15-451: Algorithm Design and Analysis

Homework 2 (due Wednesday, February 08)

Directions: *Write up carefully argued solutions to the following problems. The first task is to be complete and correct. The more subtle task is to keep it simple and succinct. Your solution should be clear enough that it should explain to someone who does not already understand the answer why it works. You may use any results proven in lecture without proof. Anything else must be argued rigorously. Unless otherwise specified, all answers are expected to be given in closed form.*

0. Sequence Alignment (20 points)

One of major computational problems in genomics is to compare two DNA sequences and then to align their common parts. In aligning two sequences, you consider not only characters that match identically, but also spaces or gaps in one sequence (or, conversely, insertions in the other sequence) and mismatches, both of which can correspond to mutations. In sequence alignment, you want to find an optimal alignment that, loosely speaking, maximizes the number of matches and minimizes the number of spaces and mismatches. More formally, you can determine a score for each possible alignment by adding points for matching characters and subtracting points for spaces and mismatches.
The score for each pair of non-blank characters is specified in advance, in a table $s(a, b)$, where a and b are any two charactersin the alphabet. The entries in this table can be either positive ornegative. Negative means it's bad to line up these two characters,therefore it's better to keep them apart. Each space (or a gap) inan alignment is penalized by some fixed constant $d > 0$.
Consider the following two strings 01101 and $1010$where the penalty are given by

	0	1
0	1	-1
1	-1	3

with the alignment give below

0	1	1	0	1	$-$
$-$	1	0	$-$	1	0
-2	3	-1	-2	3	-2

Thus the penalty for this alignment is -1. Can you find a cheaper alignment?
To solve this problem given two input strings x and y, we build a matrix F such that the entry$F_{i,j}$ is the score of the optimal alignment of $x[1..i]$ and $y[1..j]$.
(a) Write down the base cases ($F_{0,j}$, $F_{i,0}$)(b)Now assume $i > 0$, $j > 0$ and write down a recurrence for $F_{i,j}$

(c)What is the asymptotic time and space complexities of computing the matrix F?
(d)How do we recover the optimal alignment itself?

1. Fitting with L_1 Fidelity (20 points)

As in class we assume we have a data set $p_1, \ldots, p_n$ and the goal in class was to find a function$f = (f_1, \ldots f_n)$ that minimized the following penalty function:

$$\min_f \left\{ \sum_{i=1}^{n} (f_i - p_i)^2 + \alpha \sum_{i=1}^{n-1} \delta(f_i, f_{i+1}) \right\}$$

Where $\delta(a, b) = 0$ if $a = b$ and 1 otherwise.The first term is often call the **fidelity** term while the second is called the **smoothness** term.There are many variants of this problem depending on the particular fidelity and/or smoothness terms.
Let's just consider using an L_1 fidelity term, that is we minimize the following:

$$\min_f \left\{ \sum_{i=1}^{n} |f_i - p_i| + \alpha \sum_{i=1}^{n-1} \delta(f_i, f_{i+1}) \right\}$$

(a)Suppose α is very large is there should be no jumps in f, i.e. $f_i = a$What is an optimal value for a and why?How fast can you find this value of a?(b)Let $\rho(p_i, \ldots, p_j)$ be the cost for an L_1 optimal fitting from part a.Show how to use DP to find an optimal f of the equation.Start by giving and analyzing an $O(n^3 \log n)$ algorithm.(c)Show how to improve the runtime of your DP algorithm to $O(n^2 \log n)$.

2. Stacks n' Queues (15 points)

Two fundamental data structures in computer science are stacks and queues. Consider simulating a stack's LIFO property using a single queue. Thepush operation can be implemented by enqueueing the new item, then dequeue and enqueue from the queue until the new item is at the front of the list. To pop, simply dequeue. One drawback of this implementation is that push takes $O(n)$ time while an ideal stack push and pop in $O(1)$ time.

(a)Devise an algorithm which pushes in amortized $o(n)$ time and pops in constant time with two queues.(b)Prove the correctness of push and pop.(c)Analyze and prove the amortized cost of push and pop operations.

3. Dynamic Array (15 points)

Suppose we want to implement a *table* data structure, where atable is like an array, except that it can grow dynamically. Inparticular, the operation Append(A, x) increases the length of $A$by 1 and inserts element x in this new position.
In Section 18.4 of CLR, they describe an implementation that doublesthe amount of memory allocated for a table every time more space isrequired. Thus, the sequence of allocation sizes will be powers oftwo: an allocation of length 1 gets doubled to 2, which gets doubledto 4, etc.
The drawback of this scheme is that almost 1/2 of the allocated spacecould be wasted. For example, a table of 33 elements would require amemory allocation large enough to hold 64 elements. 31 of theselocations would be wasted.
One way to reduce the wasted space would be have the memory managerallocate blocks whose sizes are

perfect squares. At any time, a tableof size n would be stored in a block of size b^2, where $b = \lceil \sqrt{n} \rceil$. If adding an element to this table wouldyield $n + 1 > b^2$, then we would get a block of size $(b + 1)^2$ fromthe allocator, copy the n elements over to the new block, and addthe new element. The "cost" of this Append would be n. Thecost of appending an element where there is already space would bejust 1. (a)Prove that the amount of wasted space for a table of size n would be$O(\sqrt{n})$.(b)Prove that for any sequence of n Append operations starting from anempty table, the amortized cost of any Append operation would be$O(\sqrt{n})$.