CSE 101 Homework 3
Due Thursday, May 29
Back-Tracking and Greedy Algorithms

Directions: For each of the first four problems, a "high level" greedy strategy is given. For some of the problems, the strategies give a correct (optimal) solution, and for others, it sometimes gives incorrect (suboptimal) solutions. For each, decide whether the greedy strategy produces optimal solutions. If it is, give a proof that it is correct, then describe what data structures and preprocessing you would use to give an efficient version, and give a time analysis.

If it is not correct, give a counter-example showing the other strategy is incorrect. Then give a back-tracking algorithm for the problem, and give an upper bound on the time your algorithm would take.

For the last problem, you need to turn in a table of values. You will probably need to write and run a program to generate the values, but all you should hand in are the numerical values (and the input values they correspond to).

Largest Independent Set for a tree The problem is to find the largest independent set for the special case when the input graph is a tree. (Edges are between nodes and their parents and children.) Remember, an independent set S of a graph is a set of nodes that does not contain both of the endpoints of any edge, i.e. for any edge $\{x, y\}$ either $x \notin S$ or $y \notin S$. So here, we must have a set of nodes of the tree S so that we cannot have both a node and its parent in the set.

Greedy strategy: Find any leaf x in the tree, i.e., any node with no children. Add x to S. Delete x and x's parent from the tree. Repeat. (Note: when you repeat, the graph might become a *forest*, a collection of unconnected trees, rather than a single tree. In that case, x could be any leaf of any tree. If x does not have a parent, than only it is deleted.)

Maximum matching A *matching* in an undirected graph is a set of edges $M \subseteq E$, so that no two edges in M share a common endpoint, i.e., we cannot have $\{x, y\}$ and $\{x, z\}$ both in M for any three nodes $x, y \neq z$. The maximum matching problem is to find a largest matching in a given graph.

Greedy strategy: For each edge, sum the degrees of its endpoints, $s(\{x, y\}) = deg(x) + deg(y)$. Put the edge with the minimum such sum in M and remove both the endpoints from G. Repeat until no edges are left.

Room assignment You are given a list of classes C and a list of classrooms R. Each class c has a positive enrollment $E(c)$ and each room r has a positive integer capacity $S(r)$. You want to assign each class a room in a way that minimizes the total sizes of rooms used. However, the capacity of the room assigned to a class must be at least the enrollment of the class. You cannot assign two classes to the same room.

Greedy strategy: Repeat until all classes are assigned, or a class cannot be included in any unassigned room. Assign the largest unassigned class to the smallest unassigned room larger than or equal to its enrollment.

Longest Increasing Subsequence: Consider the following problem. The input is an array of integers $A[1, ..n]$. The output format is a *subsequence*, a list of not necessarily consecutive elements from A in the same order they appear in A, $A[i_1], A[i_2]....A[i_k]$ with $i_1 < i_2 < ... < i_k$. The constraint is that the sequence is increasing, i.e., that $A[i_1] < A[i_2] < .. < A[i_k]$. The objective is to find an increasing subsequence of the longest possible length, i.e., maximize k.

Greedy strategy: Pick the smallest element in the array, and make it the first element of the list. Delete all elements that occur earlier in the array. Repeat until the array is empty.

Addition chains An *addition-chain* for n is a sequence of numbers starting with 1 and ending with n so that any element of the sequence other than 1 is the sum of two (not necessarily distinct) earlier elements. For example, two addition-chains for 13 are $1, 2 = 1 + 1, 4 = 2 + 2, 8 = 4 + 4, 9 = 8 + 1, 13 = 9 + 4$ and $1, 2 = 1 + 1, 3 = 1 + 2, 5 = 3 + 2, 8 = 5 + 3, 13 = 8 + 5$. The *cost* of an addition-chain is the total number of additions, or, equivalently, the number of elements -1. So both addition chains above have cost 5.

For each member of your group, take the least significant digit in your student ID. Sum them up and add forty. Use a backtracking approach to find the smallest addition chain for values of n between this number and this number plus 9. Turn in the smallest chain for each value of n. Hint: using a greedy approach, not backtracking, will usually result in a chain that's too long. On the other hand, you'll need to put in some pruning rules or your backtracking algorithm will not run in a reasonable amount of time.

Extra credit (3 points on final grade) if you can prove that your algorithm runs in time polynomial in n.