1. (16 points)
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.(2 points)

True

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

False

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

True

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

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. (4 points)

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 three problems from problem 2 to problem 5.

2. 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]. (18 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].

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

j

i

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

m[1,1] = m[2,2] = m[3,3] = m[4,4] = 0
m[1,2] = 5*10*2 = 100
m[2,3] = 10*2*15 = 300
m[3,4] = 2*15*4 = 120
m[1,3] = min((m[1,1]+m[2,3]+5*10*15),(m[1,2]+m[3,3]+5*2*15))
= min(1050, 250) = 250
m[2,4] = min((m[2,2]+m[3,4]+10*2*4),(m[2,3]+m[4,4]+10*15*4))
= min(200, 900) = 200
m[1,4] = min((m[1,1]+m[2,4]+5*10*4)+(m[1,2]+m[3,4]+5*2*4)+(m[1,3]+m[4,4]+5*15*4))
= min(400, 260,550) = 260

3. Probability Analysis (18 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 $\mathrm{E}[X] = E[X_1] + E[X_2] + \cdots + E[X_n] = \sum_{i=1}^{n} \frac{1}{i} = \ln n + O(1)$

4. Greedy Algorithm (18 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 by proving that the greedy choice property holds.

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.

Suppose $A_{i,j}$ contains activities that start after a_i finishes and finish before a_j starts, and a_m is the activity with the latest starting time in $A_{i,j}$, we need to prove that there is an optimal solution that contains a_m.

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

5. Dynamic Programming (18 points)

Let $S = \{a_1, a_2, \ldots, a_n\}$ be a set of n positive integers and let k be an integer. Give an $O(kn)$-time bottom-up dynamic programming algorithm to decide if there is a subset U of S that $\sum_{a_i \in U} a_i = k$. Your algorithm should return T (for 'true') if U exists and F (for 'false') otherwise.

Your answer should include:
1) The recurrence relation and a clear justification for it.
2) Pseudo code for the algorithm.

Let us define $f[i, j]$ to be a 2D boolean array. The state $f[i, j]$ will be **true** if there exists a subset of elements chosen from $S_i = \{a_1, a_2, \cdots, a_i\}$ with sum value of j. Ultimately we want to know the value of $f[n, k]$.

By making a choice on a_i, we can split the problem of $f[i, j]$ into two subproblems. When we choose some elements from S_i to form a subset, will a_i be included?

1. If a_i is included in the subset, the problem is equivalent to: if there exists a subset of $S_{i-1} = \{a_1, a_2, \cdots, a_{i-1}\}$ with sum value of $j - a_i$. In this case, we have $f[i, j] = f[i-1, j-a_i]$, and j should not be smaller than a_i.
2. If a_i is not included in the subset, the problem is equivalent to: if there exists a subset of S_{i-1} with sum value of j. In this case, we have $f[i, j] = f[i-1, j]$.

A feasible solution to either of these two subproblems results in a feasible solution to the original problem. Also we need to handle the boundary cases carefully. The above analysis gives us the recurrence

$$f[i, j] = \begin{cases} \textbf{true}, & \text{if } j = 0, \\ \textbf{false}, & \text{if } i = 0 \text{ and } j > 0, \\ f[i-1, j], & \text{if } i > 0 \text{ and } 0 < j < a_i, \\ f[i-1, j] \vee f[i-1, j-a_i], & \text{otherwise.} \end{cases}$$

Pseudo code:

```
SUBSET-SUM(a, n, k)
for i ← 0 to n
    f[i, 0] ← true
for j ← 1 to k
    f[0, j] ← false
for i ← 1 to n
    for j ← 1 to k
      if j < a[i]
          then f[i, j] ← f[i-1, j]
      else f[i, j] ← f[i-1, j] or f[i-1, j-a[i]]
return f[n, k]
```