1. (20points)
Explain the following terminologies.

a) Optimal substructure (2 points)

An optimal solution to a problem (instance) contains optimal solutions to subproblems.

b) Asymptotic bound (2points)

It's the bound when input sizes are large enough.

Indicate whether each of the following is true or false.

a) If $f(n) = \Theta(g(n))$, then $f(n) = O(g(n))$ is also true.(2points)

True

b) $T(n) = 2T(n/2) + n\lg n$ can be solved by the Master Theorem. (2points)

False

c) Both dynamic programming and greedy algorithm are recursive in nature.(2points)

True

d) Counting sort algorithm is a comparison based sorting algorithm, and it is stable.(2points)

False

e) A typical randomized algorithm needs to make assumptions on input distributions.(2points)

False

Binary search can be viewed as a divide and conquer algorithm. Please describe the tasks for divide, conquer, and combine step, respectively. And give the recurrence for the running time. (6points)

Divide: Check middle element
Conquer: Recursively search one subarray
Combine: return the result of find or not find

The recurrence is :
$T(n) = T(n/2) + \Theta(1)$

You can choose four problems from problem 2 to problem 6.

2. Solve $T(n) = 3T(n/2) + cn$, $T(1) = d$ using the recursion tree method. (20 points)

See lecture notes on recurrences.

3. Find an optimal parenthesization of a matrix-chain product whose sequence of dimensions is <5, 10, 2, 15, 4>. You need to show your work, including the recurrence relationship of your calculation, and intermediate results of m[i,j]. (20 points)

Let m[i, j] be the minimum number of scalar multiplications needed to computer the matrix $A_{i\ldots j}$. The original problem is now to solve m[1,n].

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

j

i	1	2	3	4
1	0	100	250	260
2		0	300	200
3			0	120
4				0

4. Probability Analysis (20 points)

The following program determines the maximum value in an unordered array A[1..n]. Suppose that all numbers in A are randomly drawn from the interval [0,1]. Let X be the number of times line 5 is executed. Show that $\mathrm{E}[X] = \Theta(\ln n)$.

Hint: $\sum_{i=1}^{n} \frac{1}{i} = \ln n + O(1)$

```
1   max ← 0
2   for i ← 1 to n
3       do
4          if A[i] > max
5              then max ← A[i].
```

Let X_i be the indicator random variable to indicate whether line 5 is executed for $A[i]$.

Since X_i has a $\frac{1}{i}$ chance of being the largest one in A[1..i], $\mathrm{E}[X_i] = \frac{1}{i}$.

So $\mathrm{E}[X] = E[X_1] + E[X_2] + \cdots + E[X_n] = \sum_{i=1}^{n} \frac{1}{i} = \ln n + O(1)$

5. Greedy Algorithm (20 points)
Suppose that instead of always selecting the first activity to finish, we instead select the last activity to start that is compatible with all previously selected activities. Describe how this approach is a greedy algorithm, and prove that it yields an optimal solution.

The greedy choice of this algorithm is to select the last activity to start that is compatible with all previously selected activities. We show the greedy choice property and optimal substructure property for its correctness.

Supposing a_m is the activity with the latest starting time in $S_{i,j}$, we need to prove that there is an optimal solution that contains a_m. Let $A_{i,j}$ be an optimal solution and a_k the last activity in $A_{i,j}$. If a_k is a_m we are done. Otherwise, we create $A_{i,j}'=(A_{i,j}-\{a_k\})U\{a_m\}$. a_m starts later than a_k so as every others activities were compatible with a_k, they are also compatible with a_m. As a result, $A_{i,j}'$ is a valid solution which has the same number of activities than the optimal solution $A_{i,j}$, it is also an optimal solution.

The optimal substructure property remains the same as we select the first activity to finish.

6. Dynamic Programming (20 points)
Consider the coin changing problem with n denominations D[1..n] and total amount m. Assume one of the coins is a penny. Write an $O(nm)$ time dynamic programming algorithm that will always determine the fewest number of coins needed to make exact change for the given amount. As an example, if the denominations are 1, 3, and 4, the total amount m=6, then your algorithm should compute the optimal solution uses 2 coins.

Your answer should include:
1) The recurrence relation and a clear justification for it.
2) Pseudo code for the algorithm.
3) Create the table for the instance mentioned above, and fill in the table.

1)

Let us define $c[j]$ to be the minimum number of coins we need to make change for j cents. Let the coin denominations be $d_1, d_2, \ldots, d_k$. Since one of the coins is a penny, there is a way to make change for any amount $j \geq 1$.

Because of the optimal substructure, if we knew that an optimal solution for the problem of making change for j cents used a coin of denomination d_i, we would have $c[j] = 1 + c[j - d_i]$. As base cases, we have that $c[j] = 0$ for all $j \leq 0$.

To develop a recursive formulation, we have to check all denominations, giving

$$c[j] = \begin{cases} 0 & \text{if } j \leq 0\,, \\ 1 + \min\limits_{1 \leq i \leq k} \{c[j - d_i]\} & \text{if } j > 1\,. \end{cases}$$

2)

```
COMPUTE-CHANGE(n, d, k)
for j ← 1 to n
    do c[j] ← ∞
       for i ← 1 to k
           do if j ≥ d_i and 1 + c[j − d_i] < c[j]
                 then c[j] ← 1 + c[j − d_i]
                      denom[j] ← d_i
return c and denom
```

3) c[1] = 1 c[2]=2 c[3]=1 c[4]=1 c[5]=2 c[6]=2