HW 3 Alg redo

1)

	A	B	C	D	E	F	G	H	I	J
ord	1	2	8	6	4	5	10	3	7	9
BFS layer	0	1	2	2	2	3	3	1	1	1

vFound = A

vO = B
vO = H
vO = I
vO = J

vFound = B

vO = D

vFound = H

vO = E

vFound = I

vFound = J

vO = C

vFound = C

vO = F
vO = G

2)

DFS

	A	B	C	D	E	F	G	H	I	J
ord	1	2	8	3	5	9	10	4	6	7
f-val	1	9	3	10	8	4	5	7	6	2

DFS(A)
DFS(B)
DFS(D)
DFS(H)
DFS(E)
DFS(G)
DFS(J)
DFS(C)
DFS(F)
DFS(I)

3)

a) diameter(graph):

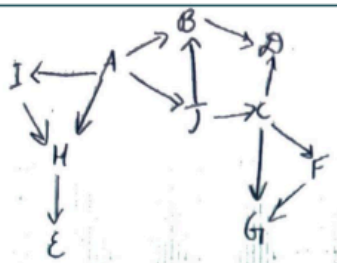
```
sizes = []
for vertex in graph:
    num = BFS_layer_counter(vertex)
    sizes.append(num)
return biggest(sizes)
```

b) BFS running time = $O(m+n)$
BFS_layer_counter running time is the same because it's BFS code w/ a counter for # of layers

As all other actions are done in constant time for a graph w/ n vertices, the runtime is $O(m+n) \cdot n = \boxed{O(nm+n^2)}$

c) BFS is run on every vertex in the graph such that the BFS also counts the number of layers for each vertex it's run from. Then the values are saved to an array and the biggest value is returned as it represents the max distance among all vertices pairs aka the graph's diameter.

4)



* directed graph from q's 1 and 2

A B C D E F G

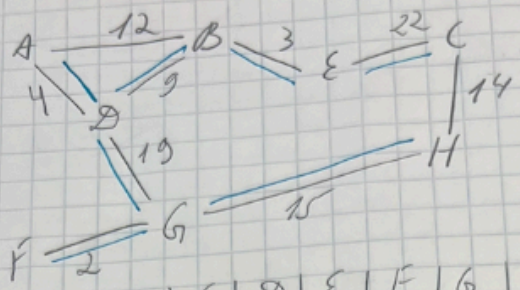
al = [[B, H, I, J], [D], [F, G], [ ], [ ], [G], [ ],
[E], [H], [B, C]]

H I J

worst case for computing in-degree (# of edges pointing to a vertex) of a given vertex: $O(n+m)$

5) To find if 2 vertices are connected in a graph represented by an adjacency matrix start w/ the first vertex and look for the edge that connects to the first unvisited vertex (aka the first 1 that connects to an unvisited vertex). Continue doing this until the second (desired) vertex is reached. If such a vertex can't be reached then the graph is disconnected. Worst case scenario is that all vertices need to be visited and each outgoing edge needs to be checked so it's $O(n) \cdot n = O(n^2)$

6)



A	B	C	D	E	F	G	H
4	9	34	0	12	21	19	34

iter	V	vOther	weight	length
1	D	A	4	(4)
		B	9	9
		G	19	19
2	A	B	12	16
	D	B	9	(9)
		G	19	19
3	B	E	3	(12)
	D	G	19	19
4	D	G	19	(19)
	E	C	22	34
5	G	F	2	(21)
	E	C	22	34
		H	15	34
6	G	H	15	(34)
	E	C	22	34
7	E	C	22	(34)
	H	C	14	48

* look @ every possible way to reach a vertex before circling shortest path val

* every iteration finds exactly one vertex

aka only one gets circled *