"Time & Space Complexity"

Q1 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++;
    }
}
```

Sol:- The time complexity of this code is $O(n^2)$ as it uses nested loops, where the outer loop runs n times and the inner loop runs i times where i varies from 1 to n.
The total number of operations performed can be calculated by summing the total number of operations in each iteration of the outer loop. The inner loop will run i times on the ith-iteration of outer loop. so the number of operations is $(1+2+3+\_\_+n)$ which is $n(n+1)/2$, which is $O(n^2)$.

Date ___/___/___

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.

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

sol:- Given, $T(n) = 3T(n-1) + 12n$

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

$\Rightarrow 3T(0) + 12$

put value of $T(0)$

$\Rightarrow 3 \times 5 + 12$

$T(1) \Rightarrow 27$

$T(2) = 3T(n-1) + 12n$

$= 3T(2-1) + 12 \times 2$

$\Rightarrow 3T(1) + 24$

put value of $T(1)$

$\Rightarrow 3 \times 27 + 24$

$\Rightarrow 81 + 24$

Hence, $T(2) = 105$.

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

sol:- Let the solution be $T(n) = O(n)$, now let's prove this using induction method.

for that to happen $T(n) <= cn$, where c is constant

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

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

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

$\vdots$

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

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 $T(n) = O(n)$.

Q.4 Given a recurrence relation:

$T(n) = 16T(n/4) + n^2 \log n$

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

$T(n) = 16T(n/4) + n^2 \log n$

Here, $a = 16$, $b = 4$, $k = 2$, $p = 1$

$b^k = b^2 = 4^2 = 16$

Here, $a = b^k$, Also $P > -1$

$T(n) = O(n^{\log_b a} \log^{P+1} n)$

$T(n) = O(n^{\log_4 16} \log^2 n)$

$= O(n/2 \log_2 n)$

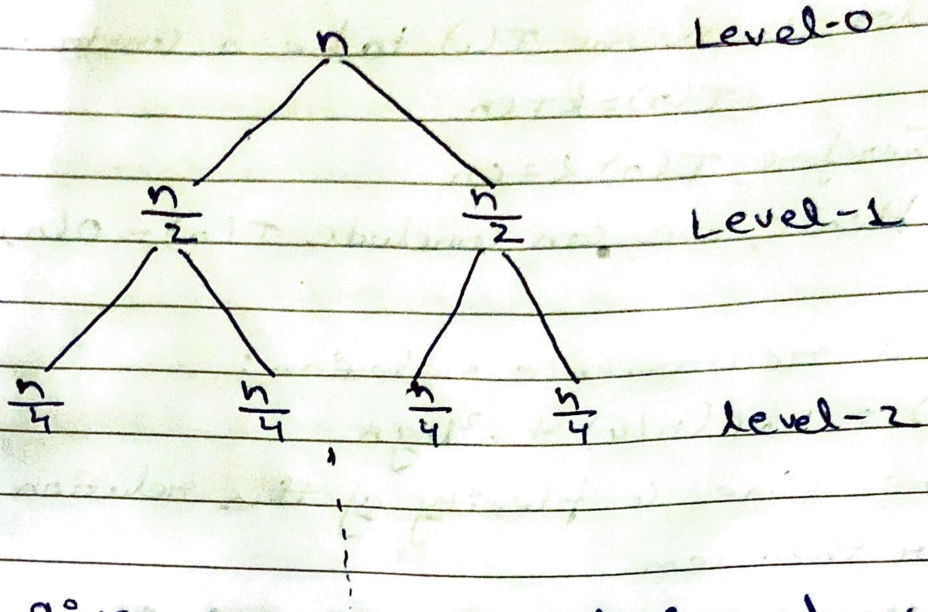
Q.5 solve the following recurrence relation using recursion tree method $T(n) = 2T(n/2) + n$

Sol:- The given recurrence relation shows:-

A problem of size n will get divided into 2 subproblems of size n/2.

Then, each sub-problem of size n/2 will be divided into 2 subproblems of size n/4 and so on.

At the bottom most layer, the size of sub-problems will reduce to 1.



The given recurrence relation shows:-
The cost of dividing a problem of size n into its 2 sub-problems and then combining its solution is n.
The cost of dividing a problem of size $n/2$ into its 2 sub-problems and then combining its solution is $n/2$ and so on.
This is illustrated through following recursion tree where each node represents the cost of corresponding sub-problem.

Step 2:- Determine cost of each level:-

Cost of level 0 = n
Cost of level 1 = $n/2 + n/2 = n$
Cost of level 2 = $n/4 + n/4 + n/4 + n/4 = n$ and so on.

Step 3:- Determine total number of levels in recursion tree.

