Dynamic Programming

1. Suppose that you are the instructor for an algorithms course. As part of an assignment, you have asked your students to write code that returns a longest palindromic subsequence of a string. When you realize this means that you have grade more than 200 answers, you decide to make them write code so that you can grade it automatically. Naturally, you develop some test cases and check that the students' answer have three key properties : a) that it is of the maximum length, b) that it is palindromic, c) that it is in fact a subsequence.

   How would you go about the third point? You may use dynamic programming, or better still don't and come up with a linear algorithm.

2. [HT: Santosh Vempala] Peachtree Ponders vs Fulton Gaelics is a legendary rivalry in the game of hobbesball. In order to decide the champion, the two teams agree to play $2n-1$ games against each other for some natural number n. The first team to win n games is declared the winner. Based on the current strength of both teams, the probability of Ponders winning any single game is p while that of Gaelics winning is $q = 1-p$. The supporters of Ponders would like to calculate the probability that their team will be the champion. Give an algorithm to calculate this probability. Prove the correctness of your algorithm and bound its complexity.

   Hint: Let A(i, j) denote the probability that the Ponders will be the champion, when Ponders and Gaelics need to win i more matches and j more matches respectively.

FFT

1. Suppose you are giving an interviewing candidates for a developer job. One of the standard questions at your company is to ask the candidate to give an efficient algorithm for substring search. You've done this many times before and seen many correct answers, including building a finite state automaton, Knuth-Morris-Pratt, Boyer–Moore, and Rabin-Karp. Today, however, a candidate says "oh just look at the binary representation of the string and use the FFT." How might that work?

2. One way to quickly evaluate a polynomial is to use a divide-and-conquer strategy. Suppose that $A(x) = a_0 + a_1 x + \ldots a_{n-1}x^{n-1}$ is an order n (degree $n-1$) polynomial, where n is a power of two. Then, letting
   a. $A_l(x) = a_o + a_1 x + \ldots + a_{n/2-1}x^{n/2-1}$ and $A_u(x) = a_{n/2} + a_{n/2+1}x + \ldots a_{n-1}x^{n/2-1}$, we have that $A(x) = A_l(x) + x^{n/2}A_u(x)$. This then comes the basis for a recursive algorithm with the base case being where the order of the polynomial is 1.

```
def polyval(A):
    n = len(A)
    if n == 1:
        return A[0]
    return polyval(A[:n/2]) + x**(n/2) * polyval(A[n/2:])
```

Note that the needed powers $x^1, x^2, x^4, \ldots x^{n/2}$ can be actually be precomputed in $O(\log n)$ time and used in all of the recursive calls. After this precomputation, the time to solve evaluate a polynomial of order n is $T(n) = 2T(n/2) + O(1) = O(n)$ by the master theorem.

Adopt this strategy to the composition of polynomials A with B, i.e. $A(B(x))$. For simplicity assume that both A and B are both order n where n is a power of 2.

How long does it take to compute the powers $B^1(x), B^2(x), B^4(x), \ldots B^{n/2}(x)$?
How long does it take to combine the results of the recursive calls?
Write down the recurrence for the running time.
Apply the master theorem.

Maximum Flow

1. Suppose that a flow network (V, E, c, s, t) admits a flow f, where $v(f) > 0$.

   Show that the network must have positive flow path. That is to say, there is an s-t path p where $f(u,v) > 0$, for every $(u,v) \in p$.

   Consider a flow f', where $f'(u,v) \leq f(u,v)$ for every edge (u,v). Show that $f - f'$ (traditional function definition) is a flow in the same network. What is its flow value?

   Show that every maximum flow can be expressed as a sum of at most $|E|$ augmenting paths. Hint: Note that our algorithms used more paths than this. Exploit the fact that you know the maximum flow.

2. Confirm that augmenting a flow f in a network by a residual flow f' produces a valid flow. That is, the capacity constraints must be met and the flow must be conserved at every vertex.

   Recall that the residual capacity is defined to be $c_f(u,v) = c(u,v) - f(u,v)$ if $(u,v) \in E$, $c_f(u,v) = f(v,u)$ if $(v,u) \in E$, and $c_f(u,v) = 0$ otherwise.

The augmentation of the flow f by f' is defined as $(f+f')(u,v) = f(u,v) + f'(u,v) - f'(v,u)$ if $(u,v) \in E$ and as zero otherwise.

Suggestion: Multiply terms by $[(u,v) \in E]$, which is 1 if the statement inside the brackets is true and 0 otherwise.

3. Consider an $n \times n$ grid where each square contains a number 0,1,2,3,4. The task is to highlight a subset of the the edges between adjacent squares so that for each square the number of highlighted edges touching the square is equal to the number inside of the square. An example is given below. Explain how you would cast this as a maximum flow problem.

1	0	2	1
1	1	1	0
0	3	3	1
0	1	1	0

Linear Programming and Duality

1. Find the dual of the following linear program
   maximize $2x_1 + x_2 + 4x_3$
   subject to $x_1 + x_2 + 2x_3 = 3$
   $2x_1 + x_2 + 3x_3 = 5$
   $x_1, x_2, x_3 \geq 0$

2. Let $G = (V, E)$ be a directed graph where each edge has an associated length l_{ij}. Express the problem of finding a shortest path from vertex s to vertex t as a linear program. Compute the dual and see if you can find an intuitive interpretation.

3. A square matrix B is unimodular if it has determinant $+1$ or -1. One consequence is that the solution to $Bx = b$ is integral (all integers) when b is integral. (Equivalently, the vector $x = B^{-1}b$ is integral when b is). This follows from Cramer's rule.

   A matrix is said to be *totally unimodular* if every $t \times t$ submatrix has determinant in $\{-1, 0, 1\}$. Thus, in a linear program where the constraints are of the form $Ax = b$ and

A is totally unimodular, every choice of basis results in a unimodular matrix B. It follows that all the basic solutions will be integral.

The incidence matrix M of a directed graph is a $|V| \times |E|$ matrix where $M_{v,e} = 1$ if e leaves v, $M_{v,e} = -1$ if e enters v, and $M_{v,e} = 0$ otherwise. Note that each column of M has exactly one $+1$ and one -1 entry.

Show that M is totally unimodular. Argue by induction starting from $t = 1$. Consider the three cases: the submatrix has a zero column, the submatrix has a column with one non-zero entry, every column in the submatrix has two non-zero entries.

Use this to show that the constraint matrix for the maximum flow problem is totally unimodular.

Approximation Algorithms

1. CLRS 35-1, 35-6 (Feel free to reproduce for each other in Piazza).

Randomized Algorithms

1. Suppose that you roll a fair k-sided die twice with values $1, \ldots k$ on its faces. Letting X_1 and X_2 be the values obtained, what is $E[\max(X_1, X_2)]$? What is $E[\min(X_1, X_2)]$? You can check your work by confirming that $E[\max(X_1, X_2)] + E[\min(X_1, X_2)] = E[X_1] + E[X_2]$. Show that this latter equation is true regardless of how X_1 and X_2 are distributed.

2. Consider the following class Reservoir:

```
class Reservoir():
        def __init__(self):
                self.x = None #The random sample so far
                self.n = 0 #The number of elements from which x is chosen
        def add(self, y):
                self.n += 1
                if random() < 1.0/self.n: #Replacing self.x with probability 1/n.
                        self.x = y
                return self.x
```

Prove that the method add returns an element chosen uniformly at random from the set of arguments y that have been passed into add over the lifetime of an instance of the class.