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);
    } }</pre>
```

Answer:

- O(n(lg(n))²)
- For the i loop, 1,2,4,8,16,32,...
 - o If n=1, log₂1=0 times
 - \circ If n=4, $\log_2 4=2$ times => 1,2
 - o If n=8, $log_28=3$ times => 1,2,4
 - \circ If n=16, $\log_2 16=4$ times => 1,2,4,8
 - o If n=32, $log_232=5$ times => 1,2,4,8,16
 - o Thus i loop will run log₂n times
- For the j loop, it is the same as i loop, it is just the reverse 32,16,8,8,4,2,1
 - o Thus, it is also it is also log₂n times
- For the k loop,
 - \circ The smallest possible j=1, for (int j = n; j > 0; j /=2)
 - The largest number of iterations for k is:
 - for (int k = j; k < n; k += 2)</p>
 - when k=j=2 => 2,4,6,8,10,....
 - Let m = number of iterations
 - 2m < n</p>
 - m < n/2</p>
 - thus, max number of iteration is n/2 = O(n)
 - The smallest number of iterations for k is:
 - for (int j = n; j > 0; j /=2) {
 - for (int k = j; k < n; k += 2) {</p>
 - K=j=n, there is no k iteration
 - The next smallest number of iterations for k is:
 - K=j=n/2, => n/2 + 2, n/2 + 4, n/2 + 6
 - Let m = number of iterations
 - n/2+2m < n</p>
 - m < (n-n/2)/2 = n/4</p>
 - thus, min number of iteration is n/4 = O(n)

- Hence the inner loop is Big-O of n.
- The i and j loop runs lg(n) times, and the kth loop runs minimally n/4 times. Hence it is n(lg(n))^2

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

Answer:
// assume a < b
int gcd(a,b)
{
  if (b != 0) return gcd(b, a%b);
  else return a;
}</pre>
```

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):

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

Answer:

For all these questions we have, a = 4 and b = 2. Thus $n^{\log ba} = n^{\lg 4} = n^2$.

- (a) For this recurrence, in which f(n) = n, we have $f(n) = O(n^{\log_b a \epsilon})$, where $0 < \epsilon \le 1$. Hence, we can apply case 1 of the master method. The solution for this recurrence is $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$.
- (b) For this recurrence, in which $f(n) = n^3$, we have $f(n) = \Omega(n^{\log_b a + \epsilon})$, where $0 < \epsilon \le 1$ and the regularity condition holds for f(n), since

$$af(n/b) = 4f(n/2) = 4(\frac{n}{2})^3 = \frac{n^3}{2} \le c \cdot f(n)$$

for some constant $1/2 \le c < 1$ and all sufficiently large n. Therefore, the solution to this recurrence (by applying case 3 of the master method) is $T(n) = \Theta(f(n)) = \Theta(n^3)$.

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.

Answer:

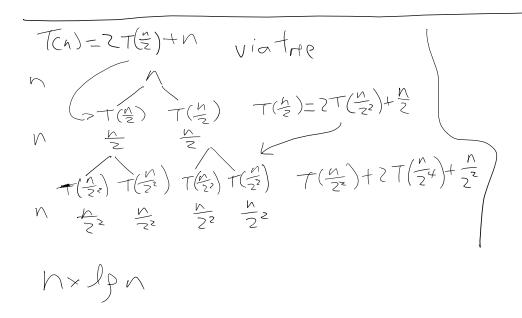
$$T(n) = 2T\left(\frac{n}{2}\right) + n$$
 $T(n) = O(nlgn)$

$$T(n) \le 2\left(c\frac{n}{2}lg\frac{n}{2}\right) + n = cn(lgn - lg2) + n$$
$$= cnlgn - cn + n = cnlgn - (c - 1)n$$

$$T(n) \le cnlgn - (c-1)n$$

 $\le cnlgn \ \ via \ substitution \ method$

Tree Method



Master Method:

$$T(n) = 2T(\frac{2}{2}) + h$$

 $\alpha = 2$, $b = 2$, $f(n) = h$
 $g(n) = \frac{1}{n} \log_2 2$
Since $f(n) = g(n)$
 $= n$
 $ase 2$
 $n \times lgn$