1. a)

Algorithm:

Let N be the total number of nodes in a graph and adj[N][N] as the adjacency matrix of the graph, where N can be either 0 or 1.

In the graph, there are six nodes we can consider to be represented by the YUCCA property. Starting with Node A, in order for the graph to be YUCCA, the matrix row for Node A needs to be all 1's except for the A column. This means that there should be five nodes marked with 1's, and thus representing five instances of outdegree N-1 and one instance of indegree 0.

1) Input the matrix.
2) Check if nodes are visible.
3) Start code as setting all nodes as not visited.
4) Code runs through the matrix.
    a) If there is a node and has not been transversed, the graph is not YUCCA. Return as false.
    b) If there is a node and has been transversed, the graph is YUCCA. Return true.
5) Based on step 4:
    a) False, print a statement to say the graph is not YUCCA.
    b) True, print a statement to say the graph is YUCCA.

Time Complexity:

For every node, we make N traversals for both calculating how many 1's appear in the matrix and checking if every "i" belongs to adj[i][node] = 0. This means that for each node, the time traversal is 2N, and for N nodes it is $2N^2$. Therefore, the time complexity of the algorithm is $O(n^2)$.

b) See separate YUCCA.cpp file for code, outputs below.

Input: Graph 1
{0, 1, 1, 0, 1},
{1, 0, 1, 0, 0},

{0, 1, 0, 0, 1},
{1, 1, 1, 0, 1},
{0, 1, 1, 0, 0}
Output: Not YUCCA

Input: Graph 2
{0, 1, 1, 0},
{1, 0, 1, 1},
{0, 0, 0, 0},
{1, 0, 1, 0}
Output: Not YUCCA

Input: Graph in Part A
Converted into adjacency matrix:

	A	B	C	D	E	F
A	0	1	1	1	1	1
B	0	0	1	0	0	0
C	0	1	0	1	0	0
D	0	0	0	0	1	0
E	0	0	0	0	0	0
F	0	0	0	0	1	0

∴
{0, 1, 1, 1, 1, 1},
{0, 0, 1, 0, 0, 0},
{0, 1, 0, 1, 0, 0},
{0, 0, 0, 0, 1, 0},
{0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 1, 0}

Output: Not YUCCA