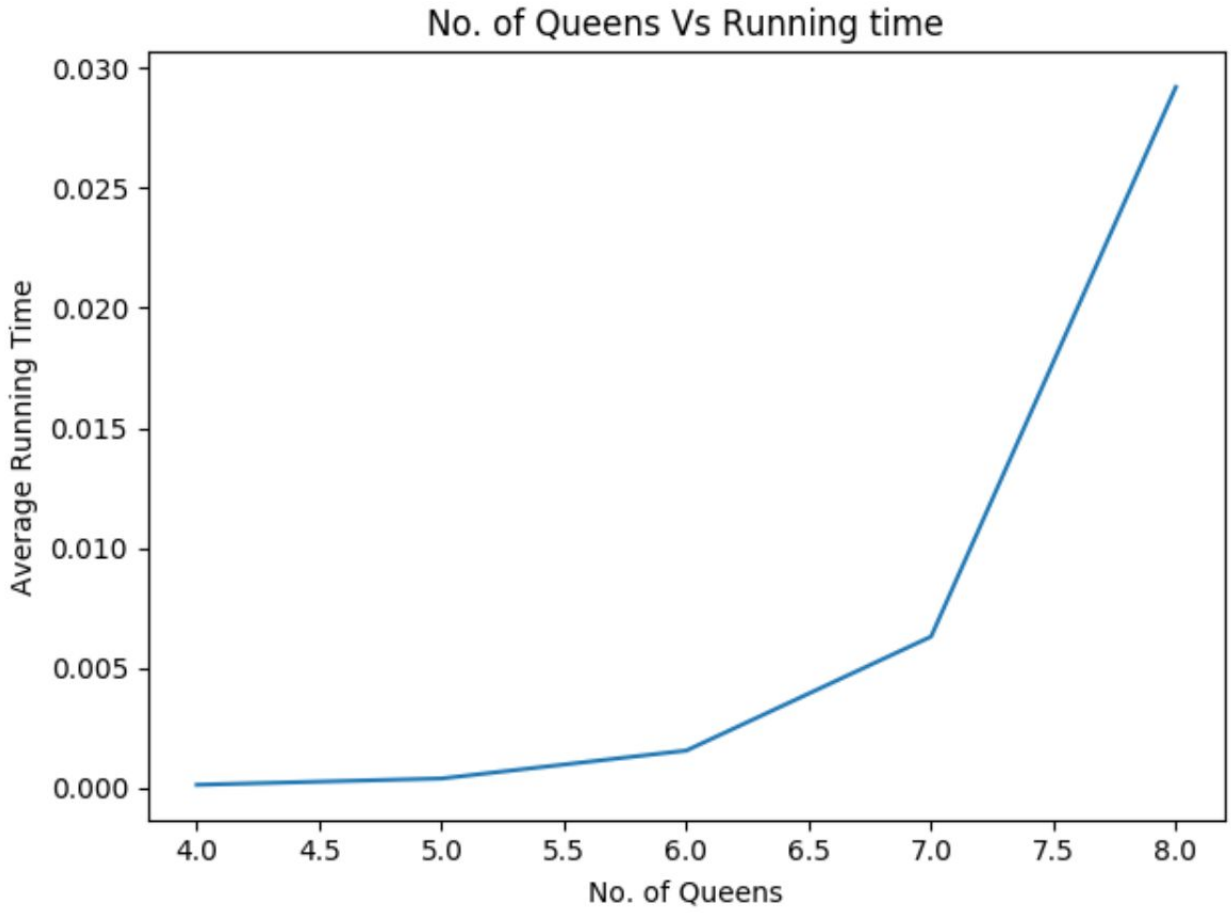
Aashish Adhikari
Note: I had to study from the internet and the book to solve these questions. But the answers are original.

1. (7 points) In this assignment you will solve the N Queens problem. The problem is to place N Queens on an N X N chess board so that they do not attack each other. In chess a queen attacks another queen if they are in the same row, column or diagonal. Your program(s) should take an integer N as an argument and return all solutions for the problem.

