Question 1. Analyze the time complexity of the following Java code and suggest a way to improve it:

Ans. int sum = 0;

  for(int i = 1; i <= n; i++) {

  for(int j = 1; j <= i; j++) {

  sum++;  }  }

ans. Time complexity= O(n^2)

One way to improve the time complexity of this code is to use a mathematical formula to find the sum instead of using nested loops

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

Ans. given T(n) = 3T(n-1) + 12n

Substituting the values in the relation:

T(1) = 3T(0) + 12

=> T(1) = 15 + 12 = 27

T(2) = 3T(1) + 12 \* 2

=>T(2) = 3 \* 27 + 24 = 81 + 24

Hence T(2) = 105.

  Q3: Given a recurrence relation, solve it using a substitution method.  Relation : T(n)= T(n - 1) +c

Let the solution be T(n) = O(n), now let’s prove this using the induction method.

For that to happen T(n) <= cn where c is some constant.

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

T(n - 1) = T(n - 2) + c

T(n - 2) = T(n - 3) + c

|

|

T(2) = T(1) + c

—------------------------ Adding all above equations

T(n) = T(1) + cn

Let us assume T(1) to be a constant value.

T(n) = k + cn

Therefore, T(n) <= cn

Hence we can conclude T(n) = O(n).

Question 4: Given a recurrence relation:  T(n) = 16T(n/4) + n2logn  Find the time complexity of this relation using the master theorem.

Sol. From the given recurrence relation we can obtain the value of different parameters such as a,b ,p and k.

The relation T(n)=16T(n/4)+n2logn

Here, a=16,

B=4

K=2

P=1

Bk=42=16

Here a=bk

Also p>-1

Hence T(n) =Thitha(nlogab\*logp+ln)

Therefore T(n) = Thitha(log164\*log1+ln)=Thitha(n1/2log2n).

Question 6. T(n) = 2T(n/2) + K, Solve using Recurrence tree method

As we know that (n/2^k) =1

n = 2^K

Taking log both sides

log(n) = log(2^k)

log(n) = klog(2)

k = log(n)/log(2)

k = log2(n)

Height of tree is log(n) base 2

Step3. Calculate cost at each level

Level 0 = K

Level 1 = K + K = 2K

Level 2 = K + K + K + K= 4K and so on…

Step 4. Calculate number of nodes at each level

Level 0 = 2^0 = 1

Level 1 = 2^1 = 2

Level 2 = 2^2 = 4 and so on…

Step 5. Calculating final cost:

The total cost can be written as,

Total Cost = Cost of all levels except last level + Cost of last level

Total Cost = Cost for level-0 + Cost for level-1 + Cost for level-2 + .... + Cost for level-log(n) + Cost for last level

The cost of the last level is calculated separately because it is the base case and no merging is done at the last level so, the cost to solve a single problem at this level is some constant value. Let's take it as O(1)

Let's put the values into the formulae,

T(n) = K + 2\*K + 4\*K + .... + log(n)` times + `O(1) \* n

T(n) = K(1 + 2 + 4 + .... + log(n) times)` + `O(n)

T(n) = K(2^0 + 2^1 + 2^2 + ....+ log(n) times + O(n)

In the GP formed above, a = 1 and r = 2, after solving this we get, T(n) = K \* (1 / (2 - 1)) + O(n)

T(n) = K + O(n)

T(n) = O(n)