Time & Space Complexity Assignment Questions

Question 1. Analyse 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++;
}     }
```

Answer:

Time Complexity - > O(n^2)

Improvement:

The original code can be optimized to achieve a better time complexity. Instead of using nested loops to calculate the sum, we can use a formula to directly compute the sum of the first 'n' natural numbers:

By replacing the nested loops with a direct formula, the improved code would be:

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

Using this formula, we can calculate the sum without the need for nested loops, resulting in a time complexity of O(1), which is much more efficient than the original O(n^2) complexity.

Question 2. Find the value of T (2) for the recurrence relation T(n) = 3T(n-1) + 12n .

Given that T (0) = 5.

Answer:

To find the value of T (2) for the recurrence relation T (n) = 3 T (n-1) + 12 n, we'll use the given initial condition T (0) = 5 and work our way up step by step.

T (0) = 5 (Given initial Condition)

Now, Let's find T (1) using the recurrence relation.

T (1) = 3 T (1-1) + 12 (1) = 3 T (0) + 12 = 15 + 12 = 27

Next, Let's find T (2) using the recurrence relation.

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

So, the value of T (2) for the given recurrence relation with the initial condition T (0) = 5 is 105.

Question 3. Given a recurrence relation, solve it using a substitution method

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

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

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

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

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

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

K times…

$T(n) = T(n-k) + kc$

$T(1) = 1$

$n-k = 1$

$k = n - 1$

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

$T(1) + nc - c$

$1 + c(n-1)$

Time Complexity: $O(n)$

Question 4: Give a recurrence relation

$T(n) = 16T(n/4) + n2 \log n$

Find the time complexity of this relation using the master theorem

$T(n) = aT(n/b) + \text{theta}(n^{\wedge} k \log p\ n)$

n= size of the problem

a = number of subproblems

n/b = size of subproblem

a>=1, b>1, k>=0

p = real number

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

a=16

b= 4

k=1

p=0

a=16   b ^k = 4

p>=0

T(n) = O (n ^k log p n)

T(n) = O (n log (n))

Question 5. Solve the following recurrence relation using recursion tree method

$$T(n) = 2T(n/2) + n$$

Answer

To solve the recurrence relation T(n) = 2T(n/2) + n using the recursion tree method, we'll build a tree to visualize the recursive calls and then find a pattern to derive the time complexity.

Step 1: Build the Recursion Tree

Let's construct the recursion tree for the given recurrence relation:

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

/ / /

T(1) T(1) T(1)

The recursion tree has a depth of $\log_2(n)$ since we repeatedly divide n by 2 in each level until it reaches 1.

Step 2: Calculate the Cost at Each Level

At each level, the cost of the work done at that level is n, as it is the only operation performed before splitting into two subproblems of size n/2.

Level 0: n

Level 1: n/2 (Two subproblems of size n/2, each with cost n/2)

Level 2: n/4 (Four subproblems of size n/4, each with cost n/4)

...

Generalizing for Level i: n/(2^i)

Step 3: Summing Up the Costs

Now, let's sum up the costs at each level:

Total Cost = n + n/2 + n/4 + n/8 + ... + 1

This is a geometric series with a common ratio of 1/2 (each term is half of the previous one) and a first term of n.

Step 4: Solve the Geometric Series

The sum of a geometric series with a first term 'a' and a common ratio 'r' up to 'k' terms is given by the formula:

Sum = a * (1 - r^k) / (1 - r)

In our case, a = n, r = 1/2, and k = $\log_2(n)$ (since there are $\log_2(n)$ levels in the recursion tree).

Total Cost = n * (1 - $(1/2)^{\log_2(n)}$) / (1 - 1/2)

Now, let's simplify further:

Total Cost = n * (1 - 1/n) / (1/2)

= 2n * (1 - 1/n)

= 2n - 2

Step 5: Time Complexity

Finally, we have derived the time complexity of the recurrence relation using the recursion tree method:

T(n) = 2n - 2

Therefore, the time complexity of the given recurrence relation T(n) = 2T(n/2) + n using the recursion tree method is $\Theta(n)$.

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

Answer

To solve the recurrence relation T(n) = 2T(n/2) + k using the recursion tree method, we'll construct the recursion tree and analyze the cost at each level.

Step 1: Build the Recursion Tree

Let's construct the recursion tree for the given recurrence relation:

```
     T(n)
     /  \
  T(n/2)  T(n/2)
  /  \   /  \
```

```
T(n/4) T(n/4) T(n/4) T(n/4)
   ...    ...    ...
   /      /      /
T(1)    T(1)    T(1)
```

The recursion tree has a depth of $\log_2(n)$ since we repeatedly divide n by 2 in each level until it reaches the base case T(1).

Step 2: Calculate the Cost at Each Level

At each level, the cost of the work done at that level is k, as it is the constant term added in each recursive call.

Level 0: k

Level 1: k (Two subproblems of size n/2, each with cost k)

Level 2: k (Four subproblems of size n/4, each with cost k)

...

Generalizing for Level i: k

Step 3: Summing Up the Costs

Now, let's sum up the costs at each level:

Total Cost = k + k + k + ... + k ($\log_2(n)$ times)

Total Cost = k * $\log_2(n)$

Step 4: Time Complexity

Finally, we have derived the time complexity of the recurrence relation using the recursion tree method:

$T(n) = k * \log_2(n)$

Therefore, the time complexity of the given recurrence relation $T(n) = 2T(n/2) + k$ using the recursion tree method is $\Theta(\log_2(n))$.