Question 1

You can represent the state space of this problem as tuples where the first number represents the number of gallons in the four gallon jug (referred to as jug4) and the second number represents the number of gallons in the three gallon jug (referred to as jug3). Its general case looks like (jug4, jug3). From this the intial state is $(0, 0)$ and the goal state is $(n, 2), n \in [0, 4]$. Actions you can take are the empty jug4 and jug3 which sets their value to 0, fill jug4 which sets its value to 4, fill jug3 which sets its value to 3, pour jug4 into jug3 which will add the value of jug4 to the value of jug3 up to it full value of three, and the final action you can do is to pour jug3 into jug4 which will add the value of jug3 to the value of jug4 up to its full value of 4.

One way to acheive the solution is to:

Action	**State**
fill jug4	(4,0)
pour jug4 into jug3	(1,3)
empty jug3	(1,0)
pour jug4 into jug3	(0,1)
fill jug4	(4,1)
pour jug4 into jug3	(2,3)
empty jug3	(2,0)
pour jug4 into jug3	(0,2)

Question 2

You can represent the state space as three arrays (one for each pole) containing a the set of numbers corresponding to what discs are on that pole. We number each disc with a value from 1 to 64 such that disc1 is the smallest disc and disc64 is the largest. The order of values in each array is not important since we know that the discs must be sorted in ascending order on the pole. The initial state would be [1..64][][] and the goal state would be [][1..64][]. Actions can be described as moving discN to poleN.

For the sake of brevity the solution provided will only go up to the first 5 discs but there is an obvious pattern that would allow the rest of the solution to be inferred:

Action	**State**
disc1 to pole2	[2..64][1][]
disc2 to pole3	[3..64][1][2]
disc1 to pole3	[3..64][][1,2]
disc3 to pole2	[4..64][3][1,2]
disc1 to pole1	[1,4..64][3][2]
disc2 to pole2	[1,4..64][2,3][]
disc1 to pole2	[4..64][1..3][]
disc4 to pole3	[5..64][1..3][4]
disc1 to pole3	[5..64][2,3][1,4]
disc2 to pole1	[2,5..64][3][1,4]
disc1 to pole1	[1,2,5..64][3][4]
disc3 to pole3	[1,2,5..64][][3,4]
disc1 to pole2	[2,5..64][1][3,4]
disc2 to pole3	[5..64][1][2..4]
disc1 to pole3	[5..64][][1..4]
disc5 to pole3	[6..64][5][1..4]
disc1 to pole1	[1,6..64][5][2..4]
disc2 to pole2	[1,6..64][2,5][3,4]
disc1 to pole2	[6..64][1,2,5][3,4]
disc3 to pole1	[3,6..64][1,2,5][4]
disc1 to pole3	[3,6..64][2,5][1,4]
disc2 to pole1	[2,3,6..64][5][1,4]
disc1 to pole1	[1..3,6..64][5][4]
disc4 to pole2	[1..3,6..64][4,5][]
disc1 to pole2	[2,3,6..64][1,4,5][]
disc2 to pole3	[3,6..64][1,4,5][2]
disc1 to pole3	[3,6..64][4,5][1,2]
disc3 to pole2	[6..64][3..5][1,2]
disc1 to pole1	[1,6..64][3..5][2]
disc2 to pole2	[1,6..64][2..5][]
disc1 to pole2	[6..64][1..5][]

Question 3

Breadth first search:

Step	**Closed Queue**	**Open Queue**
expand A	[B, D, E]	[A]
expand B	[D, E, G]	[A, B]
expand D	[E, G, C, F]	[A, B, D]
expand E	[G, C, F, I]	[A, B, D, E]
expand G	[C, F, I]	[A, B, D, E, G]
expand C	[F, I]	[A, B, D, E, G, C]
expand F	[I, H]	[A, B, D, E, G, C, F]
expand I	[]	[]

Depth first search:

Step	Closed Queue	Open Queue
expand A	[B, D, E]	[A]
expand B	[D, E, G]	[A, B]
expand G	[D, E, I]	[A, B, G]
expand I	[]	[]

Question 4

Breadth first search final path: S, A, B, H, G

Step	Closed Queue	Open Queue
expand S	[A, D]	[S]
expand A	[D, B]	[S, A]
expand D	[B, E]	[S, A, D]
expand B	[E, C, H]	[S, A, D, B]
expand E	[C, H, F]	[S, A, D, B, E]
expand C	[H, F]	[S, A, D, B, E, C]
expand H	[F, G]	[S, A, D, B, E, C, H]
expand F	[G]	[S, A, D, B, E, C, H, F]
expand G	[]	[]

