# EECS 3101: Design and Analysis of Algorithms
## Tutorial 7 – Solutions

1. Given lists of positive integers $A = a_1 \dots a_n, B = b_1, \dots, b_n$, describe a greedy algorithm to rearrange $A, B$ so as to maximize $\prod_{i=1}^{n} a_i^{b_i}$.

   **Solution:**

   Algorithm: sort both $A, B$ in non-increasing order.

   Correctness proof: Suppose there is a optimal solution different from that produced by the greedy algorithm above. Look at the greedy solution and consider the first place $j$ it differs from the optimal solution. Since the quantity being maximized always contains $\sum_{i=1}^{n} b_i$ factors, the greedy solution uses $b_j$ factors of $a_j$ while the optimal solution uses fewer $a_j$ factors. So the optimal solutions uses a smaller number to make up the same number of factors implying that it must be of lower magnitude than the greedy solution, which contradicts its optimality.

2. Clickomania: `https://icpcarchive.ecs.baylor.edu/external/45/4564.pdf`

   **Solution:** This is a problem where there is no maximization – instead the solution is TRUE or FALSE.

   The key to this problem is given in the decription

   > Some facts are known about solvable puzzles:
   > 1. The empty string is solvable.
   > 2. If x and y are solvable puzzles, so are xy, AxA, and AxAyA for any uppercase letter A.
   > 3. All other puzzles not covered by the rules above are unsolvable.

   This suggests we can construct a dynamic programming solution. Suppose we denote by $S[i, j]$ the solvability of the string $A[i \dots j]$ of the input. Then we can write the following recurrence for $S[i, j]$:

   $$\begin{aligned} S[i,j] &= T \text{ if } i > j \\ &= F \text{ if } i = j \\ &= [(A[i] = A[j]) \wedge S[i+1, j-1]] \\ &\quad \vee_{k=i+1}^{j-1} [(A[i] = A[k] = A[j]) \wedge S[i+1, k-1] \wedge S[k+1, j-1]] \end{aligned}$$

   Much like the problem on optimal matrix multiplication this problem fills the table of $S[i, j]$ values starting from the main NW-SE diagonal (corresponding to $i = j$) and filling in similar diagonals (corresponding to $j - i =$ constant) to the top right corner (corresponding to $i = 1, j = n$.

   The proof of correctness in trivial and is omitted. The running time is $O(n^3)$, since we fil in a table of $n^2$ entries and each entry is computed in $O(n)$ time. It can be shown that the algorithm takes $\Theta(n^3)$ time, but the proof is omitted.

3. Given binary strings $x, y, z$, design a dynamic programming algorithm that checks if $z$ is an interleaving of $x, y$.

   Not solved in class.