Tutorial 3

Question 1:

Find the computational complexity of the following piece of code using Big-oh notation:

```
for (int i = 1; i < n; i *= 2) {
 for (int j = n; j > 0; j /=2) {
  for (int k = j; k < n; k += 2) {
   sum += (i + j * k);
  } } }
```

Outer Loop: i*=2 is O(logn) because i*=2 is growing exponentially by double n is currently is. Thus its growing exponentially.To reach n, i need to double about log2(n)

Middle loop: since its j/= 2 is shrinking exponationally, due to it be half in each iteration.Thus will run in log2n times till it reaches 0

Inner loop:since k is +=2, it only covers half the increments. So while in all actuallity it can be n/2, we ignore constant thus making it O(n/2) = O(n)

Putting all together: O(logn) * O(logn) * O(n) = O(n log^2 n)

Question 2:

Write a recursive function GCD(n,m) that returns the greatest common divisor of two integer n and m according to the following definition (recurrence relation):

GCD(n,m) = {
 m, if m <= n and n mod m = 0 {
  GCD(m,n), if n < m {
   GCD(m, n mod m), otherwise

Example:
Enter the first number: 54
Enter the second number: 24
The GCD of 54 and 24 is 6

Question 3:

Use the master method to give tight asymptotic bounds for the following recurrences (if the master method cannot be applied give your argument):

Master algo: T(n) = aT(n/b) + O(n^d)
a = number of subproblem the algo splits the problem into
n/b = size of each subproblem(dividing the original problme of size n)
O(n^d) = time taken to divide probelm and combine the result after solving the subproblem

(a) $T(n) = 4T(n/2) + n$. O(n^2)

(b) $T(n) = 4T(n/2) + n^3$. O(n^3)

case 1: a > b^d = use n^(logb*a)
case 2: a = b^d = use n^dlogn
case 3: a <b^d = use n^d

Question 4:

The following is the running time of a recursion merge sort algorithm:

$$T(n) = 2T(n/2) + O(n)$$

Using the substitution method, proof that the time complexity of this algorithm is O(n lg n). Verify your answer with the tree method and the master method.