Part A

1)
    a) Accessibility can depend on how good the agent's sensors are. An environment can be accessible for a costly robot, but inaccessible (or less accessible) for a cheap robot with cheap sensors.

    b) Determinism of an environment can also depend on the robot to some extent. If a robot has faulty effectors (motors) then the environment will become stochastic for that robot, but more predictable for a robot with a better set of effectors
    c) Whether or not the environment is static basically depends on whether there are other agents and whether the environment evolves by itself. This is typically beyond the capabilities of the robot.

    [It is however possible for two agents A1 and A2 in the same environment E, such that A1 thinks the environment is _more_ dynamic (changes too fast) and A2 thinks that environment is basically static (doesn't change all that fast). Notice that the environment changes at the same absolute temporal rate. The reason A1 may find that E is more dynamic than A2 does is essentially related to the length of the deliberation/action cycle of the agent relative to the environment change. Consider a road where there is, on the average, a car every 7 minutes. It is possible that A1 takes on average 8 minutes to think up a plan to cross the road and another 10 minutes to execute it, while agent A2 takes 25 seconds to think up a plan and a minute to execute it. Clearly, A2 finds the road essentially static while A1 finds it more dynamic.]

2) a)
    a) Yes. Some of the goals that A1 can achieve (such as clean all the rooms and get back to the right side room) can be achieved by A2 too.
    b) No. For example, it will be impossible for A2 to achieve the goal "clean all the rooms and and get back to the starting room" (when the starting room can be both right or left)
    c)
        ■ A2 is likely to take longer time since it will be searching in a state space that is exponentially larger than that searched by A1 (if the size of space searched by A1 is |S|, then the one searched by A2 will be 2^|S|

        ■ In general, the quality of the plans and actions made by A2 may not be as good as that of A1. For example, A2 may have to do a go-right action to make sure that finally it is in the right room, but A1 may realize that by the time it did the second suck, it is already in the right room, so it can avoid the redundant go-right action.

b)

a) To visualize the answers to this consider a vacuum world which has faulty motors, because of which, when it says suck, it may not suck all the dirt. When it says go-right/go-left it may not reach those rooms with certainty.

(i) Consider two cases: the domain is fully accessible for both agents; and the domain is not accessible for both agents.

If the domain is not accessible, in general, there may not be any goals that A2 can achieve with certainty (although it can achieve them with increasing probability).

If the domain is accessible, the situation is slightly better (e.g. the robot can keep sucking until it finds that there is no
dirt), or keep going right until it eventually reaches the right room. (I say "slightly better" because even here, in the worst case you may keep sucking and find there is always a little bit of dirt that is still left.).
b) Clearly, no.
c)
i) A2 effectively is searching a larger space. So it takes longer
ii) A2 may have to do additional actions (e.g. extra sucks and extra go-rights) because its actions are stochastic (motors are misbehaving).

3) No. It doesn't make sense. For b=1, we have a tree with a single path of length d.
Depth first expands all d nodes.
IDDFS expands 1+ 2 + 3 + .. +D = D(D+1)/2
So ratio is (D+1)/2

4)

a) Of course, depth first search. If all goal nodes are at the same depth there is no point using BFS.

b) Solutions for part (1) and (3)
- If the goals are all clustered, it may have been possible that they are all among 3.1,3.2,and 3.3.. In this case, full DFS will get to them much later than our iterative approach does. Notice that iterative broadening improves its ability to look around the search space (follow local gradients) at the expense of redundant search, but without sacrificing completeness. Hill climbing has the ability to follow the gradients, but it is not complete
--> Notice also the connection with random-restart search (RRS) that I discussed in the class. RRS tries to get mobility among the leaf nodes by cutting off unpromising search after a backtrack cutoff, and starting over with a different permutation of children nodes. RRS is

asymptotically complete (in that it exhausts the search space), if we keep increasing the cutoff limit

Solution for part(2)

This is called the iterative broadening search.
If you have 0 as the root node. 1, 2 and 3 as the first level nodes, 1.1, 1.2, 1.3, 2.1,2.2, 2.3, 3.1, 3.2 ,3.3 as the second level nodes and so on

In the first iteration we expand nodes 0, 1, 1.1

In the second iteration we expand nodes 0, 1, 1.1, 1.2, 2, 2.1, 2.2

In the third iteration we expand all nodes.

