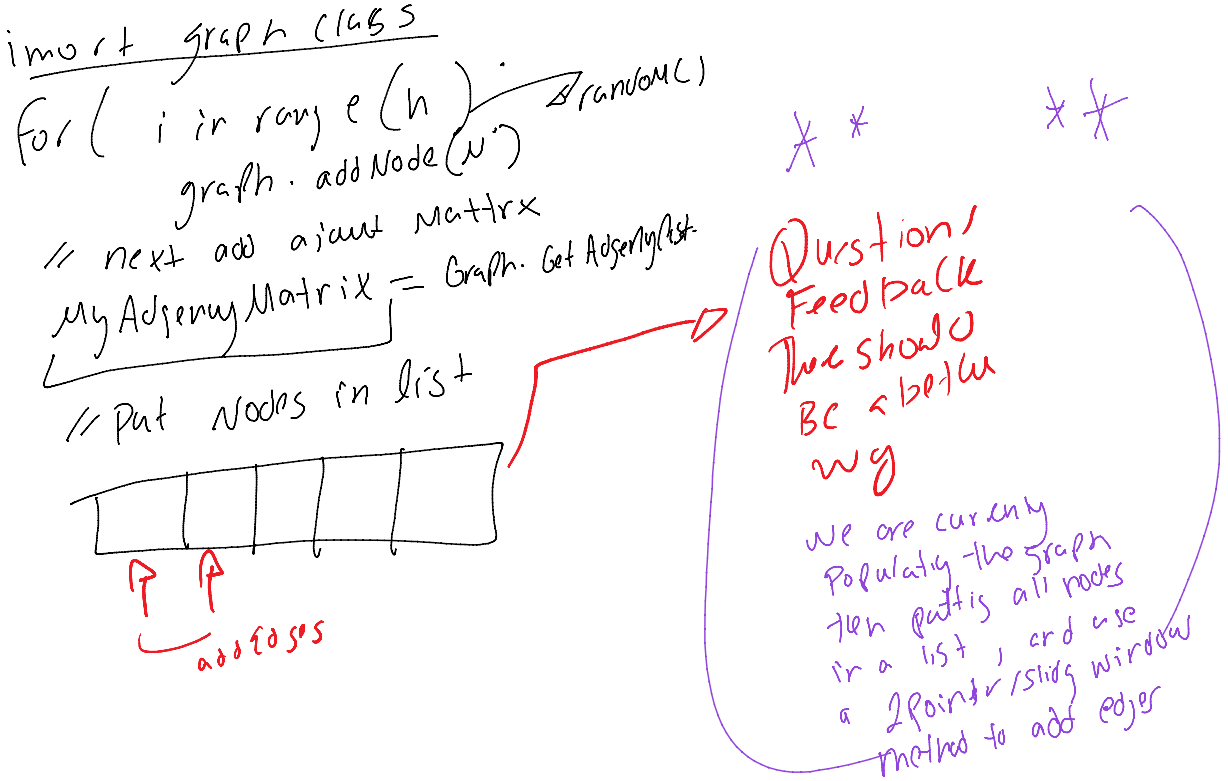
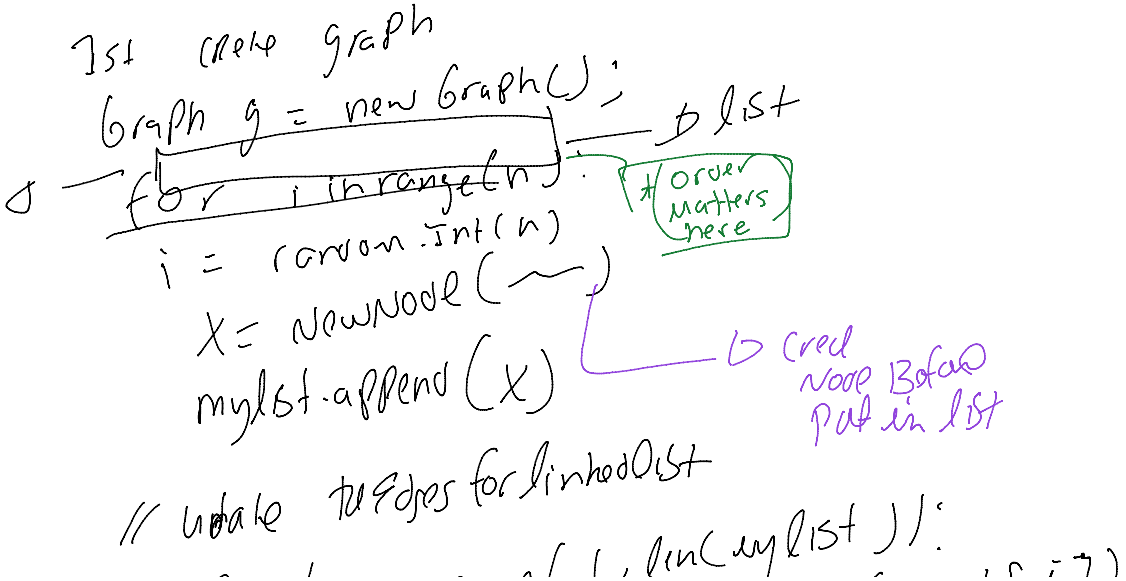
3 part b-d

Wednesday, March 25, 2020 6:34 PM

(2 points) (You must submit code for this question!) In your Main class, create a non-recursive method called Graph createRandomUnweightedGraphIter(int n) that creates n random nodes with randomly assigned unweighted, bidirectional edges. You should use some of the methods you implemented in part (a). Make sure you're either implementing an adjacency list or an adjacency matrix to keep track of your edges!



c) (2 points) (You must submit code for this question!) In your Main class, create a non-recursive method called Graph createLinkedList(int n) that creates a Graph with n nodes where each node only has an edge to the next node created. For example, if you create nodes 1, 2, and 3, Node 1 only has an edge to Node 2, and Node 2 only has an edge to Node 3.



// update the edges for ...

for (i in range(1, len(mylist)):
    myList[i-1].neighbors.append(mylist[i])

// Put Back in graph

for nodes in mylist:
    graph.add(nodes)

(3 points) (You must submit code for this question!) In a class called GraphSearch, implement ArrayList<Node> DFSRec(final Node start, final Node end), which recursively returns an ArrayList of the Nodes in the Graph in a valid Depth-First Search order. The first node in the array should be start and the last should be end. If no valid DFS path goes from start to end, return null.

make a helper func to create a helper func.

def DFS Rec int target value , mySet,)

myNeighbors = node.start.neighbors
if (node.val == target value) list.append(node)
if neighbor = null return 0
for(node in neighbors):
    if (node not in Set):
        set.add(node)
        arraylist.append(node)
        dfs(node, target, set)

if target not in Arraylist:
    return Null
else return arraylist