Time and space complexity Assignment

Question 1. Analyze the time complexity of the following Java code 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++;
   }
}
```

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

Question 3: Given a recurrence relation, solve it using a substitution method.
Relation : T(n) = T(n - 1) + c

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

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

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

Ans 1.— Time Complexity of the following Java Code is - $O(n^2)$

- We can reduce the time complexity of it to — $O(1)$

```
public class SumCalculator {
public static void main(String[] args) {
      int n = 10;
      int sum = calculateSum(n);
 System.out.println("The sum is: " + sum);
}
public static int calculateSum(int n) {
   return (n * (n + 1)) / 2;
}

}
```

Ans 2 — $T(2)$ $T(0)=5$

$T(n) = 3T(n-1)+12n$

$T(1) = 3T(1-1)+12\times1$

$= 3T(0) + 12$
$= 3 \times 5 + 12$
$= 27.$

$T(2) = 3T(2-1) + 12\times2$
$= 3T(1) + 24$
$= 3\times27 + 24$
$= 81 + 24$
$= 105$

$\boxed{T(2) = 105}$

Ans 3 — Relation : $T(n) = T(n-1) + c$

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

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

$T(n-2) = T(n-2-1) + c$
$T(n-2) = T(n-3) + c$

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

K - step
$T(n) = T(n-k) + 3k$

Base case:—
$n-k = 0$ means $k = n$

$T(n) = T(0) + nc$

$\boxed{T(n) = T(0)+nc}$

Ans 4 — $T(n) = 16T(n/4) + n^2\log n$

$a = 16$, $b = 4$, $k = 2$

So we have chart according to it

this is first case

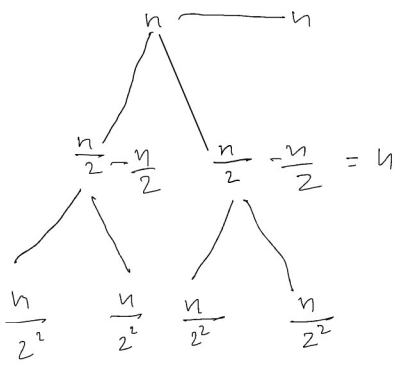
Time Complexity of it will be:- $O(n^{\log_4 2})$
$= O(n^{\log_2})$

Ans 5:— $T(n) = 2T(\frac{n}{2}) + n$

- $T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + n$

Eg:- $T(n) = 2T(n/2) + n^2$ — n^2 : Cost

- $T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + n$



$2^i \times \frac{n}{2^i} = n$

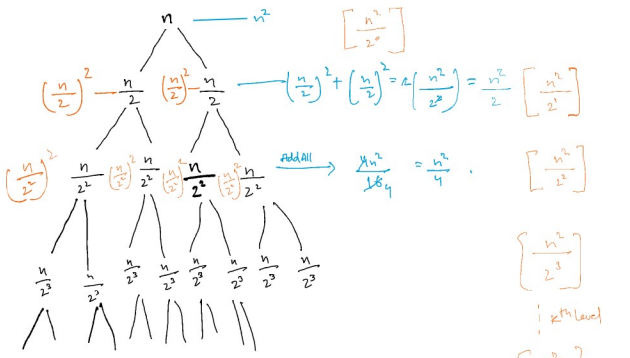
$\frac{n}{2^i} = 1$, i.e., $n = 2^i$.

Total cost = $n \times \log_2 n$

Time Complexity = $T(n) = O(n \log n)$

Eg:- $T(n) = 2T(n/2) + n^2$ (cost)

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



$$\frac{n^2}{2^0} + \frac{n^2}{2^1} + \frac{n^2}{2^2} + \frac{n^2}{2^3} + \dots + \frac{n^2}{2^k}$$

Base Case

$\frac{n^2}{2^k} = 1$

Ans 6- $T(n) = 2T\left(\frac{n}{2}\right) + K$

Level-0 $T(n) = 2T\left(\frac{n}{2}\right) + K$ Cost = K

Level-1 $T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + K$ Cost = 2K

Level-2 $T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + K$ Cost = 4K

Level-i ⟶ Cost $2^i K$

$\frac{n}{2^i} = 1 \Rightarrow n = 2^i$

$i = \log_2 n$

Total Cost = $K \times \left(1 + 2 + 4 + \dots + 2^{\log_2 n}\right)$

G.P. – $\frac{a(r^n - 1)}{r - 1}$ Here $r = 2$.

$= K \times \left(2^{\log_2 n} - 1\right) \approx K \times n$

Time Complexity = $T(n) = O(n)$