5) Suppose the search ended and returned n1 as the goal node. We will try to prove by contradiction. we want to say that n1 is not within epsilon of the optimal goal node. Thus, suppose there is another node n2 such that

g(n1) > g(n2) + eps ---->Eq(1)

which implies that g(n1) is not within eps of optimum. Now when n1 was selected for expansion, there must have been some ancestor n2' of n2 that was on the search queue. Since A* picked n1 rather than n2', it must be that

f(n1) <= f(n2')

f(n1) = g(n1) (since h(n1) has to be zero at the goal node).
f(n2') = g(n2') + h(n2')

h(n2') <= [g(n2) - g(n2')] + eps (h overestimates the true distance by eps at most)

So f(n2') <= g(n2) + eps

So, g(n1) <= g(n2) + eps --->(2)

But, (2) is contradicting our assumption (1)
So, there is no such n1.

6)
   a) a. Not unless the tree is finite. You may wind up following a path to infinity as it becomes a longer and longer partial path.
   b) Yes. Note that when it returns a solution, it makes sure that the f-values of all other nodes are greater than these nodes. So, we have a complete path that is shorter than all other partial paths. Branch and bound ensures optimality (under the assumption that costs are all positive).
   c) way less efficient. The idea in this question is to make you realize that A* uses branch and bound to ensure the optimality of the solution, and priority queue sorted in increasing f-values to improve efficiency. These can be separated--giving rise to New-A* (which really should be called mad-A*).

7) If w=2, then it will become a greedy search. It acts almost like hill-climbing, BUT it is systematic (so on a finite search space without cycles, it will terminate). In general, it is neither guaranteed to be complete, nor optimal. Since it is completely ignoring the cost of the path to get to a node, and only cares about cost from that node to the goal node, it will not be optimal even when h=h* (consider a scenario where right below the root node there are two nodes N1 and N2. N1 has heuristic value 0 and N2 has 4. The edge cost to N1 is 25. Edge cost to N2 is 3. In this case, our algorithm picks N1 and terminates--and this could very well be a costly goal).

Part B

All true and false are given 2 points only if both answer and reasoning is correct

1) False - We are talking about bets case scenario in A*. It does d expansions (b^(d/2) is too high!).
2) False - If all nodes are at the same depth, and are of the same cost (this is the definition of uniform search tree), what is the point in doing BFS? All its expansions in interior levels are wasted effort.
3) False - With h=0, A* is equivalent to uniform cost search. Remember that uniform cost search ends even though there are partial paths on the search queue, as long as all partial paths are worse than the complete path you have.
4) False - If w=1, then it will become a greedy search, which is incomplete.
5) True - Basically, Yes the higher the heuristic the better.
6) False - Hill climbing doesn't know where it has been and so it can cycle indefinitely even in finite trees.

Part C

1.
    a. Misplaced tile heuristic: 7
    b. Manhattan distance heuristic: 10
    c. PDB heuristic: 8
2. Perfect heuristic is infinity. Misplaced < PDB < Manhattan
3. There are a total of 9! =362880 distinct states in the 8 puzzle (think of it as arranging the numbers 0…8 in all permutations. For each permutation, you get a distinct puzzle configuration by putting the first three numbers in the first row, second three in the second row and the third three in the third row) Now for PDB computation, we distinguish states only in terms of the positions of the fringe tiles (1,2,3) and the blank position (0). This leaves 9-4 tiles whose actual position in the table doesn't matter. So, to get the number of distinct fringe tile patterns, we have to divide 9! By (9-4)! 9!/(9-4!) = 6*7*8*9= 3024. So there are 3024 entries in the PDB (instead of 362880 entries if we had the entire puzzle as the fringe—as 120x reduction). The PDB is thus 0.833% (or 8.333/1000) of the full state space size. (Here is another way to make sense of the calculation above. Suppose we start with an empty puzzle, and put 1 2 3 and 0 in some random places. Now this leaves 5 tile positions empty, into which we can arrange the remaining 5 non-fringe tiles. There are 5!=120 ways of doing this. Thus, every distinct configuration of the fringe tiles and blank gives rise to 120 puzzle configurations which have that specific fringe configuration, but differ only in the number of non-fringe tiles. So, we can get the number of distinct fringe tile patterns by dividing the total number of 8 puzzle states—9! By 5!.