Question 1 [4%] Show that $\log(n!) = O(n \log n)$. You must provide a proper $n_0$ and $c$ to justify your answer, Hint : $\log(a \cdot b) = \log a + \log b$.

Question 2 [4%] Is $2^{2n} = O(2^n)$? Explain your answer!

Question 3 [4%] Use the iterating method to determine an asymptotically tight upper (big $O$) bound on the recurrence $T(n) = T(n-2) + n + 5$.

Question 4 [4%] Give asymptotically tight upper bound (big $O$) for $T(n) = T(n^{1/2}) + 1$. 

Question 1 [8 points]: Maximize Number of On-time Jobs

You are given a set of \( n \) jobs. Associated with each job \( i \) is a due date \( d_i \). In order to complete job \( i \), one has to process the job \( i \) on a machine for one unit of time. Only one machine is available for processing jobs. A job \( i \) is on-time if it is completed on or before its due-date \( d_i \).

Give a greedy algorithm (pseudo code is sufficient) that finds a schedule for these jobs, which maximize the number of on-time jobs. What is the running time of your algorithm? Explain your solution!

Question 2 [8 points]

We can recursively define the number of combinations of \( m \) things of \( n \), denoted \( \binom{n}{m} \), for \( n \geq 1 \) and \( 0 \leq m \leq n \), by

\[
\binom{n}{m} = \begin{cases} 
1 & \text{if } m = 0 \text{ or } m = n \\
\binom{n-1}{m} + \binom{n-1}{m-1} & \text{if } 0 < m < n.
\end{cases}
\]

(a) Give a dynamic programming algorithm to compute \( \binom{n}{m} \).

What is the running time of your algorithm?

(b) Using the algorithm in (a), construct the table for \( \binom{6}{4} \)
**PARTITION problem**: Given a set \( Z = \{ a_1, a_2, \ldots, a_n \} \) of \( n \) positive integers.

Question: Is there a subset \( Z' \subseteq Z \) such that \( \sum Z' = \sum (Z-Z') \).

**The AIRLINE Problem**: Given

- a starting city \( X \),
- a finite set of destination cities \( D = \{ d_1, d_2, \ldots, d_m \} \),
- a set of positive integers \( T = \{ t_1, t_2, \ldots, t_m \} \), where \( t_i \) is the round trip time between \( X \) and destination \( d_i \),
- a positive integer \( K \) which is the number of aircrafts, and
- a positive integer \( M \) that is the time each aircraft can fly continuously without maintenance.

Question: Can a fleet of \( K \) aircrafts starting at \( X \) service all destinations in \( D \) without maintenance?

Note:
- an aircraft must return to \( X \) before it goes to next destination
- all aircrafts start at \( X \) and must return to \( X \) at the end of services
- all destination cities must be serviced at least once
- if \( D' \subseteq D \) is a set of destinations assigned to aircraft \( i \) and \( T' \subseteq T \) is a set of corresponding round trip times for cities in \( D' \), then \( \sum T' \leq M \)

Use PARTITION Problem to show that the AIRLINE Problem is NP-Complete.
Total: 12 points (3 points per question)

Question 1: Use iteration method to find the asymptotic tight upper bound (big O) for

\[ T(n) = T(n-a) + T(a) + O(n) \] where \( a \) is a small integer and \( a \geq 1 \).

Question 2: Show that for any real constants \( a \) and \( b \), where \( b > 0 \), \( (n+a)^b = O(n^b) \).

Note: \( a \) may be a negative number.

Question 3: Given \( n \) integers in the range of 1 to \( k \). Describe an algorithm that

- pre-processes its input in \( O(n+k) \)
- then answers any query about how many of the \( n \) integers fall into a range \([u..v]\) in \( O(1) \) where \( v \geq u \).

Question 4: Suppose that you are given an \( O(n) \) algorithm, \( \text{split}(A,p,x,r) \) where \( A \) is an array of \( n \) elements, and \( p, x \) and \( r \) are integers such that \( p \leq x < r \).

