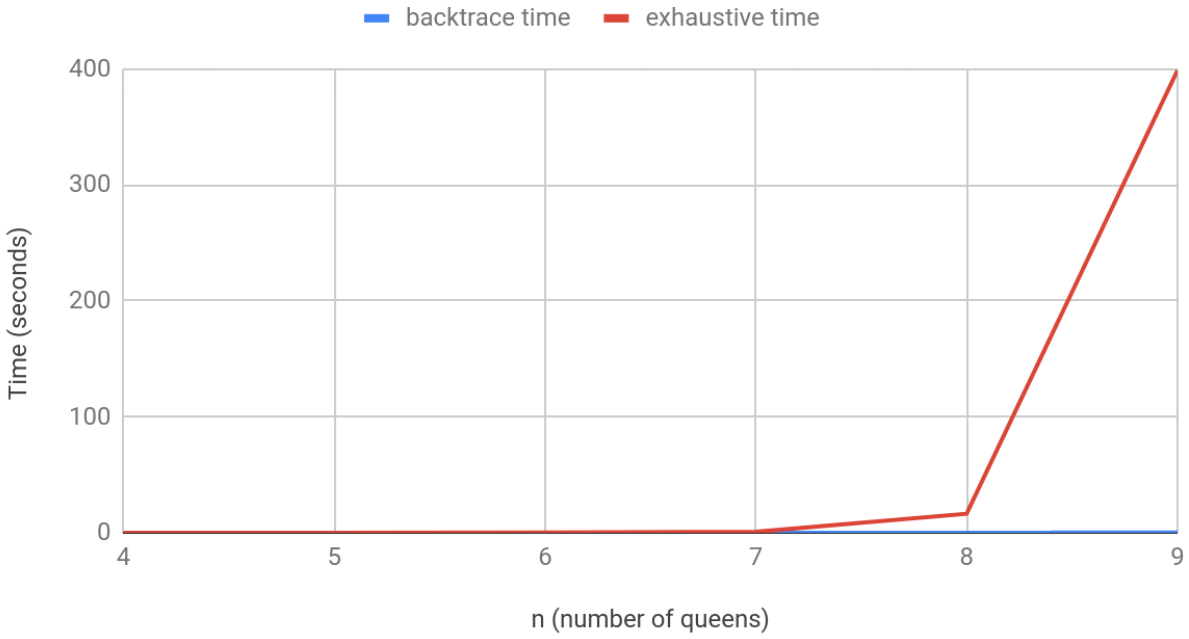
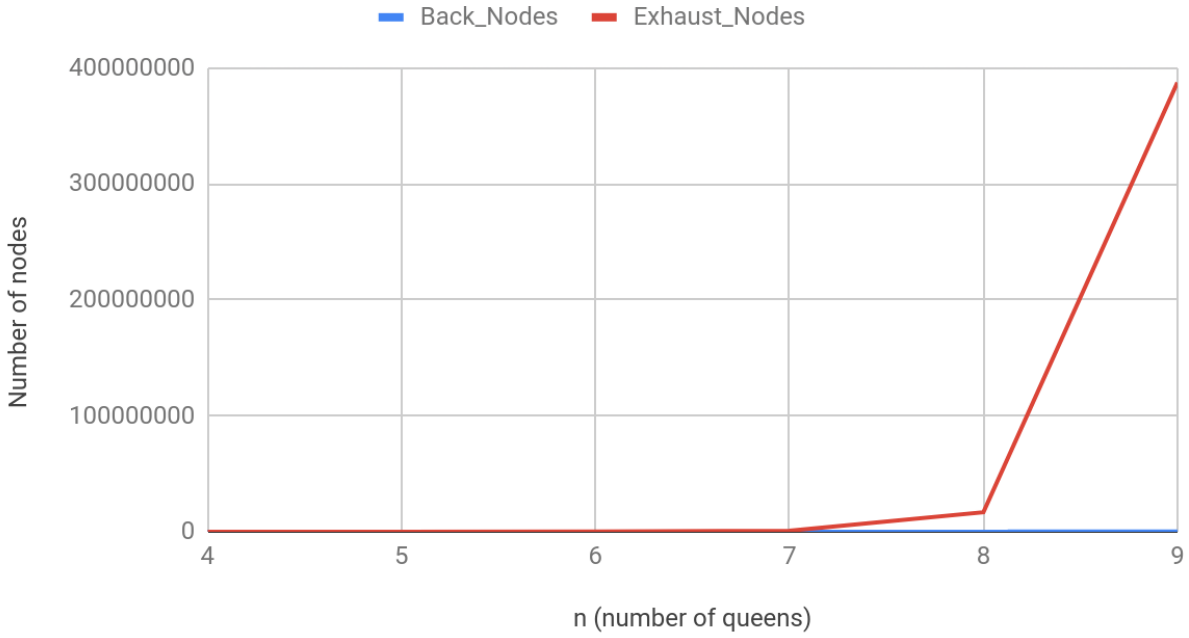
Jonathan Koning
koningj@oregonstate.edu
CS 514

Report 4

Time



Number of nodes



n	backtrace time (seconds)	exhaustive time (seconds)	Back_Nodes	Exhaust_Nodes	Solutions
4	4.80E-05	0.0002789269929	60	256	2
5	0.0001542690006	0.003294412003	220	3125	10
6	0.0005859049998	0.040435641	894	46656	4
7	0.002020703003	0.745964104	3584	823543	40
8	0.009151471007	16.33188385	15720	16777216	92
9	0.044784614	398.5929074	72378	387420489	352

Backtrace is orders of magnitude faster. This is because the tree only progresses when all queens are in valid positions. If any one queen is not in a valid position, that branch is not explored any further. These times are accurate because they are similar when similar numbers of nodes are checked. See Backtrace at n=9 and exhaustive at n=6. The number of nodes checked is accurate because exhaustive search should be equal to n^n as that is every possible permutation of n queens.

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 depth first search. What are the vertices and edges?

The vertices are the water levels in the containers. For example, the root node is a=0, b=full, c=full. The edges are the water levels changing. For example, pouring c into a would be the edge that brings you to the child node of a=(size of c), b=full, c=0. Each child node would be one possible next action. At each node, check the water levels in each container and keep track of the ancestors. If a child matches an ancestor, a loop has been created and the recursion should stop.

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.

1. Run a DFS search on the reverse of directed graph G, keeping track of the pre and post visit numbers
2. Find the sink SCCs of G by running a DFS search starting at the highest post visit node then moving to the next highest node and so on and so forth until all the nodes have been visited. Keep track of which nodes are the source nodes in each SCC.
3. Run DFS on G in increasing order of the source nodes found.

Each one of these steps is O(n+e) therefore the complexity is 3*O(n+e) which is just O(n+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).*

1. Mark vertices u and v as visited
2. Run DFS starting at u. If a back edge is created to v, then there is a cycle.