CS3933 Fall 2007 – Solo Assignment 4
Due Wednesday November 21, 2007, at 5pm.
Sample Solution

Note that, even though desired running times are not specified, faster algorithms are better and thus worth more marks.

1. (13 marks)
   (appeared in the recent Programming Challenge – easier to do once you know it's a dynamic programming problem)

   In the two-player game "Two Ends", an even number of cards is laid out in a row. On each card, face up, is written a positive integer. Players take turns removing a card from either end of the row and placing the card on their pile. The player whose cards add up to the highest number wins the game. One strategy is to always pick the largest of the two cards (and if they are the same, pick the leftmost card); this would be the greedy approach. However, there is the potential to do better than the greedy strategy.

   You are playing this game, and you are the first player. You know that the second player is going to always use the greedy strategy.

   (a) (8 marks) Design and write a dynamic programming algorithm that, given a sequence A of n positive integers, will determine the maximum number of points that you can win by.

   Let $M[1..n][1..n]$ be a 2-dimensional table of integers.
   for i from 1 to $n-1$
    if $A[i] > A[i+1]$
     $M[i][i+1] \leftarrow A[i] - A[i+1]$
    else
     $M[i][i+1] \leftarrow A[i+1] - A[i]$
   for i from n down to 1
    for j from $i+3$ to n, increasing by 2
     if $A[i+1] \geq A[j]$
      $M[i][j] \leftarrow A[i] - A[i+1] + M[i+2][j]$
     else
      $M[i][j] \leftarrow A[i] - A[j] + M[i+1][j-1]$
     if $A[i] \geq A[j-1]$

$temp \leftarrow A[j] - A[i] + M[i+1][j-1]$

else

$temp \leftarrow A[j] - A[j-1] + M[i][j-2]$

if $M[i][j] < temp$

$M[i][j] \leftarrow temp$

return $M[1][n]$

(b) (5 marks) Asymptotically analyse the time complexity of your algorithm.

Time Analysis:

Upper bound:

The first for loop iterates $n-1$ times. The second for loop, separate from the first, iterates n times, and for each of these iterations, the third for loop (nested inside the second) iterates at most $\frac{n}{2}$ times. The rest of the algorithm contributes a constant factor, so the algorithm as a whole runs in time $\in O(n+n\cdot\frac{n}{2}) = O(n^2)$.

Lower bound:

Consider only the pair of nested for loops and the iterations of the outer loop where $i \leq \frac{n}{2}$. There will be at least $\frac{n}{2}$ such iterations of the outer loop. Since the lowest j value for the inner loop is $i+3$, and j increases by 2 for each iteration until reaching n, the number of iterations of the inner loop (for each iteration of the outer loop) is at least $\frac{n}{4}-2$. Therefore the algorithm runs in time $\in \Omega(\frac{n}{2}\cdot(\frac{n}{4}-2)) = \Omega(n^2)$.

Since this algorithm runs in time $\in O(n^2)$ and $\in \Omega(n^2)$, it runs in time $\in \Theta(n^2)$.

2. (12 marks) You are at the northwest corner of the downtown of a city that is laid out in a grid of roads. The grid is composed of n east-west roads and m north-south roads. You want to go to the southeastern corner of the grid by one of the many shortest paths, so you will travel from your source to your destination without going either west or north.

Since the grid is complete, you have many options. However, each intersection (x, y) in the grid has been assigned a positive *scenic value* $s(x, y)$, and you want to find the path that it the most scenic, ie. the sum of the $s(x, y)$ values of the intersections in the path is the highest.

(a) (6 marks) Design and write a dynamic programming algorithm that, given n, m, and $n \cdot m$ values $s(x, y)$, will find the maximum total scenic value of a path from NW to SE.

Let $M[1..n][1..m]$ be a 2-dimensional table of values.

```
M[1][1] ← s(1,1)
for j from 2 to m
    M[1][j] ← M[1][j − 1] + s(1, j)
for i from 2 to n
    M[i][1] ← M[i − 1][1] + s(i, 1)
    for j from 2 to m
        M[i][j] ← M[i − 1][j] + s(i, j)
        if M[i][j] < M[i][j − 1] + s(i, j)
                M[i][j] ← M[i][j − 1] + s(i, j)
return M[n][m]
```

(b) (3 marks) Adapt your algorithm from (a) to find the (or one of the, in case of ties) most scenic path.

```
Let M[1..n][1..m] be a 2-dimensional table of values.
Let T[1..n][1..n] be a 2-dimensional table of labels.
M[1][1] ← s(1,1)
for j from 2 to m
    M[1][j] ← M[1][j − 1] + s(1, j)
    T[1][j] ← W
for i from 2 to n
    M[i][1] ← M[i − 1][1] + s(i, 1)
    t[i][1] ← N
    for j from 2 to m
        M[i][j] ← M[i − 1][j] + s(i, j)
        T[i][j] ← N
        if M[i][j] < M[i][j − 1] + s(i, j)
                M[i][j] ← M[i][j − 1] + s(i, j)
                T[i][j] ← W
Trace(T, n, m)
return M[n][m]

Trace(T, i, j)
if i > 1 or j > 1
    if T[i][j] = N
        Trace(T, i − 1, j)
    else
        Trace(T, i, j − 1)
print (i, j)
```

(c) (3 marks) Asymptotically analyse the time complexity of your algorithm, preferably the one from (b).

Time Analysis:

The tabulation computation consists of 3 for loops, with the first loop separate and the third loop nested inside the second loop. The first loop iterates $m - 1$ times. The second loop iterates $n - 1$ times, and, for each of these iterations, the inner loop iterates $m - 1$ times. The rest of the computation provides a constant factor, so the tabulation computation runs in time $\in \Theta(m + n \cdot m) = \Theta(nm)$.

$Trace$ recurses once per instantiation, and subtracts 1 from either i or j each time, stopping when they are both 0 (from initial values of n and m). The recursion depth is thus $n+m$ with a branch factor of 1, and each instantiation takes constant time, so $Trace(T, n, m)$ takes time $\in \Theta(n + m)$.

The running time of the entire algorithm is thus $\in \Theta(nm + n + m) = \Theta(nm)$.