1.

a . **False**: DFS does not always expand at least as many nodes as A search with an admissible heuristic. DFS can expand fewer nodes, especially in cases where the goal node is on a deeper level of the tree but not along the path that A would evaluate first due to the heuristic.

b. **True:** The heuristic h(n) = 0 is an admissible heuristic for the 8-puzzle because it never overestimates the cost to reach the goal. It is admissible as it is always less than or equal to the true cost from the current state to the goal.

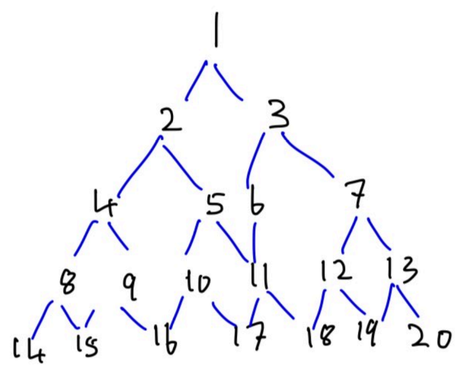
c. **False:** The A algorithm is of use in robotics. The statement that percepts, states, and actions are continuous does not negate the usefulness of A in robotics; A can be adapted for continuous spaces and is widely used in path planning.

d. **True:** Breadth-first search is complete, which means that if there is a solution, it will find it. This is true even if zero step costs are allowed because it searches level by level, and will eventually reach the goal node if it exists.

e. **True:** Manhattan distance is an admissible heuristic for the problem described.

2.

a)



b)

BFS:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

DFS:
1, 2, 3, 4, 5, 6, 7

IDS:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

c)
Bidirectional search would work well on this problem because the state space is a tree and the goal state is not too deep. The branching factor in each direction is 2, since each state has two successors.

d)
Yes, the answer to (c) suggests reformulating the problem as a shortest path problem on a tree. Dijkstra's algorithm can be used to find the shortest path from state 1 to the goal state, which requires visiting only the states on the shortest path.

e)
Yes, an algorithm that outputs the solution to this problem without any search at all. The algorithm is as follows:

- Start at state 1.
- If the current state is even, go left (to 2k).
- If the current state is odd, go right (to 2k + 1).
- Repeat steps 2 and 3 until the goal state is reached.

3)

a. Iterative lengthening search is optimal for general path costs because it expands nodes in order of increasing path cost, similar to uniform cost search. Since it discards nodes that exceed the current limit and increases the limit to the lowest discarded path cost on the next iteration, it ensures that the first solution found is the one with the lowest path cost, thus guaranteeing optimality.

b. For a uniform tree with branching factor b and solution depth d, the number of iterations will be d in the worst-case scenario. This is because the algorithm will increase the path cost limit by the smallest possible increment at each iteration, which corresponds to the unit step cost until the solution at depth d is reached.

c. If step costs are drawn from a continuous range $[\epsilon,1]$, where $0 < \epsilon < 1$, the number of iterations required in the worst case would be inversely proportional to the smallest step cost ϵ. Since the algorithm increases the path cost limit to the lowest discarded path cost of any node, in the worst case, each iteration could increase the limit by the smallest possible step cost ϵ, which would result in $1/\epsilon$ iterations to go from cost ϵ to a cost of 1.

4.

IDS can perform much worse than DFS in state space where the branching factor increases with depth and the goal node is located at a deep level.

Consider a tree where each level i has i times more nodes than level i-1. This means the branching factor is not constant but increases linearly with the depth of the tree. In such a tree, the number of nodes at level d would be proportional to d! resulting in a factorial time complexity for reaching that level.

Since IDS revisits the nodes at each depth until the goal node's depth is reached, it would have to traverse a factorial number of nodes at each iteration, leading to an overall time complexity worse than DFS, which would only need to visit each node once on its path to the goal node. Thus, while DFS would have a time complexity of O(n), with n being the number of nodes along the path to the goal, IDS would have a time complexity proportional to the sum of the factorials of the depths at each iteration, which is far worse than O(n^2) and closer to O(n!) in this scenario.

5.

a. **True**: BFS can be considered a special case of UCS. In UCS, nodes are expanded based on the lowest path cost, and BFS can be thought of as UCS, where all path costs are equal. This makes the order of node expansion in BFS equivalent to that of UCS with uniform step costs.
b. **True**: DFS is a special case of BSF. BFS uses a heuristic to determine the order of node expansion
c. **True**: Uniform-cost search is a special case of A* search. In A* search, nodes are expanded based on the cost from the start node plus a heuristic estimate of the cost to reach the goal. If the heuristic is always zero, which is an admissible heuristic, A* search behaves exactly like UCS, expanding nodes in order of their path cost from the start.

6.

a. Start node: **Lugoj**

- f score: 0 + 160 = 160
- g score: 0
- h score: 160 Lugoj to Bucharest

b. Expand Lugoj and consider its neighbors: Craiova, Timisoara, Mehadia

Craiova

- f score: 146 + 100 = 246
- g score: 146
- h score: 100 dist Craiova to Bucharest

Timisoara

- f score: 70 + 117 = 187
- g score: 70
- h score: 117 Timisoara to Bucharest

Mehadia

- f score: 74 + 140 = 214

- g score: 74
- h score: 140 Mehadia to Bucharest

c. Since Craiova has the lowest f score, it is expanded next.
d. Expand Craiova and consider its neighbors: Pitesti, Drobeta

Pitesti

f score: 246 + 94 = 340
g score: 246
h score: 94 (straight-line distance from Pitesti to Bucharest)

Drobeta

- f score: 146 + 60 = 206
- g score: 146
- h score: 60

e. Since Drobeta has the lowest f score, it is expanded next.
f. Expand Drobeta and consider its neighbors: Bucharest

Bucharest

- f score: 206 + 0 = 206
- g score: 206
- h score: 0

The A* search algorithm has now found a path from Lugoj to Bucharest with a total cost of 206. The path is Lugoj -> Craiova -> Drobeta -> Bucharest.

7.

BFS typically uses a heuristic value to order nodes, but it can operate with a comparison method instead. This method would compare nodes based on their likelihood to lead to the goal, even without numerical values.

For A search, which considers the path cost and heuristic, an analog would use a comparison method that evaluates both the cost so far and potential progress towards the goal. If this method can consistently compare the combined potential of nodes, it can replace numerical heuristics and guide the A* algorithm effectively.