CSE 102
Midterm 2 Review
Solutions to Selected Problems

1. Determine the largest integer k such that if there is a way to multiply 3×3 matrices using k multiplications (not assuming commutativity of multiplication), then there is an algorithm to multiply $n \times n$ matrices (where n is an exact power of 3) in time $o\left(n^{\lg(7)}\right)$. What would the run time of this algorithm be? (Hint: Proceed as in the analysis of Strassen's algorithm, but recur on submatrices of size $\frac{n}{3} \times \frac{n}{3}$.)

   Solution:
   This is hw4 problem #4, solution posted on webpage.

2. **Coin Changing Problem** Suppose you are given an unlimited supply of coins in each of the n denominations $d = (d_1, d_2, \ldots, d_n)$, and a number N of monetary units to be paid out using the least possible number of coins.

   a. Show that this problem exhibits *optimal substructure*, i.e. show how to divide the problem into subinstances of the same problem in such a way that every subinstance solution is a combination of other subinstance solutions.
   b. Write a recurrence relation for the *value of an optimal solution*. Include boundary and out of bounds values.
   c. Write a dynamic programming algorithm that fills in a table of values defined by the recurrence relation you wrote in (b). Your algorithm should take as input the vector *d* and the number *N*, and return the *value of an optimal solution*, i.e. the least number of coins necessary to pay *N* units, or returns ∞ if it is not possible to disburse *N* units using denominations *d*.
   d. Write a recursive procedure that, given the table of sub-instance solutions generated by the algorithm in (c), prints out a list of coins to be disbursed, i.e. print the *optimum solution itself*.

   Solution:
   For parts (a) and (b) see lecture notes 2-11-20 pages 8-12, and 2-13-20 pages 1-3. For parts (c) and (d) see the java program posted at https://classes.soe.ucsc.edu/cse102/Winter20/Handouts/CoinChange.java

3. **Discrete Knapsack Problem** Given n objects with values $v = (v_1, v_2, \ldots, v_n)$ and corresponding weights $w = (w_1, w_2, \ldots, w_n)$, a thief wishes to steal a subset of the objects of maximum total value, and whose total weight does not exceed the capacity W of his knapsack.

   a. Show that this problem exhibits *optimal substructure*, i.e. show how to divide the problem into subinstances of the same problem in such a way that every subinstance solution is a combination of other subinstance solutions.
   b. Write a recurrence relation for the *value of an optimal solution*. Include boundary and out of bounds values.
   c. Write a dynamic programming algorithm that takes the vectors *v* and *w* and the number *W*, and returns the *value of an optimal solution*, i.e. the maximum value that can be stolen.
   d. Write a recursive algorithm that, given the table of sub-instance solutions generated by the algorithm in (c), prints out the *optimum solution itself*, i.e. prints out a list of which objects to steal.

   Solution:
   For parts (a) and (b) see lecture notes 2-13-20, pages 3-10. For parts (c) and (d) see the java program posted at https://classes.soe.ucsc.edu/cse102/Winter20/Handouts/Knapsack.java

4. **Shortest Path Problem** State and prove a theorem the establishes that the *principle of optimality* holds for the Shortest-Path (SP) problem. (Input: a graph and two vertices (G, u, v). Output: the length of a shortest u-v path in G.) In other words, explain how and why a shortest path is composed of shortest paths.

   Solution:
   See lecture notes from 2-18-20 pages 1-5.

5. **Matrix-Chain Multiplication Problem** State and prove a theorem that establishes that the *principle optimality* holds for the Matrix-Chain Multiplication problem. (Input: $p = (p_0, p_1, p_2, \ldots p_n)$ giving the sizes $p_0 \times p_1, p_1 \times p_2, \ldots p_{n-1} \times p_n$ of the matrices in a Matrix-Chain product $A_1 \cdot A_2 \cdots A_n$. Output: a parenthesization of the product that minimizes the number of real number multiplications that must be performed to compute the product.) In other words, show how and why an optimal parenthesization is composed of optimal parenthesizations of subproducts.

   Theorem
   Suppose the matrix-chain subproduct $A_i \cdots A_j$ is optimally parenthesized. Assume also that the top level multiplication in this chain takes place at position k: $(A_i \cdots A_k) \cdot (A_{k+1} \cdots A_j)$, where $1 \le i \le k < j \le n$. Then both subproducts $(A_i \cdots A_k)$ and $(A_{k+1} \cdots A_j)$ are also optimally parenthesized.

