CMPE 300 ANALYSIS OF ALGORITHMS
MIDTERM ANSWERS

1.

a) $f(n) \in \Theta(g(n))$ since $\log n^2 = 2 \log n$.
b) $f(n) \in \Omega(g(n))$ since n^c grows faster than $c \log n$ for any c.
c) $f(n) \in \Omega(g(n))$. Dividing both sides by log n, we see that log n grows faster than 1.
d) $f(n) \in \Omega(g(n))$. If we take both f(n) and g(n) as exponents for 2, we get 2^n on one side and $(2^{\log n})^2 = n^2$ on the other, and n^2 grows slower than 2^n.
e) $f(n) \in \Omega(g(n))$. Dividing both sides by log n and throwing away the low order terms, we see that n grows faster than 1.
f) $f(n) \in O(g(n))$. $f(n) = 2 \log n$. Dividing both sides by log n, we see that log n grows faster than 2.
g) $f(n) \in \Theta(g(n))$ since log 10 and 10 are both constants.
h) $f(n) \in \Omega(g(n))$ since exponential function 2^n grows faster than polynomial function $10n^2$.
i) $f(n) \in \Omega(g(n))$. Take logarithm of both sides. $f(n) = \log 2^n = n$, $g(n) = \log (n \log n) = \log n + \log \log n$. Throwing away the low order terms, we see that n grows faster than log n.
j) $f(n) \in O(g(n))$. $3^n = 1.5^n\, 2^n$, and if we divide both sides by 2^n, we see that 1.5^n grows faster than 1.

2.

a) Master Theorem: Let x(n) be an eventually nondecreasing function that satisfies the recurrence relation

$$x(n) = a\, x(n/b) + f(n), \quad n=b^k, \text{ k is a positive integer, } x(1)=c$$

where $a\geq 1$, $b\geq 2$, $c>0$. If $f(n) \in \Theta(n^d)$, where $d\geq 0$, then

$$x(n) \in \begin{cases} \Theta(n^d) & \text{if } a<b^d \\ \Theta(n^d \log n) & \text{if } a=b^d \\ \Theta(n^{\log a}) & \text{if } a>b^d \end{cases} \quad \text{for all n.}$$

b) According to the theorem, a=3, b=5, d=2. Since $3<5^2$, $T(n) \in \Theta(n^2)$.
c) According to the theorem, a=2, b=2, d=1. Since $2=2^1$, $T(n) \in \Theta(n \log n)$.
d) By backward substitution,

$$T(n) = 2\, T(n/2) + n$$
$$= 2\, [2\, T(n/4) + n/2] + n = 2^2\, T(n/4) + 2\, n/2 + n$$
$$= 2^2\, [2\, T(n/8) + n/4] + 2\, n/2 + n = 2^3\, T(n/2^3) + 2^2\, n/2^2 + 2\, n/2 + n$$
$$\ldots$$
$$= 2^{\log n/2}\, T(2) + 2^{\log n/2 - 1}\, (n/2^{\log n/2 - 1}) + \ldots + 2\, n/2 + n$$

So, $T(n) = \sum_{i=0}^{\log n} 2^i 2^{\log n - i} = \sum_{i=0}^{\log n} 2^{\log n} = \sum_{i=0}^{\log n} n = n(\log n+1) \in \Theta(n \log n)$

3. (See the lecture notes)

4.

```
function CountSort (L[1:n], Out[1:n], k)
    for i=1 to k do            // initialize count array
        count [i] = 0
    endfor

    for i=1 to n do                    // calculate frequency for each list value
        count [ L[i] ] = count [ L[i] ] + 1  (*)
    endfor

    total = 1
    for i=1 to k do                              // calculate the starting index for each value
        temp = count [i]
        count [i] = total   (*)
        total = total + temp
    endfor

    for i=1 to n do                    // copy the elements to output array
        Out [count [ L[i] ] ] = L[i]   (*)
        count [ L[i] ] = count [ L[i] ] + 1
    endfor
end
```

Complexity analysis:
We can take the assignments marked with (*) as the basic operation. So, the complexity is $f(n) = 2n+k \in \Theta(n+k)$

This algorithm is efficient if k is not very large. For instance, when $k<n$, this is a linear sorting algorithm. However, for instance if $k \approx n^2$, then it is a quadratic algorithm.