Problem 1

Recall the following text search problem from class. There is an alphabet \( \Sigma \) and two strings \( P \in \Sigma^m, T \in \Sigma^n, n \geq m \) (i.e. strings of length \( m, n \), respectively, made up of characters in \( \Sigma \)). We would like to output a list of all indices \( i \) such that \( T[i, i + 1, \ldots, i + m - 1] = P \), i.e. \( t_{i+j-1} = p_j \) for \( j = 1, \ldots, m \). In class, when \( \Sigma = \{0, 1\} \) we showed how to use the FFT to solve this problem in \( O(n \log n) \) time (assuming a computer supporting infinite precision complex arithmetic). Making the same precision assumptions:

(a) (2 points) Improve this time to \( O(n \log m) \) for \( \Sigma = \{0, 1\} \).

(b) (2 points) Deal with the case when \( P \) (but not \( T \)) has “don’t care” symbols *. A * symbol can match either a 0 or a 1. Again you should assume \( \Sigma = \{0, 1\} \).

(c) (2 points) Show how to deal with the case \( |\Sigma| > 2 \), without don’t care symbols. You can assume \( \Sigma = \{1, 2, \ldots, |\Sigma|\} \). For this part, full points will be given to solutions with runtime \( O(|\Sigma|n \log(|\Sigma|m)) \).

(d) (2 points) Give a solution for \( |\Sigma| > 2 \) with don’t care symbols (in both \( P \) and \( T \)), which can match any character in \( \Sigma = \{1, 2, \ldots, |\Sigma|\} \). Again, full points will be given to solutions with runtime \( O(|\Sigma|n \log(|\Sigma|m)) \).

(e) (2 points) Devise a new algorithm improving the runtime from (d) to \( O(n \log m) \), i.e. with no dependence on \( |\Sigma| \). Note that an \( O(n \log m) \) solution here naturally implies you don’t have to separately do parts (a) through (d), since the problem solved here is most general. **Hint:** Characters \( c, c' \) are unequal iff \( (c - c')^2 > 0 \). 

Problem 2 is a dynamic programming problems and should be viewed as having 3 parts. You should find a function \( f \) which can be computed recursively so that evaluation of \( f \) on a certain input gives the answer to the stated problem. Part (a) is to define \( f \) in words (without mention of how to compute it recursively). You should clearly state how many
parameters $f$ has, what those parameters represent, what $f$ evaluated on those parameters represents, and what parameters you should feed into $f$ to get the answer to the stated problem. Part (b) is to give a recurrence relation showing how to compute $f$ recursively. In part (c) you should give the running time and space for solving the original problem using computation of $f$ via memoization or bottom-up dynamic programming. If you need to use certain data structures to make computation of $f$ faster, you should say so. **Note:** if there are multiple solutions to solve the stated dynamic programming problem, you should describe the most time-efficient one you know. If there are multiple solutions with the same asymptotic time complexity, you should describe the implementation that gives the best asymptotic space complexity.

As an example, suppose the stated problem is that you are given a weighted directed graph with $n$ vertices $\{1, \ldots, n\}$, $m$ edges, and positive weights. The problem then might ask you to print an array of shortest-path distances $d_{u,v}$ for all vertices $u, v$. This can be solved with the Floyd-Warshall algorithm. A solution to part (a) could say that $f(u, v, k)$ is a function that takes as input vertex ID’s $u, v, k$, $f(u, v, k)$ represents the length of the shortest path from $u$ to $v$ in which all intermediate vertices should be from the set $\{1, \ldots, k\}$, and finally the answer $d_{u,v}$ is $f(u, v, n)$. Then part (b) should give a recurrence relation to compute $f$ (see page 6 of Lecture Notes 5 on the course website). Part (c) in this case should argue why the running time is $\Theta(n^3)$ and how to get space $\Theta(n^2)$ using bottom-up dynamic programming.

**Problem 2**

You own a $3 \times n$ rectangular oil field divided up into $1 \times 1$ square parcels. By driving a seismic vibrator over each square parcel (see Figure 1), you were able to discover exactly which parcels have oil and which don’t. You thus now have as input a $3 \times n$ boolean array $A$ where $A[i][j]$ is 1 if the $(i, j)$ parcel has oil and is 0 otherwise.

You are in possession of 3 different types of drills. The $k$th drill can drill a $k \times k$
subsquare of your rectangular oil field in one go, and will collect all oil from the subsquare drilled. Naturally larger drills cost more fuel to use, so not all drills cost the same price to use. Using the $k$th drill costs $c_k$ dollars to operate.

Given $A$ as input, calculate the least cost required to drill all the oil in your field.

**Problem 3**

<table>
<thead>
<tr>
<th>Algorithm SimpleFactorial($n$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. if $n = 0$: return 1</td>
</tr>
<tr>
<td>2. else: return Multiply($n$, SimpleFactorial($n - 1$))</td>
</tr>
</tbody>
</table>

Figure 2: Simple recursive code to compute factorial.

<table>
<thead>
<tr>
<th>Algorithm Helper($a, b$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. if $a = b$: return $a$</td>
</tr>
<tr>
<td>2. else: return Multiply(Helper($a, [(a + b)/2]$), Helper([(a + b)/2] + 1, $b$))</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm FancyFactorial($n$):</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. return Helper(1, $n$)</td>
</tr>
</tbody>
</table>

Figure 3: Divide-and-conquer approach to compute the factorial.

In the problem parts below, $M(r)$ is the runtime to multiply two $r$-digit numbers, which you should assume satisfies $M(r) = \Theta(r^c)$ for some constant $1 \leq c \leq 2$.

(a) (3 points) Suppose Multiply is an algorithm to multiply two positive integers running in time $M(r)$. What is then the asymptotic runtime of SimpleFactorial above, in terms of $M(r)$? What is this bound when $M(r) = \Theta(r^2)$, i.e. using the grade-school algorithm? How about when using Karatsuba?

(b) (4 points) The idea behind FancyFactorial is to write

$$1 \times 2 \times 3 \cdots \times n = (1 \times 2 \times \cdots \times \left\lfloor \frac{n}{2} \right\rfloor) \cdot (\left\lfloor \frac{n}{2} \right\rfloor + 1) \times \cdots \times n)$$

then recursively compute $A$ and $B$. What is the asymptotic runtime to compute FancyFactorial($n$) in terms of $n$ and $M(n)$? **Hint:** Let $T(k)$ denote the running time to compute the product of $k$ numbers that are each $O(\log n)$ digits using a similar kind of decomposition, then upper bound the runtime we wish to bound in terms of $T(\cdot)$.
(c) (3 points) What is the runtime of **FancyFactorial** when using the grade-school algorithm? How about when using Karatsuba?

**Problem 4: Programming Problem**

Solve **SPRINGBREAK** on the programming server (https://cs124.seas.harvard.edu).