   Proof:
   Define costs a, b and c to be

   $$c = \#\text{ of real multiplications to compute the optimally parenthesized chain } A_i \cdots A_j$$
   $$a = \#\text{ of real multiplcations to compute the subchain } (A_i \cdots A_k)$$
   $$b = \#\text{ of real multiplications to compute the subchain } (A_{k+1} \cdots A_j)$$

   Then $c = a + p_{i-1}p_k p_j + b$. Assume, to get a contradiction, that the subchain $(A_i \cdots A_k)$ can be parenthesized at a cost less than a, say with number of multiplications $a' < a$. Replace the parenthesization of the subchain $(A_i \cdots A_k)$ in our optimal parenthesization of $A_i \cdots A_j$, with this lower cost parenthesization. We get a new parenthesization of $A_i \cdots A_j$ with cost

   $$a' + p_{i-1}p_k p_j + b < a + p_{i-1}p_k p_j + b = c.$$

   This contradicts that our original parenthesization of $A_i \cdots A_j$ was optimal. A similar contradiction arises if the subchain $(A_{k+1} \cdots A_j)$ is not optimal. Therefore both of the subchains $(A_i \cdots A_k)$ and $(A_{k+1} \cdots A_j)$ are optimally parenthesized if $A_i \cdots A_j$ is. ■

6. Find an optimal parenthesization of a matrix-chain product whose sequence of dimensions is $(5, 10, 3, 12, 5, 50, 6)$. Show the complete tables $m[i, j]$ and $s[i, j]$ described in the handout on Dynamic Programming.

Solution:
In this case $n = 6$ and $p[0 \cdots 6] = (5, 10, 3, 12, 5, 50, 6)$. Recall that

$$m[i,j] = \begin{cases} 0 & i = j \\ \min\limits_{i \le k < j} \left(m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \right) & i < j \end{cases}$$

for $1 \le i \le j \le n$. Also $s[i, j]$ is defined to be the value of k at which the minimum occurs in cell $m[i, j]$, for $1 \le i < j \le n$. With some effort, we have

$m[\ ]$		j 1	2	3	4	5	6
i	1	**0**	**150**	**330**	**405**	**1655**	**2010**
	2	-	**0**	**360**	**330**	**2430**	**1950**
	3	-	-	**0**	**180**	**930**	**1770**
	4	-	-	-	**0**	**3000**	**1860**
	5	-	-	-	-	**0**	**1500**
	6	-	-	-	-	**-**	**0**

and

$s[\ ]$		j 1	2	3	4	5	6
i	1	*	**1**	**2**	**2**	**4**	**2**
	2	-	*	**2**	**2**	**2**	**2**
	3	-	-	*	**3**	**4**	**4**
	4	-	-	-	*	**4**	**4**
	5	-	-	-	-	*	**5**
	6	-	-	-	-	**-**	*

The second table gives us the optimal parenthesization as

$$(A_1 \cdot A_2) \cdot \big((A_3 \cdot A_4) \cdot (A_5 \cdot A_6) \big)$$

■

7. **Continuous vs. Discrete Knapsack Problem** Given values $v[1 \cdots 6] = (5, 5, 9, 4, 4, 12)$, weights $w[1 \cdots 6] = (1, 4, 3, 4, 1, 6)$ and capacity $W = 9$.

   a. Suppose these data constitute an instance of the discrete knapsack problem. (Determine $x \in \{0, 1\}^6$ that maximizes $\sum_{i=1}^{6} x_i v_i$ subject to the constraint that $\sum_{i=1}^{6} x_i w_i \leq W$.) Solve this problem using dynamic programming.
   Solution:

	$V[i,j]$		j									
i	w	v	0	1	2	3	4	5	6	7	8	9
1	1	5	**0**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**	**5**
2	4	5	**0**	**5**	**5**	**5**	**5**	**10**	**10**	**10**	**10**	**10**
3	3	9	**0**	**5**	**5**	**9**	**14**	**14**	**14**	**14**	**19**	**19**
4	4	4	**0**	**5**	**5**	**9**	**14**	**14**	**14**	**14**	**19**	**19**
5	1	4	**0**	**5**	**9**	**9**	**14**	**18**	**18**	**18**	**19**	**23**
6	6	12	**0**	**5**	**9**	**9**	**14**	**18**	**18**	**18**	**21**	**23**

   Steal the following objects: **1 2 3 5** (total weight 9 and total value 23)

   b. Suppose these data constitute an instance of the continuous knapsack problem. (Determine $x \in [0, 1]^6$ that maximizes $\sum_{i=1}^{6} x_i v_i$ subject to the constraint that $\sum_{i=1}^{6} x_i w_i \leq W$.) Solve this problem using a greedy algorithm, using the selection function $f(i) = v_i / w_i$.
   Solution:
   $x = \left(1, 0, 1, 0, 1, \frac{2}{3}\right)$, i.e. steal all of objects 1, 3 and 5, and steal $2/3$ of object 6. The total value is 26, and total weight is 9

   c. Regard these data as an instance of the discrete knapsack problem again. Attempt to solve it using a greedy algorithm, using the selection function $f(i) = v_i / w_i$. If at some point in the greedy loop, you cannot include all of an object, skip it and go to the next "best" object, according to f. Does your answer have the same value that you found in part (a)?
   Solution:
   $x = (1, 1, 1, 0, 1, 0)$, i.e. steal objects 1, 2, 3 and 5. The total value is 23, and the total weight is 9. Note that this is the very same solution as that found in part (a). Note that in general, one cannot expect a greedy strategy to correctly solve all instances of the discrete version of this problem, although it does in this case.