1 Let P be a set of n points evenly distributed on the unit circle, and let S be a set of m line segments with endpoints in P. The endpoints of the m segments are not necessarily distinct; n could be significantly smaller than $2m$.

(a) Describe an algorithm to find the size of the largest subset of segments in S such that every pair is disjoint. Two segments are disjoint if they do not intersect even at their endpoints.

(b) Describe an algorithm to find the size of the largest subset of segments in S such that every pair is interior-disjoint. Two segments are interiordisjoint if their intersection is either empty or an endpoint of both segments.

(c) Describe an algorithm to find the size of the largest subset of segments in S such that every pair intersects.

(d) Describe an algorithm to find the size of the largest subset of segments in S such that every pair crosses. Two segments cross if they intersect but not at their endpoints.

For full credit, all four algorithms should run in $O(mn)$ time

2 Consider the following weighted DAG G:

- The vertices are $[\![m]\!] \times [\![n]\!] \cup \{s, t\}$.
- For each $i \in \{1, \ldots, m-1\}$ and $j, j' \in [\![n]\!]$, there is an edge $(i, j) \to (i+1, j')$ with weight $f(i, j, j')$, for some function f that can be evaluated in constant time.
- For each $j \in [\![n]\!]$, there is an edge from s to $(1, j)$ with weight g_j , and an edge from (m, j) to t with weight h_j, for some given values $g_1, \ldots, g_n$,$h_1, \ldots, h_n$.

We want to compute a shortest path from s to t in G. Here, we really want to output an optimal path, not just the optimal total weight.

For full credit, use space close to $O(m + n)$, up to some logarithmic factors. The running time should remain close to $O(mn^2)$, up to some logarithmic factors.

3 Let $D[1 : n]$ be an array of digits, each an integer between 0 and 9. A *digital subsequence* of S is a sequence of positive integers composed in the usual way from disjoint substrings of D. For example, the sequence $3, 4, 5, 6, 8, 9, 32, 38, 46, 64, 83, 279$ is a digital subsequence of the first several digits of π:

$$3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3, 2, 3, 8, 4, 6, 2, 6, 4, 3, 3, 8, 3, 2, 7, 9$$

The length of a digital subsequence is the number of integers it contains, not the number of digits; the preceding example has length 12. As usual, a digital subsequence is increasing if each number is larger than its predecessor.

Describe and analyze an ecient algorithm to compute the longest increasing digital subsequence of D. *[Hint: Be careful about your computational assumptions. How long does it take to compare two k-digit numbers?]*

For full credit, your algorithm should run in $O(n^4)$ time; faster algorithms are worth extra credit. The fastest algorithm I know for this problem runs in $O(n^{3/2} \log n)$ time; achieving this bound requires several tricks, both in the design of the algorithm and in its analysis, but nothing outside the scope of this class. (I believe the running time can be reduced to $O(n^{3/2} \log \log n)$, but I haven't worked out the details).

4 Consider the following variant of the classical Tower of Hanoi problem. As usual, there are n disks with distinct sizes, placed on three pegs numbered 0, 1, and 2. Initially, all n disks are on peg 0, sorted by size from smallest on top to largest on bottom. Our goal is to move all the disks to peg 2. In a single step, we can move the highest disk on any peg to a different peg, provided we satisfy two constraints. First, we must never place a smaller disk on top of a larger disk. Second — and this is the non-standard part — *we must never move a disk directly from peg* 0 *to peg* 2.

Describe and analyze an algorithm to compute the exact number of moves required to move all n disks from peg 0 to peg 2, subject to the stated restrictions. For full credit, your algorithm should use only $O(\log n)$ arithmetic operations in the worst case. For the sake of analysis, assume that adding or multiplying two k-digit numbers requires $O(k)$ time. *[Hint: Matrices!]*

5 A basic arithmetic expression is composed of characters from the set $\{1, +, \times\}$ and parentheses. Almost every integer can be represented by more than one basic arithmetic expression. For example, all of the following basic arithmetic expression represent the integer 14:

$$1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1$$
$$((1+1)\times(1+1+1+1+1))+((1+1)\times(1+1))$$
$$(1+1)\times(1+1+1+1+1+1+1)$$
$$(1+1)\times(((1+1+1)\times(1+1))+1)$$

Describe and analyze an algorithm to compute, given an integer n as input, the minimum number of 1s in a basic arithmetic expression whose value is equal to n. The number of parentheses doesn't matter, just the number of 1s. For example, when $n = 14$, your algorithm should return 8, for the final expression above. The running time of your algorithm should be bounded by a small polynomial function of n.

6 Describe an algorithm to compute the edit distance between two strings $A[1 : m]$ and $B[1 : n]$ in $O(m \log m + n \log n + K^2)$ time, where K is the number of match points (i.e. $A[i] = B[j]$).