1 Problem 1

The algorithm will print GCD and LCM of X and Y.
To prove: GCD(X, Y) is given by $\frac{x+y}{2}$ while LCM(X, Y) is given by $\frac{u+v}{2}$
Proof: Invariant for GCD computation are $x > 0 \land y > 0 \land GCD(X, Y) = GCD(x, y)$. Verify both steps maintain this invariant using the number theory result that $GCD(x, y) = GCD(x-y, y) = GCD(x, y-x)$. At termination we have $x = y$ and therefore $GCD(x, y) = GCD(x, x) = \frac{x+y}{2} = \frac{y+y}{2}$.
Invariant for LCM computation is $uy + vx$. Verify that both steps of algorithm maintain $uy + vx$ as invariant. Therefore from the initialization conditions when algorithm terminates $uy + vx = XY + XY$. Now since $x = y = GCD(X, Y)$ we get $\frac{u+y}{2} \times GCD(X, Y) = XY$. Using the number theory result we know that $\frac{u+y}{2}$ is the LCM.

2 Problem 2(0.1)

(a)$n-100 = \Theta(n-200)$
(b)$n^{\frac{1}{2}} = O(n^{\frac{5}{3}})$
(c)$100n + \log(n) = \Theta(n + \log(n)^2)$
(d)$\log n = \Theta(10\log 10n)$
(e)$\log 2n = \Theta(\log 3n)$
(f)$10\log n = \Theta(\log(n^2))$
(g)$n^{0.01} = \Theta(n^{0.11})$
(h)$n^{\frac{n}{\log n}} = \Omega(n^{\log \log n})$
(i)$n^{0.1} = \Omega((\log n)^{10})$
(j)$\log n^{\log n} = \Omega(\frac{n}{\log n})$
(k)$n^{\frac{n}{2}} = \Omega(\log n^3)$
(l)$n^{\frac{n}{2}} = \Theta(5^{\log n})$
(m)$n2^n = O(3^n)$
(n)$2^n = \Theta(2^{n+1})$
(o)$n! = \Omega(2^n)$
(p)$\log n^{\log n} = O(2^{\log n^2})$
(q)$\sum_{i=1}^{n} i^k = O(n^{k+1})$
3 Problem 3(0.2)

The given function \( g(n) \) turns out to be a geometric series and it evaluates to

\[
g(n) = \begin{cases} 
  \frac{c^{n+1}}{1-c^{n+1}} & \text{if } c > 1 \\
  \frac{1}{n-c} & \text{if } c < 1 \\
  n & \text{if } c = 1
\end{cases}
\]  

(1)

(a) When \( c < 1 \), lower bound on Equation 1 can be obtained by analyzing the case when \( n = 0 \) which is 1. Similarly upper bound is obtained by analyzing the case when \( n \to \infty \) which comes out to be \( \frac{1}{1-c} \). Since we got two constants bounding Equation 1 \( g(n) \) is \( \Theta(1) \)

(b) When \( c > 1 \) similar analysis will produce upper and lower bounds as \( c^n \) which makes \( g(n) \) as \( \Theta(c^n) \)

(c) When \( c = 1 \), the bound of \( \Theta(n) \) is straight forward as each term of \( g(n) \) evaluates to 1 and there are \( n \) terms.

4 Problem 4(0.3)

(a) Base case : \( \text{Fib}[6] = 8 \geq 2^3 \)
Hypothesis : \( \text{Fib}[i] \geq 2^{i/2} \forall i \in (6, \ldots, k) \)
Induction : \( \text{Fib}[k+1] = \text{Fib}[k] + \text{Fib}[k-1] \geq 2^{k/2}(1 + \sqrt{2}) \)

\( \text{Fib}[k+1] \geq 2^{k/2} \times \sqrt{2} \)
Therefore \( \text{Fib}[k+1] \geq 2^{k+1/2} \)

(b) Using the generating function derivation shown in class Fibonacci numbers can be represented as

\[
\text{Fib}[n] = \frac{1}{\sqrt{5}}(\phi^n - \varphi^n)
\]  

(2)

where \( \phi = \frac{1+\sqrt{5}}{2} \) and \( \varphi = \frac{1-\sqrt{5}}{2} \). Now choosing \( c = \log \phi \) we need to prove that \( \text{Fib}[n] \leq \phi^n \). We use induction to prove that.
Base case : \( \text{Fib}[0] = 1 \leq \phi^0 = 1 \)
Hypothesis : \( \text{Fib}[i] \leq \phi^i \forall i \in (0, \ldots, k-1) \)
Induction : \( \phi^n = \phi^2 \phi^{n-2} = (1 + \phi)\phi^{n-2} \)
Using induction hypothesis \( \phi^n \geq \text{Fib}[n-1] + \text{Fib}[n-2] = \text{Fib}[n] \)
Therefore \( c \) is given as \( \log \phi \)

(c) Any number in between 0.5 and \( \log \phi \) will suffice. Largest is \( \log \phi \).

5 Problem 5(1.31)

(a) \( N \) is an \( n \)-bit number. \( N! \) is given by 1.2.3...N.
Upper Bound : Assuming each number 1,2,3,...,\( N \) is represented by \( n \) bits, the result of multiplying \( N \) \( n \)-bit number will give a number of \( N \times n \) bits where \( N = 2^n \). Hence it is \( O(N^n) \)
Lower Bound : Since each number \( i \in (1, 2, \ldots, N) \) can be optimally represented
by \( \log i \) bits, total number of bits in \( N! \) is given by \( \sum_{i=1}^{N} \log i \) which is \( \log N! \).

Using Sterling’s approximation or using a factor argument we know \( N! \geq \frac{N^N}{e^N} \)
which implies that total number of bits in \( N! \) is lower bounded by \( N \log N \). It turns out to be \( \Omega(N^n) \). Combining both we get \( \Theta(N^n) \).

(b) A simple iterative algorithm to solve the problem is given by:

```plaintext
Input : N
Output : N!
prod = 1
for i = 1 to N
    prod = prod * i
return prod
```

Complexity analysis: We present a worst case bound. Assuming each of the number 1,2,3,...,N are \( n \)-bit long each multiplication computes product of a \( i \times n \) bit number with \( n \) bit number. Therefore total time taken by the for-loop is given by \( \sum_{i=1}^{N} (i \times n) \) which turns out to be \( O(N^2 \times n^2) \).

6 Problem 6(1.32)

Given numbers \( X \) and \( Y \), apply Euclid algorithm (page 20) to compute \( \text{GCD}(X,Y) \). Followed by that get \( \text{LCM}(X,Y) \) by \( \frac{X \times Y}{\text{GCD}(X,Y)} \), \( \text{GCD} \) computation takes \( O(n^3) \) where \( n \) are the number of bits in \( X, Y \). Final \( \text{LCM} \) computation takes \( O(n^2) \). Therefore the algorithm is bounded by \( O(n^3) \).