Algorithm	Time Complexity	Space Complexity	Complete	Optimal
BFS	O(b^(d+1))	O(b^(d+1))	Yes	Yes (assuming that all actions have uniform cost)
UCS	O(b(C∗/ϵ))	O(b(C∗/ϵ))	Yes	Yes (under certain conditions)
DFS	O(b^d)	(O(bn))	No	No
DLS	O(b^limit)	O(b * limit)	No	No
IDS	O(b^d)	O(bn))	Yes	Yes (assuming that all actions have uniform cost)
A*	O(bd)	O(bd)	Yes	Yes (if admissible heuristic)

Question 2

BFS
Goal found
(Location: battery
Sample Extracted?: True
Holding Sample?: False
Charged? True, 'drop_tool')
Number of states generated = 83

DFS
Goal found
(Location: battery
Sample Extracted?: True
Holding Sample?: False
Charged? True, 'move_to_battery', 11)
Number of states generated = 63

DLS
Goal found
(Location: battery
Sample Extracted?: True
Holding Sample?: False

Charged? True, 'move_to_battery', 11)
Number of states generated = 63

3 Subproblems (with BFS)
Goal found
(Location: sample
Sample Extracted?: False
Holding Sample?: False
Charged? False, 'move_to_sample')
Number of states generated = 3

Goal found
(Location: sample
Sample Extracted?: True
Holding Sample?: True
Charged? False, 'pick_up_sample')
Number of states generated = 28

Goal found
(Location: battery
Sample Extracted?: True
Holding Sample?: True
Charged? True, 'move_to_battery')
Number of states generated = 18

Question 3

Number of states generated = 61 for both A* and UCS

Question 5

a) What were the engineering advances that led to Deep Blue's success? Which of them can be transferred to other problems, and which are specific to chess?

The engineering advances were large search capacity (highly non-uniform and deep), the evaluation function being performed in hardware (time to execute evaluation is a fixed constant), hybrid of software and hardware for search, parallelism, and a state-of-the art chess chip.

Most of these can be transferable to other problems. Some features that are chess-exclusive include some aspects of the best score/credit calculation, such as the absolute singular (in chess sometimes there is only one legal move, giving that move a large credit with no risk), and using a null move to detect the threat of mate

b) AlphaZero is compared to a number of modern game-playing programs, such as StockFish, which work similarly to Deep Blue. The paper shows that AlphaZero is able to defeat StockFish even when it is given only 1/100 of the computing time. Why is that? Please frame your answer in terms of search and the number of nodes evaluated.

As mentioned in the paper, DeepBlue used alpha-beta domain-specific enhancements, which I assume Stockfish also uses given they are similar. Instead, AlphaZero uses a Monte-Carlo tree search algorithm. Each search consists of a series of simulated games of self-play that traverse a tree from root to leaf. Each simulation proceeds by selecting in each state s a move a with low visit count, high move probability and high value (averaged over the leaf states of simulations that selected a from s) according to the current neural network $f\theta$. This makes it so it doesn't have to evaluate every possible move, just the most promising moves - Stockfish likely has to search through many more successors.

In addition AlphaGo Zero tuned the hyper-parameter (something we saw in class can be done via search!) of its search by Bayesian optimization, reusing the same hyper-parameters for all games without game-specific tuning.