## Problem 1

Consider the following problem: You have two jugs of water with capacities 4 and 13 liters. You also have an infinite supply of water. Can you use the two jugs to get exactly 2 liters of water? Cast this as a search problem: what is your state space, initial state, action space, goal condition, and costs of actions?

State Space: All integer combinations of 0-4 liters in small jug and 0-13 liters in large jug.
Initial state: Two empty jugs
Action space: Fill smaller jug, pour smaller jug into larger jug, pour out larger jug.
Goal condition: Exactly 2 liters of water
Cost of action: Number of actions to reach the goal
Fill 4liter jug and pour into 13 liter jug. Do this 3 times until 13 liter jug has 12 liters and 4 liter jug is empty.
Next fill the 4liter jug and pour 1 liter into the 13 liter. Empty the 13liter and then pour 3 liters into 13 liter jug.
Refill 4liter and pour into 13 liter (7 liters now)
Refill 4liter and pour into 13 liters(11 liters now)
Refill 4 liter and pour into 13 liter(13 liters now), and 4 liter bottle still has 2 liters.
Empty 13 liter and we now have 2 liters.

## Problem 2

Describe a search space in which iterative deepening search performs much worse than depth-rst search. In this situation, give big-O descriptions of the running time for iterative deepening and for depth-rst search.

In the case where you have a search space that has a branching factor of 1, the DFS will perform in $O(n)$ runtime whereas the iterative deepening search will perform in $O(n^2)$.

## Problem 3

Sometimes there is no good evaluation function for a problem, but there is a good comparison method: a way to tell if one node is better than another, without assigning numerical values to either. Show that this is enough to do a best - rst search. (a) Describe in words and pseudo code how you would implement this method. (b) Show what the differences in time and memory would be (if any) compared to a standard greedy search where you know the particular value associated with each node.

a) If you can only compare two node, for each nodes, compare them and then enqueue onto a priority queue. The priority queue will implement the comparision method to evaluate whether a node is better than another.

b) So the difference between the two searches is essentially that greedy search is similar to a DFS so it is more memory efficient than a best first search which is similar to a BFS. However, the greedy search does not necessarily find the optimal path therefore the best first search has a better runtime.

## Problem 4

There are n people $0, 1, ..., n-1$. They all need to cross the bridge but they have just one ashlight. A maximum of two people may cross at a time. It is nighttime, so someone must carry the single ashlight during each crossing. They all have different speeds such that the time taken to cross the bridge for person i is given by T(i) minutes. You may assume without loss of generality that for $i < j, T(i)T(j)$. The speed of two people crossing the bridge is determined by the slower of the two. You need to nd the shortest amount of time in which all the people can cross the bridge. As an example suppose there are 4 person and time taken $(T(i))$ are $1, 2, 5$ and $10 for i = 0, 1, 2, 3$ respectively.

1. The sum of time that it takes for everyone to cross. This is admissible because it does not account for the time it would take for someone to cross back and therefore is the shortest time that people can cross and it will never be longer. This is definitely consistent because the sum of times for everyone to cross is less than the total amount of time following the regular constraints because I relaxed the constraint that people need to go back.

2. The shortest amount of time it takes one person to cross. Admissible because if everyone took the same amount of time then that would be the fastest possible time and if everyone else crossed slower than the fastest person it's still an optimistic heuristic. It's consistent because it is the fastest time for someone to cross and everyone will either be slower or at the same time.

3. The total number of people left to cross. Admissible because it will never be more people than there totally are that need to cross.

## Problem 5

Which of the following are admissible, given admissible heuristics $h1$ and $h2$?

i. $h(n) = min(h1(n), h2(n))$
Admissible

ii. $h(n) = wh1(n) + (1w)h2(n)$, where $0 \leq w \leq 1$
Admissible

iii. $h(n) = max(h1(n), h2(n))$
Admissible

## Problem 6

Which of the following are consistent, given consistent heuristics $h1$ and $h2$?

1. $h(n) = min(h1(n), h2(n))$
   Consistent
2. $h(n) = wh1(n) + (1w)h2(n)$, where $0 \leq w \leq 1$
   Consistent
3. $h(n) = max(h1(n), h2(n)$
   Consistent

## Problem 7

Assume we have a very large search space with a very large branching factor at most nodes, and we do not have any heuristic function. What search method would be good to use in this situation? Why?

The large search space rules out DFS because the search time would take too long and it would spend too much time going down one path, therefore we can examine iterative deepening search or BFS. Since we know that this search space has a large branching factor we can eliminate BFS since a large branching factor would make the space complexity way too large since BFS puts every child node onto the queue therefore although IDDFS has a slower runtime than BFS it saves on the space complexity.

## Problem 8

Suppose that your successor function is to choose the next available square (reading the board left-toright, up-to-down, like English) and write down a number in that square. What is the branching factor of this successor function?

It will be 9 because for a given square with an empty state, there are 9 possible numbers that can fill that space.

## Problem 9

Will Depth First Search (DFS) be nite on the Sudoku problem? Let r be the number of empty positions on the initial Sudoku board. What is the maximum search depth of DFS in terms of r.

If there are $r$ empty spaces then you will reach a maximum search depth of $r$ because it will only search for empty spaces and fill them.

## Problem 10

What is the best-case run time complexity of DFS, in terms of r? What is the worst-case?

Best-case: $O(9^r)$ Worst-case: $O(9^r)$ where 9 is the branching factor.

## Problem 11

What are the best-case and worst-case run times for Breadth First Search (BFS)?

Best case:$O(r)$ Worst case: $O(9^r)$ where 9 is the branching factor.

## Problem 12

We saw in class that the (worst-case) space complexity of BFS is often quite large. In the case of Sudoku, suppose you try to run BFS for a board with r = 40 on your laptop. Assume (optimistically) that storing a state in memory requires only 1 byte. How much memory might your laptop need by the time it reached the solution?

Since each position on the sudoku board has 9 possible values. The worst case space complexity will be $9^{40}$ bytes.

## Problem 13

In simulated annealing, we use a local representation of the problem. For Sudoku, the local representation is a complete assignment to the $r$ originally empty positions on the board. What is the size of this entire state space, in terms of $r$?

$9^r$

## Problem 14

What is a successor function you might use for simulated annealing?

Take two values and swap them.