Time and Space complexity

Assignment Solutions

PW SKILLS

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++;
   }
}
```

Solution :

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 i-th iteration of the outer loop. So the number of operations is (1+2+3+...+n) which is n(n+1)/2, which is 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.

Solution : 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.

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

Solution: 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.

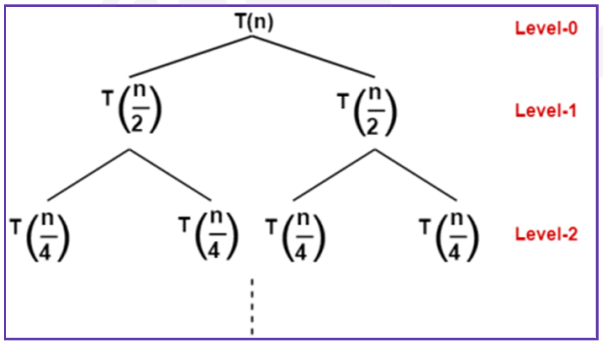
Solution: From the given recurrence relation we can obtain the value of different parameters such as a, b, p, and k.
The relation : T(n) = 16T(n/4) + n2logn
Here, a = 16
b = 4
k = 2
p = 1
bk = 42 = 16
Here a = bk
Also p > -1
Hence T(n) = θ(nlogab * logp+1n)
Therefore T(n) = θ(nlog164 * log1+1n) = θ(n1/2log2n)

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

Solution : 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.

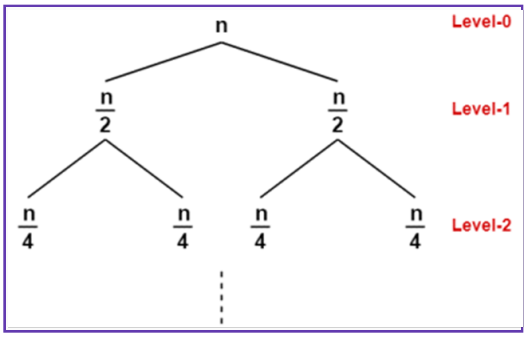
This is illustrated through following recursion tree-



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 the corresponding sub-problem



Step-02:
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-03:
Determine total number of levels in the recursion tree-

- Size of sub-problem at level-0 = n/20
- Size of sub-problem at level-1 = n/21
- Size of sub-problem at level-2 = n/22

Continuing in similar manner, we have
Size of sub-problem at level-i = n/2i
Suppose at level-x (last level), size of the sub-problem becomes 1. Then n / 2x = 1
2x = n
Taking log on both sides(with base 2), we get
xlog2 = logn
x = log2n

∴ Total number of levels in the recursion tree = log2n + 1 Step-04:

Determine number of nodes in the last level-

- Level-0 has 20 nodes i.e. 1 node
- Level-1 has 21 nodes i.e. 2 nodes
- Level-2 has 22 nodes i.e. 4 nodes

Continuing in similar manner, we have Level-log2n has 2log2n nodes i.e. n nodes

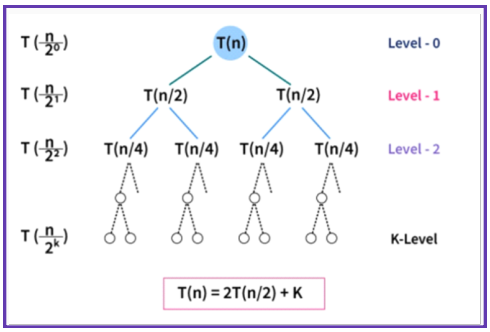
Step-05:
Determine cost of last level
Cost of last level = n x T(1) = θ(n)

Step-06:
Add costs of all the levels of the recursion tree and simplify the expression so obtained in terms of asymptotic notation-

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

= n x log2n + θ (n)
= nlog2n + θ (n)
= θ (nlog2n)

Question 6. T(n) = 2T(n/2) + K, Solve using Recurrence tree method.
Step1. Drawing Recursion Tree



@pw Team please redesign the image

Step2. Calculating height of tree
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)