\( \text{split}(A,p,x,r) \) splits elements of \( A[p..r] \) to two sublists, \( A[p..x] \) and \( A[x+1..r] \) such that each element in \( A[p..x] \) has value smaller than each element in \( A[x+1..r] \).

Use \( \text{split}(A,p,x,r) \) to modify the quicksort so that the worst case running time is \( O(n \log n) \).

Note: you just need to give the pseudo code
Question 1 [8%]

n jobs are available at the same time and to be scheduled for execution on one machine. Associate with each job $J_i$, is a processing time $p_i$ and a weight $w_i$. The cost of processing $J_i$ is $w_i \cdot c_i$ where $c_i$ is the completion time of $J_i$. The total cost of processing all n jobs is $\Sigma (w_i \cdot c_i)$.

<table>
<thead>
<tr>
<th></th>
<th>$c_1$</th>
<th>$c_2$</th>
<th>...</th>
<th>$c_{n-2}$</th>
<th>$c_{n-1}$</th>
<th>$c_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>$J_2$</td>
<td>...</td>
<td>$J_{n-1}$</td>
<td>$J_n$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: $c_i = p_1 + p_2 + ... + p_i$

Consider the following greedy strategy: Process jobs according to nonincreasing of order of their ratios $w_i / p_i$ will minimize the total cost.

Show that there is an optimal solution such that the job with largest ratio is scheduled first.

Question 2 [8%]

Consider an n by m array of positive integers $D[1..n, 1..m]$. Give a dynamic programming algorithm to find the least cost path to walk through array $D$.

Example: A sample 4 by 5 array $D$.

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>3</th>
<th>19</th>
<th>90</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>14</td>
<td>2</td>
<td>12</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>47</td>
<td>32</td>
<td>14</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

A path covers exactly m positions (i.e. one position per column) of array $D$. It may begin at any position in the 1st column and may end at any position in the last column. At any position $D[i,j]$, it is only possible to move to one of the three positions: $D[i-1,j+1]$, $D[i,j+1]$ or $D[i+1,j+1]$ (note: assume that $D[0,j]$ and $D[n+1,j]$ are $\infty$ for all $j$). The cost of a path is the sum of integers in the positions through which it passed.
**PARTITION problem**: Given a set $Z = \{a_1, a_2, \ldots, a_n\}$ of $n$ positive integers.

Question: Is there a subset $Z' \subseteq Z$ such that $\sum Z' = \sum (Z-Z')$.

**Scheduling Jobs with Late Penalty Problem**:

Given:

- A set of $m$ jobs, one machine and a positive integer $T$.
- Associated with each job $i$ is a processing time $p(i)$, deadline $d(i)$ and late penalty $w(i)$.
- Assume all jobs are available at time 0 and to be scheduled on the machine.
- If job $i$ is completed after its deadline $d(i)$, then it incurs a late penalty $w(i)$.

Question: Is there a schedule for $m$ jobs such that the sum of the late penalties is not more than $T$? i.e. If $L$ is a set of all late jobs in a schedule, then $\sum_{i \in L} w(i) \leq T$

Use PARTITION Problem to show that the Scheduling Jobs with Late Penalty Problem is NP-Complete.
**PARTITION problem**: Given a set $Z = \{a_1, a_2, \ldots, a_n\}$ of n positive integers.

Question: Is there a subset $Z' \subseteq Z$ such that $\sum Z' = \sum (Z-Z')$.

**Scheduling Jobs with Late Penalty Problem on 3 Machines**:

Given:

- A set of m jobs, **threemachines** and a positive integer T.
- Associated with each job i is a processing time $p(i)$, deadline $d(i)$ and late penalty $w(i)$.
- Assume all jobs are available at time 0 and to be scheduled on **3machines**.
- If job i is completed after its deadline $d(i)$, then it incurs a late penalty $w(i)$.