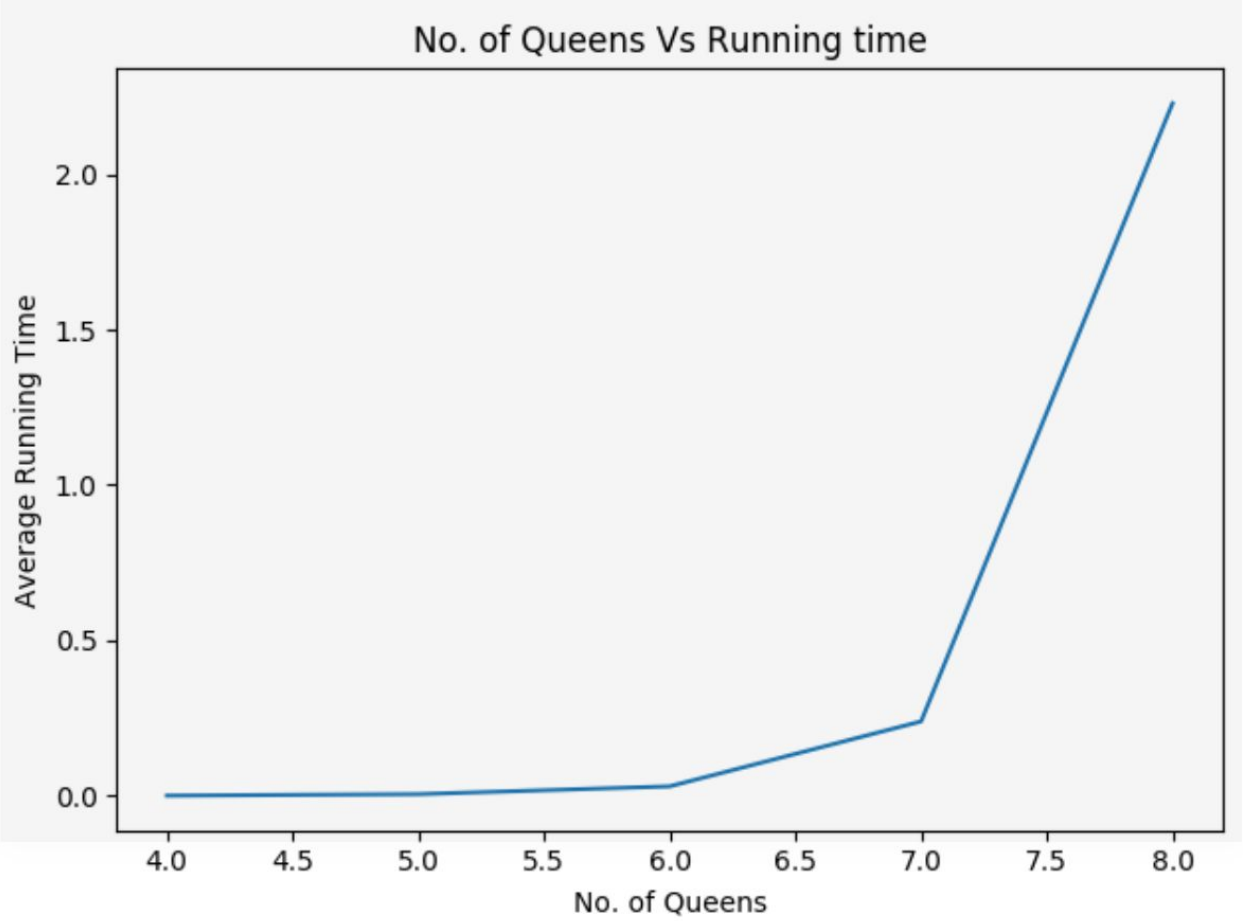
Table for N vs Time Taken for N_Queens
Note: The blue entries represent backtracking.

N =4	0.00014328	N =4	0.00087165	
N = 5	0.00040699	N = 5	0.0047461	
N = 6	0.00156903	N = 6	0.29964923	
N = 7	0.0063159	N = 7	0.2390158	
N = 8	0.0292036	N = 8	2.22880	

Plot for Backtracking



Plot for Exhaustive Search



2. (1 point). We have three containers of sizes *a* pints, *b* pints, and *c* pints where *a* > *b* > *c.* Initially the first container is empty and the others are full of water. You are allowed one type of action: pour water from one container to another until the first one is empty or the second one is full. We want to know if there is a sequence of actions that results in exactly 2 pints in one of the containers. Formulate this problem as a graph problem. What is the nature of the graph and what algorithm on graphs would solve the problem?

Solution:
I would consider each container as a node and the action of pouring from a node A to B as an edge. Hence there cannot be an edge from a to b, a to c, and b to c. Hence, a

appears to be a sink. It takes the form of a directed graph. I think, may be, we can use depth first search to find the solution to this problem if it exists. Also, i had read about minimum cut -maximum flow a while ago and I think that can also be a way to look into this problem.

3.(1 point). You are given a directed graph. Give an *O(n+e)* algorithm that determines whether or not it has a vertex *s* from which all other vertices are reachable.

Solution:
A vertex v in a graph G from which all other nodes can be reached is called a mother vertex. There can be multiple of such vertices in a graph. We can use DFS to determine the mother vertex. We know that such a vertex is post-visited only after all of its descendants are post-visited in DFS.
Hence, if we have a vertex V that was post-visited at the end, we can show that no vertex Z exists that has an edge from itself to V. There are two ways in which V and Z are encountered. When V is encountered first, if there was an edge from Z to V, then there can be two cases, either V is post-visited before Z which contradicts our assumption that V is a mother vertex or Z should be reachable from V. When Z is encountered first, however, if an edge from Z to V exists, then V is post-visited before Z since V is reachable from Z and the parent Z is post-visited after the child V in DFS.

The algorithm for this check looks as below:

1. Take any arbitrary node N and do a graph traversal from it.
    a. In the process, note down the last vertex V post-visited.
2. Now, do a DFS starting from V, the vertex that finished last in step 1. If V is a mother vertex, all nodes in the must be encountered during this second run of DFS.

Time Complexity:
Step 1 takes O(|V|+|E|)
Step 2 takes O(|V|+|E|)
Total time complexity: O(|V|+|E|) + O(|V|+|E|) = O(|V|+|E|)

4. (1 point). Design a linear-time algorithm which, given an undirected graph G and a particular edge *(u, v)*, determines if there is a cycle containing *(u, v).*

Solution: We know that for a graph to contain a cycle with an edge e connecting vertices A and B iff there exists a path from A to B even when we remove this particular edge e. Hence, we can check if node B is reachable even when we remove the edge e from the graph to determine if there is a cycle containing the edge e.
Algorithm:

Let function TRAVERSE(G) represents a DFS traversal of a graph G.

1. TRAVERSE(G- (A,B)).
2. If we encounter B even during this traversal, then there is a cycle containing (A,B) in G.

Time Complexity:
DFS runs in linear time. We are adding one extra task of removing an edge. This also takes linear time. Hence time complexity = O(n) + O(n) = O(n)