CISC-365*
Test #2
October 17, 2008

Student Number (Required) _____

Name (Optional) _____

This is a closed book test. You may not refer to any resources.

This is a 50 minute test.

Please write your answers in ink. Pencil answers will be marked, but may not be reconsidered after the test papers have been returned.

The test will be marked out of 50.

Question 1	/30
Question 2	/20
TOTAL	/50

Question 1 (30 marks)

The team of programmers that you supervise has been promoted. Low stress tasks are no longer on the table – your team now only handles high-stress tasks (which require one week off in preparation) and killer-stress tasks (which require two weeks off in preparation). Your job once again is to find a schedule for the next n weeks that will maximize the profit to the company.

For each week i, you are given the profit values High(i) and Killer(i) that will come from completing the respective task.

In Week 0, the team can do either task. In Week 1, the team can do either task provided they did not work in Week 0. In Week 2, the team can do the high-stress task if they took Week 1 off, and they can do the killer-stress task if they took both Week 0 and Week 1 off (etc.)

For example, suppose the upcoming task profits are

Week	0	1	2	3
High-stress profit	10	30	15	20
Killer-stress profit	5	20	40	60

Here the optimal solution is to do the high-stress task in Week 0, take Weeks 1 and 2 off, then do the killer-stress task in Week 3, with a total profit of 10+60 = 70.

Construct a Dynamic Programming solution to this problem.

a) Explain how this problem satisfies the Principle of Optimality (Your explanation must be clear, but a rigourous proof is not required).

Suppose that the last task in an optimal solution O is one of the two tasks in week i. Then the rest of the optimal solution must consist of tasks that complete by week i-2 (if the final task is high-stress) or week i-3 (if the final task is killer-stress). In either case, it is clear that the solution contained in O for this subproblem must be optimal – if it were not, we could substitute an optimal solution for this subproblem and thereby improve on O ... but since O is optimal, it cannot be improved.

Thus the problems satisfies the Principle of Optimality.

b) Give the recurrence relation for this problem, and explain your answer.

Let T(i) be the optimal value we can obtain when choosing tasks from weeks 0 through i

T(0) = max{High(0), Killer(0)} **- if we are just considering week 0, we can choose either task**

T(1) = max{T(0), **- do no task in Week 1, or**
High(1), **- we can do the high-stress task in Week 1, or**
Killer(1) **- the killer-stress task in Week 1**
}

T(2) = max{T(1), **- do no task in Week 2, or**
High(2) + T(0), **- do High(2), and whatever is optimal in Week 0, or**
Killer(2) **- we can do Killer(2), but no more**
}

T(i) = max{T(i-1), **- do no task in Week i, or**
High(i) + T(i-2), **- do High(i), and whatever is optimal in Week(i-2), or**
Killer(i) + T(i-3) **- do Killer(i), and whatever is optimal in Week(i-3)**
}

c) Explain and justify the order in which you will compute solutions to subproblems. If you plan to use a table to store solutions to subproblems, this is the place to describe it.

Since each value of T(i) depends only on the values of T(i-1) and T(i-2), the values will be computed in the order T(0), T(1), T(2) ...

A simple 1 dimensional array will be used to store these values.

d) Explain how you will determine the details of the optimal solution.

Once the value of the optimal solution is known (it will be the value of T(n-1)), we can detemine by examination whether this value is equal to T(n-2), High(n-1) + T(n-3), or Killer(n-1)+T(n-4) – this tells us which task (if any) we should choose in Week n-1. Once we know which task we choose in Week n-1, we step back to the appropriate preceding week and continue to determine which tasks are included in the optimal solution.

Question 2 (20 marks)

You have been elected to organize the COMPSA Canoe Trip down the NottaLottaWatta River. Canoes are available for rent at a sequence of trading posts along the river. You will start the trip by renting a canoe at Post 0 (where the river begins) and end the trip in Post n (the end of the river). However you don't have to keep the same canoe the whole way – you can stop at any post, drop off the canoe you have and rent another one. You can only travel downstream. For all pairs (a,b) with a < b, the cost of renting a canoe at Post a and dropping it off at Post b is given by Cost(a,b).

For example if there are four posts in total, the costs might be

Cost(a,b) matrix		Post b			
		0	1	2	3
Post a	0	x	10	15	50
	1	x	x	40	20
	2	x	x	x	35
	3	x	x	x	x

Your job is to plan the sequence of canoe rentals to minimize the total cost. In the example shown, the optimal solution is to rent a canoe from Post 0 to Post 1 (cost 10), then rent a canoe from Post 1 to Post 3 (cost 20) with a total cost of 30.

Consider the following recurrence relation:

Opt(1) = Cost(0,1)

for x > 1

```
Opt(x) = min { Cost(0,x),
               Opt(1) + Cost(1,x),
               Opt(2) + Cost(2,x),
                ...,
               Opt(x-1) + Cost(x-1,x)
             }
```

a) Explain why this recurrence relation correctly computes the minimum cost of reaching trading Post x.

To get to Post x, we either go there directly (costing Cost(0,x)) or we make one or more stops. If our last stop before Post x is Post i, then the cost is the sum of Cost(i,x) plus the cheapest way of getting from Post 0 to Post i – which is given by Opt(i). Since we don't know beforehand what the value of i should be, we consider all possible values for i {1, 2, 3, ... i-1} and choose the minimum of all the possibilities ... which is exactly what the recurrence relation does.

b) Determine the computational complexity of using a Dynamic Programming approach to solve this problem, based on the recurrence relation given above. Explain your answer.

Opt(x) requires the comparison of x different values, each of which is computed in constant time (requiring at most the addition of two previously computed values). Thus the calculation of Opt(x) requires c*x time. When we sum the calculation time for all the required values of Opt(x), we see that it is c*(1+2+3+ ... + n), which we know is O(n^2).

Tracing back to extract the details of the optimal solution also takes O(n^2) time, since at each point we must consider O(n) possibilities.

Special Bonus Question: (0 marks)

What is the meaning of the figure above?