Question: Is there a **non-preemptive** schedule for m jobs such that the sum of the late penalties is not more than T? i.e. If $L$ is a set of all late jobs in a schedule, then $\sum_{i \in L} w(i) \leq T$

Use **PARTITION Problem** to show that the Scheduling Jobs with Late Penalty on 3 Machines Problem is NP-hard.
Question 1 [25 points]  PRAM Algorithm

Given an array A of n numbers and a number x. The goal is to compute number of elements in A that are greater than x. Give an O(log n) time n processor CREW PRAM algorithm for solving this problem.

Question 2 [25%] Dynamic Programming Algorithm

Let A={a_1, a_2, ..., a_n} be a finite set of distinct coin types (e.g. a_1= 50, a_2=25, a_3=10 etc). We assume that each a_i is an integer, a_1 > a_2 > ... > a_n > 0 and each type is available in unlimited quantity.

(a) Let a_n = 1. Give a dynamic programming algorithm that makes up an exact amount C using minimum number of coins. C is an integer > 0.

(b) Using algorithm in (a), construct a table for A={8, 4, 3, 1} and C = 14.

Question 3 [25%] Show that Scheduling jobs with release times and deadlines (SJRD) problem is NP-Complete. Hint : Use Partition Problem.

**SJRD Problem**: You are given a set of n jobs. Associate with each job i is a processing time p_i, a deadline d_i and a ready time r_i. A job is not available for processing until its ready time and must be completely processed by its deadline.

Question : Is it possible to construct a nonpreemptive schedule for the given n jobs on one machine without violating any ready time and any deadline?

Note : Assume information of all jobs are available at time = 0.

**PARTITION problem** : Given a set Z = {a_1, a_2, ..., a_n} of n positive integers.

Question : Is there a subset Z' \subseteq Z such that \( \sum Z' = \sum (Z-Z') \).
Question 4 [25%] Approximation Algorithm

0/1 Knapsack Optimization Problem: Given n objects and a knapsack.

Each object i has an integer weight $w_i$ and an integer profit $p_i$.
The knapsack has a capacity $M$.
Assume $\sum w_i > M$ and each $w_i \leq M$.
If object $i$ is placed in the knapsack then $x_i = 1$, otherwise $x_i = 0$.

Objective: To obtain a filling of the knapsack that maximize the total profit earned, i.e. maximize $\sum p_i x_i$ such that $\sum w_i x_i \leq M$, where $x_i \in \{0, 1\}$ for $1 \leq i \leq n$.

From CSC810, Mr. NotVerySmart knows that 0/1 Knapsack decision problem is NP-Complete. He proposes a fast approximation algorithm for 0/1 Knapsack optimization problem. The algorithm is as follows:

Step 1. Sort objects in nonincreasing order of their weights.
Assume the final ordering is $w_1 \geq w_2 \geq \ldots \geq w_n$

Step 1. total_profit = 0; total_weight = 0; $i = 1$;

Step 2. while ( (total_weight + $w_i$) \leq M ) {
        total_weight = total_weight + $w_i$ // put the object into knapsack
        total_profit = total_profit + $p_i$ // update the total profit
        $i = i + 1$;
    }

Step 3. return total_profit;

(a) [10%] Provide an example to show that the approximation bound of Mr. NotVerySmart’s algorithm is greater than $C$, where $C$ is a positive constant, i.e. you need to show an example such that $\text{optimal_profit} / \text{approximated_profit} > C$.

(b) [15%] Mr. NotVerySmart realizes that the above algorithm has poor performance. He proposes to modify Step 1 as follow:

Step 1: Sort objects in nonincreasing order of their profits.
Assume the final ordering is $p_1 \geq p_2 \geq \ldots \geq p_n$

Show that the approximation bound of Mr. NotVerySmart’s modified algorithm is less than or equal to $M$, where $M$ is the capacity of the knapsack, i.e. you need to prove that $\text{optimal_profit} / \text{approximated_profit} \leq M$. 