Problem 1 Proof:

Consider a search space with a path cost function *g*, and assume that *h* is an admissible heuristic for the space. Thus we know that an Algorithm A search which selects the nodes off of the open list according to:

$$f(n) = g(n) + h(n)$$

is guaranteed to find an optimal solution. This search is called an Algorithm *A** Search. In many cases, instead of using *A** we use *wd A**, which is a *weighted* version of *A** that places more emphasis on the heuristic function. In this case, we select nodes off of the open list according to: $f(n) = g(n) + w * h(n)$ for some constant $w > 1$.

Prove that the solution found by *wd A** is at most a factor of *w* from optimal. More precisely, let *ng* be the goal node found by *wd A**, *nd opt* be the optimal goal node, and *c' = g'(nd opt)* be the optimal cost to the goal node *nd opt*; show that

$$g(ng) \leq w * c'$$

Your proof should be short and mathematical, and should not take more than 7p 8 lines.

Suppose that the graphsearch ended in finding node ng but g(ng) > w * c'. But, we will show there is a node still on the open list, nt, where f(nt) ≤ w * c' and this node would therefore be taken off the openp list instead of ng.

Consider an optiomal path s = n0 n1 n2… nopt. We know that for every node along the optimal path, f'(n) = g'(n) + h'(n).

We know that at each step of the graph search, at least one node on the optimal path is still on the open list. Let nl be the leftp most of these nodes. At this point in time, we know that f(nl) = g'(nl) + w*h(nl)
but h(nl) ≤ h'(nl) because h is admissible. Given that, it follows that

g'(nl) + w * h(nl) ≤ g'(nl) + w * h(nl) and therefore

g'(nl) + w * h(nl) ≤ w * (g'(nl) + h'(nl)) which means

g'(nl) + w * h(nl) ≤ w * c'

Therefore, the node nl would have been taken off the open list before ng.

Problem 2 Graph Searches:

(a) Uniformt cost Search

EXPL: S J I [F A] B [D C E] H

Path: S >A >B >D >H

(b) Greedy Bestt first Search

EXPL: S J F I H

Path: S >J >I >H

(c) Algorithm A Search

EXPL: S J [I F] C D [E H]

Path: S >J >I >H *or* S >J >I >C >D >H

(d) Iterative Deepening A Search

fp limit = 11

EXPL: S J I F

fp limit = 12

EXPL: S J I C D F

fp limit = 13

EXPL: S J I H

NOTE: other orderings are possible due to order in which child nodes are selected (any consistent ordering is correct).

(e) Steepest Ascent (Descent?) Search

EXPL: S J F (stuck!)

Path: (none found)

(f) Breadth First: Expl:

S A [J B] C [I F D] G

Path: S A C G

(g) Depth First:

Expl: S A B [C] D H

path: S A B D H

Question 3:

a. State space: States are all possible city pairs (i, j). The map is *not* the state space.
   Successor function: The successors of (i, j) are all pairs (x, y) such that Adjacent(x, i) and Adjacent(y, j).
   Goal: Be at (i, i) for some i.
   Step cost function: The cost to go from (i, j) to (x, y) is max(d(i, x), d(j, y)).

b. In the best case, the friends head straight for each other in steps of equal size, reducing their separation by twice the time cost on each step. Hence (iii) is admissible.

c. Yes: e.g., a map with two nodes connected by one link. The two friends will swap places forever. The same will happen on any chain if they start an odd number of steps apart. (One can see this best on the graph that represents the state space, which has two disjoint sets of nodes.) The same even holds for a grid of any size or shape, because every move changes the Manhattan distance between the two friends by 0 or 2.

d. Yes: take any of the unsolvable maps from part (c) and add a self-loop to any one of the nodes. If the friends start an odd number of steps apart, a move in which one of the friends takes the self-loop changes the distance by 1, rendering the problem solvable. If the self-loop is not taken, the argument from (c) applies and no solution is possible.

Question 4:

a. Initial state: one arbitrarily selected piece (say a straight piece).
Successor function: for any open peg, add any piece type from remaining types. (You can add to open holes as well, but that isn't necessary as all complete tracks can be made by adding to pegs.) For a curved piece, add *in either orientation*; for a fork, add *in either orientation* and (if there are two holes) connecting *at either hole*. It's a good idea to disallow any overlapping configuration, as this terminates hopeless configurations early. (Note: there is no need to consider open holes, because in any solution these will be filled by pieces added to open pegs.)
Goal test: all pieces used in a single connected track, no open pegs or holes, no overlapping tracks.
Step cost: one per piece (actually, doesn't really matter).
All solutions are at the same depth, so depth-first search would be appropriate. (One could also use depth-limited search with limit $n-1$, but strictly speaking it's not necessary to do the work of checking the limit because states at depth $n - 1$ have no successors.) The space is very large, so uniform-cost and breadth-first would fail, and iterative deepening simply does unnecessary extra work. There are many repeated states, so it might be good to use a closed list.

b. A solution has no open pegs or holes, so every peg is in a hole, so there must be equal numbers of pegs and holes. Removing a fork violates this property. There are two other "proofs" that are acceptable: 1) a similar argument to the effect that there must be an even number of "ends"; 2) each fork creates two tracks, and only a fork can rejoin those tracks into one, so if a fork is missing it won't work. The argument using pegs and holes is actually more general, because it also applies to the case of a three-way fork that has one hole and three pegs or one peg and three holes. The "ends" argument fails here, as does the fork/rejoin argument (which is a bit handwavy anyway).

c. The maximum possible number of open pegs is 3 (starts at 1, adding a two-peg fork increases it by one). Pretending each piece is unique, any piece can be added to a peg, giving at most $12 + (2 \cdot 16) + (2 \cdot 2) + (2 \cdot 2 \cdot 2) = 56$ choices per peg. The total depth is 32 (there are 32 pieces), so an upper bound is $168^{32}/(12! \cdot 16! \cdot 2! \cdot 2!)$ where the factorials deal with permutations of identical pieces. One could do a more refined analysis to handle the fact that the branching factor shrinks as we go down the tree, but it is not pretty.