1) Analyze the time complexity of the following java codes and suggest a way to improve it.

```
Int sum = 0;

for( int i = 1; i <= n; i++){

    for(int j = 1; j <= i; j++){

            Sum++;

    }

}
```

Ans:

The provided Java code calculates the sum of integers from 1 to `n` in a nested loop. The outer loop runs from 1 to `n`, and for each iteration of the outer loop, the inner loop runs from 1 to the value of the outer loop variable. In the innermost part of the loop, the variable `sum` is incremented.

The time complexity of this code can be analyzed as follows:

- The outer loop runs `n` times.
- For each iteration of the outer loop, the inner loop runs `i` times, where `i` is the value of the outer loop variable at that iteration.

So, the total number of iterations of the innermost part of the loop can be expressed as:

1 + 2 + 3 + ... + n

This is an arithmetic series with a sum formula of (n (n + 1)) / 2.

Therefore, the time complexity of the given code is O(n^2) because it involves two nested loops.

To improve the time complexity, you can use mathematical insights to directly calculate the sum from 1 to `n` using the arithmetic series formula:

int sum = (n (n + 1)) / 2;

This would provide a much more efficient solution with a time complexity of O(1), as it directly computes the sum in constant time regardless of the value of `n`.

2) Find the value of T(2) for the recurrence relation T(n) = 3T(n-1)+ 12n, given that T(0) = 5.

Ans:

The given recurrence relation is T(n) = 3T(n-1) + 12n with an initial condition T(0) = 5.

An iterative approach to find T(2):

1. T(0) = 5 (Given)
2. T(1) = 3T(0) + 12(1) = 3 5 + 12 = 15 + 12 = 27
3. T(2) = 3T(1) + 12(2) = 3 27 + 24 = 81 + 24 = 105

So, the value of T(2) for the given recurrence relation is 105.

3) Give a recurrence relation, solve it using the substitution method.

Relation: T(n) = T(n-1) + c

Ans:

The recurrence relation T(n) = T(n-1) + c using the substitution method.

Recurrence Relation: T(n) = T(n-1) + c

Base Case: T(1) = c

Substitution:
Let's substitute T(n-1) in terms of T(n-2):
T(n) = T(n-2) + c + c

Substitute T(n-2) in terms of T(n-3):
T(n) = T(n-3) + c + c + c

Continuing this pattern, we substitute $T(n-k)$ in terms of $T(n-k-1)$ k times:
$T(n) = T(n-k) + k \ c$

When we reach $k = n - 1$, we have:
$T(n) = T(1) + (n - 1) \ c$
$T(n) = c + (n - 1) \ c$
$T(n) = n \ c$

The solution to the recurrence relation $T(n) = T(n-1) + c$ using the substitution method is:
$T(n) = n \ c$

4) Given a recurrence relation. T(n) = 16T(n/4) + n2 logn. Find the complexity of this relation using the master theorem.

Ans:

The given recurrence relation is:
$T(n) = 16T(n/4) + n^2 \ \log(n)$

To analyze the complexity of this relation using the Master Theorem, we'll compare it to the general form of the Master Theorem:

$T(n) = aT(n/b) + f(n)$

In the given relation:
- $a = 16$
- $b = 4$
- $f(n) = n^2 \ \log(n)$

Comparing this to the Master Theorem, we need to find the value of "c" in the expression $f(n) = \Theta(n^c \ \log^k(n))$, where $k \geq 0$.

In our case, $f(n) = n^2 \ \log(n)$, so $c = 2$ and $k = 1$.

Now we can apply the Master Theorem:

1. If $f(n) = \Theta(n^c \ \log^k(n))$ where $c > \log_b(a)$, then $T(n) = \Theta(f(n))$.
2. If $f(n) = \Theta(n^c \ \log^k(n))$ where $c = \log_b(a)$, then $T(n) = \Theta(n^c \ \log^{(k+1)}(n))$.
3. If $f(n) = \Theta(n^c \ \log^k(n))$ where $c < \log_b(a)$, and if $a \ f(n/b) \leq k \ f(n)$, then $T(n) = \Theta(n^{\log_b(a)})$.

In our case, a = 16, b = 4, c = 2, and k = 1. Let's calculate c and log_b(a):

c = 2
log_b(a) = log_4(16) = 2

Since c = log_b(a), we are in case 2 of the Master Theorem.

According to case 2, the complexity of the recurrence relation is:
T(n) = Θ(n^c log^(k+1)(n))
T(n) = Θ(n^2 log^2(n))

So, the complexity of the given recurrence relation T(n) = 16T(n/4) + n^2 log(n) using the Master Theorem is Θ(n^2 log^2(n)).

5) Solve the following recurrence relation using recursion tree method T(n) = 2T(n/2) + n

Ans:

To solve the given recurrence relation T(n) = 2T(n/2) + n using the recursion tree method, we'll break down the recursive calls into a tree structure and analyze the pattern.

Recurrence Relation: T(n) = 2T(n/2) + n

Recursion Tree:
At each level of recursion, the problem is divided into two subproblems of size n/2, and the cost of each level is n. The tree will have $\log_2 n$ levels (since we're dividing the problem size by 2 at each level).

```
          T(n)
         /   \
      T(n/2)  T(n/2)
      /   \    /   \
T(n/4) T(n/4) T(n/4) T(n/4)
   ...    ...    ...    ...
 leaf   leaf   leaf   leaf
```

Analysis:

Each level of the recursion tree contributes a total cost of n (due to the "n" term in the recurrence relation). Since there are $\log_2 n$ levels, the total cost is $\log_2 n$ n.

Total Cost:
T(n) = cost per level number of levels
$T(n) = n \ \log_2 n$

The solution to the given recurrence relation $T(n) = 2T(n/2) + n$ using the recursion tree method is $T(n) = n \ \log_2 n$.

6) T(n) = 2T(n/2) + k, solve using recurrence tree method.

Ans:

To solve the recurrence relation $T(n) = 2T(n/2) + k$ using the recursion tree method, we'll build a recursion tree and analyze the pattern.

Recurrence Relation: $T(n) = 2T(n/2) + k$

Recursion Tree:
At each level of recursion, the problem is divided into two subproblems of size n/2, and the cost of each level is k. The tree will have $\log_2 n$ levels (since we're dividing the problem size by 2 at each level).

```
         T(n)
        /    \
     T(n/2)   T(n/2)
     /  \      /  \
  T(n/4) T(n/4) T(n/4) T(n/4)
     ...   ...    ...   ...
   leaf   leaf   leaf   leaf
```

Analysis:
Each level of the recursion tree contributes a total cost of k (due to the "k" term in the recurrence relation). Since there are $\log_2 n$ levels, the total cost is $\log_2 n$ k.

Total Cost:
T(n) = cost per level number of levels

$T(n) = k \log_2 n$

The solution to the given recurrence relation $T(n) = 2T(n/2) + k$ using the recursion tree method is $T(n) = k \log_2 n$.