CS 4804 Homework 1
Solution Sketches

---

1. **(20 points)**

   a) This is just a complete binary tree with a depth of 4. The nodes are labelled from 1 to 15 from top to bottom, left to right.

   b) The respective algorithms would produce the following sequences of visited states:
   BFS: 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
   DFS: 1,2,4,8,9,5,10,11
   IDS: 1,1,2,3,1,2,4,5,3,6,7,1,2,4,8,9,5,10,11

   c) A bi-directional search would work perfectly. The forward search can use BFS and the backward can use DFS, with the predecessor function defined as $\text{Pred}(x) = \lfloor x/2 \rfloor$.

   d) The forward branching factor is 2 and the backward branching factor is 1.

   e) From the previous results it should be obvious that a search starting at the goal node and proceeding backwards to the start node would result in no searching. In other words, just proceed in one direction, back from the goal node. Since there is only one operator, this will result in 'no searching'!

2. **(10 points)** The book shows that an IDS is better than a BFS when the branching factor is 2. In general, this will be true whenever the number of nodes at a given depth d is exponential with respect to d, so you must show that in the grid space, the number of nodes is *not* exponential. In the grid space, the number of nodes is linear with respect to d, with the number of nodes at depth $d = 4d$. Now, given that the number of nodes at a given depth is $O(d)$, both use the same order of memory. The BFS search will generate $\sum_{i=1}^{d+1} 4i - 1 = 2d^2+6d+3$ nodes. IDS on the other hand will generate $\sum_{i=1}^{d=1}(4i(d-i+1))-1 = 4\frac{d^3+3d^2+2d}{6}-1$, which is an order of magnitude greater node generation. Another way of saying this is that, in IDS, the same node will be visited over and over again at many levels.

3. **(20 points)** The state space is all the permutations of the black, white, and empty tiles, for a total of $\frac{7!}{3!3!} = 140$ different states. The goal test is one of the seven states where all the white tiles are left of all the black tiles. One simple heuristic is to count the number of black tiles in the three left positions and the number of white tiles in the three right positions, and add them. Since every one of these tiles must be moved, and only one tile can be moved at a time, this is an admissible heuristic.

4. **(10 points)** Trivial. You should have a list of states that lead from the start state to a goal state. Each state should have the g() and h() values calculated.

5. **(10 points)** This function produces an optimal search when $w <= 1$. When w increases, it can cause the heuristic term to overestimate the cost to the goal node, which will lead to a non-optimal solution. When $w = 0$ this performs a standard UCS search, which is both optimal and complete. When $w = 1$ this is a standard A* search, which is again optimal and complete. When $w = 2$ a greedy search is performed. This search is not optimal as the h() term may be overestimated, and it is not complete since it does not have any memory, and so can continue indefinitely down a wrong path.

6. **(30 points)** This problem is posed as a maximization problem whereas A* search is traditionally doing path cost minimization. So you first have to come up with a notion of cost, so that minimizing this becomes equivalent to maximizing points. The second twist is that the cost (or points) is accrued only on alternate moves. And finally, you have to relax the problem to discover a heuristic.

   The start state should be a randomly generated list of odd length, and the goal state is an empty list. The operator is to remove an element from one of the ends of the list, with a cost of 0 on an odd move and the removed element's value on an even move. Notice that the cost now becomes the value of cards that *do not count* toward the score. Minimizing this cost is the same as maximizing the score! This formulation has the advantage that a path's costs are non-decreasing as you go down it.

   As a heuristic, sum all the $\lfloor \frac{n}{2} \rfloor$ smallest elements. Why is this admissible? One way to think of the relaxation of the problem is to say that you can remove a card from anywhere, not just the ends. In this case, it is clear that you get the best score by picking the top half of the cards (sorted numerically in decreasing order) on the odd moves, so that the cost becomes the lower half of the cards.

   We ran through 10 trials with lists of length 11, 21, and 31, with the results summarized below.

List Length	Nodes A*	Branching factor A*	Nodes UCS	Branching factor UCS
11	16.1	1.06	133.0	1.39
21	43.5	1.06	3797.1	1.39
31	121.0	1.07	21369.9	1.31

   It is clear that informed search is better than uninformed search.