Size of sub problem at level 0 = $n/2^0$
Size of sub problem at level 1 = $n/2^1$
Size of sub problem at level 2 = $n/2^2$
Continuing in similar manner, we have
size of sub problem at level i = $n/2^i$
suppose at level-x (last level), size of sub problem becomes 1. Then $n/2^x = 1$.

$2^x = n$

Taking log on both sides (with base 2) we get,

$x \log 2 = \log n$

$x = \log_2 n$

Total number of levels in recursion tree $= \log_2 n + 1$

step 4:- Determine number of nodes in last level

level 0 has 2^0 nodes that is 1 node

level 1 has 2^1 nodes that is 2 nodes

level 2 has 2^2 nodes that is 4 nodes

Continuing in similar manner, we have level-$\log_2 n$ has $2^{\log_2 n}$ nodes i.e. n nodes

step 5:- Determine cost of last level

cost of last level $= n \times T(1) = \theta(n)$

step 6:- Add costs of all levels of the recursion tree and simplify the expression so obtained in terms of asymptotic notation

$$T(n) = \underbrace{\{n + n + n + \_\_\_\}}_{\log_2 n \text{ levels}} + \theta(n)$$

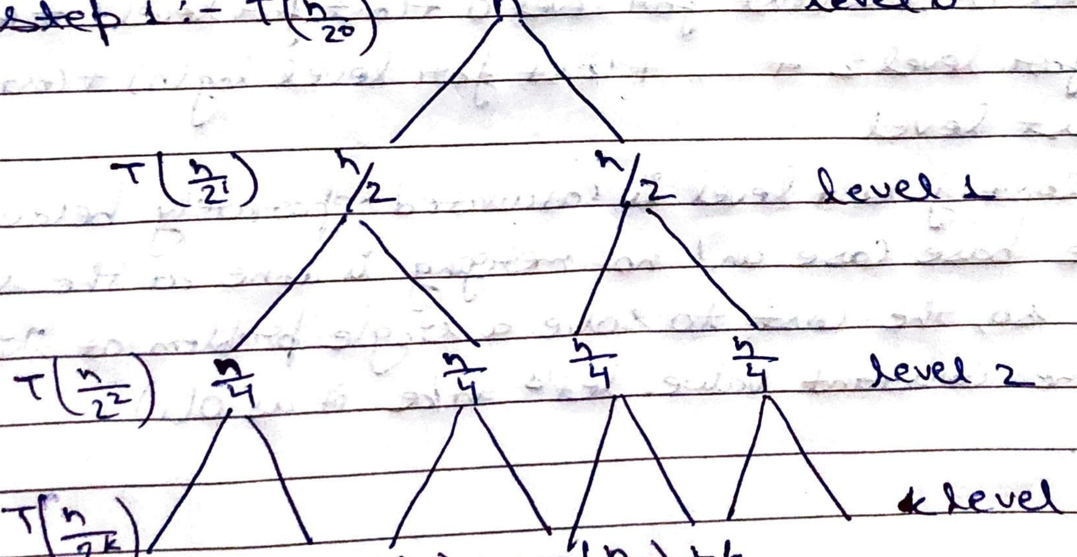
$$= n \times \log_2 n + \theta(n)$$

$$= n \log_2 n + \theta(n)$$

$$= \theta(n \log_2 n)$$

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

Sol- step 1:- $T\left(\frac{n}{2^0}\right)$



n
level 0
$T\left(\frac{n}{2^1}\right)$
n/2
n/2
level 1
$T\left(\frac{n}{2^2}\right)$
n/4
n/4
n/4
n/4
level 2
$T\left(\frac{n}{2^k}\right)$
k level

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

Date ___/___/___

step2:- Calculating height of tree

As we know that $\left(\frac{n}{2^k}\right) = 1$

$n = 2^k$

~~Ta~~king log both side

$\log(n) = \log 2^k$

$\log(n) = k \log 2$

$k = \frac{\log n}{\log 2}$

$k = \log_2 n$

Height of tree is log(n) base 2

step 3:- Calculating cost at each level

level 0 = k

level 1 = k+k = 2k

level 2 = k+k+k+k = 4k and so on.

step4:- 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 at 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 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)

Saathi

Date ___ / ___ / ___

let's put the values into the formulae,

$T(n) = k + 2 \times k + 4 \times k + \_\_ + \log(n) \text{ times} + O(1) \times n$

$T(n) = k(1 + 2 + 4 + \_\_ + \log(n) \text{ times}) + O(n)$

$T(n) = k(2^0 + 2^1 + 2^2 + \_\_ + \log(n) \text{ times}) + O(n)$

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

$T(n) = k + O(n)$

$T(n) = O(n)$