Depth first search final path: S, D, E, F, H, G

Step	Closed Queue	Open Queue
expand S	[A, D]	[S]
expand D	[A, E]	[S, D]
expand E	[A, B, C, F]	[S, D, E]
expand F	[A, B, C, H]	[S, D, E, F]
expand H	[A, B, C, G]	[S, D, E, F, H]
expand G	[]	[]

Uniform cost search final path: S, A, B, H, G, total 12

Step	Open Queue	Closed Queue
expand S	[A:3, D:4]	[S:0]
expand A	[D:4, B:7]	[S:0, A:3]
expand D	[E:6, B:7]	[S:0, A:3, D:4]
expand E	[B:7, C:10, F: 10]	[S:0, A:3, D:4, E:6]
expand B	[C:10, F: 10, H:11]	[S:0, A:3, D:4, E:6, B:7]
expand C	[F: 10, H:11]	[S:0, A:3, D:4, E:6, B:7, C:10]
expand F	[H:11]	[S:0, A:3, D:4, E:6, B:7, C:10, F:10]
expand H	[G:12]	[S:0, A:3, D:4, E:6, B:7, C:10, F:10, H:11]
expand G	[]	[]

Question 5

Uniform Cost Search:

Step	Open Queue	Closed Queue
expand 1	[5:5, 8:24]	[1]
expand 5	[8:24, 6:40]	[1, 5]
expand 8	[10:39, 6:40, 3:47]	[1, 5, 8]
expand 10	[6:40, 3:47, 9:65]	[1, 5, 8, 10]
expand 6	[3:47, 9:65, 2:78]	[1, 5, 8, 10, 6]
expand 3	[4:54, 9:65, 2:78]	[1, 5, 8, 10, 6, 3]
expand 4	[9:65, 2:78]	[1, 5, 8, 10, 6, 3, 4]
expand 9	[2:78, 7:100]	[1, 5, 8, 10, 6, 3, 4, 9]
expand 2	[7:100]	[1, 5, 8, 10, 6, 3, 4, 9, 2]

Greedy BFS:

Step	Open Queue	Closed Queue
	[1]	[]
expand 1	[8, 1]	[]
expand 8	[3, 8, 1]	[]
expand 3	[4, 3, 8, 1]	[]
expand 4	[9, 3, 8, 1]	[4]
expand 9	[7, 3, 8, 1]	[4]

A *:

Step	Open Queue	Closed Queue
	[1]	[]
expand 1	[5:84, 8:84]	[1]
expand 5	[8:84, 6: 100]	[1, 5]
expand 8	[3: 84, 10: 96, 6: 100]	[1, 5, 8]
expand 3	[4: 84, 10: 96, 6: 100]	[1, 5, 8, 3]
expand 4	[10: 96, 6: 100, 9:07]	[1, 5, 8, 3, 4]
expand 10	[9:100, 6: 100]	[1, 5, 8, 3, 4, 10]
expand 9	[7: 100, 6: 100, 2:123]	[1, 5, 8, 3, 4, 10, 9]
expand 7	[6: 100, 2:123]	[1, 5, 8, 3, 4, 10, 9, 7]

Question 6

a) No, h1 does not take into account the shape of the maze. It ignores most given information for the question.

b) Yes. We know that h1 and h2 are both admissible, so the max of the two cannot be greater than them. The greatest of two values less than something will still be less than that thing.

c) Yes. If the max of the two is admissible the min must be less than that and there for admissible.

d) No. It is possible that h2 provides a maxmial solution and adding to it will overestimate making it inadmissible.

e) Yes. Subtracting from a viable solution will lower it leaving it still admissible.

f) If we have a maze with only one path through it of length n. In this case h2(n) = n. If we multiply this by any value greater than 1 it will be longer than any possible path through this making it inadmissible.

Question 7

We could use the heuristic of counting the number of black tiles to the right of a white tile as its value.

This is admissible because a white tile with a black to its right must be at least one to get over that tile. This way the heuristic will never overestimate making it admissible.

This heuristic is only dependant on the number of black tiles to the left of a white tile. This completely ignores the placement of the blank tile. In this aspect it respects monotonicity.

This heuristic leverages the aspects of the tiles (including the potential colors, and location restrictions) making it informed. It will not work in different scenarios.