Explanation:
Here each call add a level to the stack :

1. addUpto(4)
2. -> addUpto(3)
3. -> addUpto(2)
4. -> addUpto(1)
5. -> addUpto(0)

Each of these calls is added to the call stack and takes up actual memory.
So it takes O(n) space.

Time complexity of recurrence relations

Pre-requisites

- Recursion
- What is time complexity?

List of concepts involved

- Substitution method
- Recurrence tree method

Time complexity of recurrence relations:

A recurrence relation is a functional relation between the independent variable x, dependent variable f(x) and the differences of various order of f (x).
For example, we studied the fibonacci series where we have seen that any ith term is a function of previous two terms, i.e. (i-1)th term and (i-2)th term.

For any independent variable 'n', the corresponding fibonacci value is obtained by:
f(n) = f(n - 1) + f(n - 2) .....(1)
The equation (1) is known as recurrence relation.
Similarly, many algorithms are recursive. When we analyze them, we get a recurrence relation for time complexity. In order to find time complexity of such relations we will be discussing two methods in this class in great detail.

Substitution method:

The Substitution Method Consists of two main steps:
1. Guess the Solution.
2. Use principles of mathematical induction to find the boundary condition and show that our guess is correct.

Let's go with various examples to discuss more on the Substitution method, how it works and how to find time complexity using this method.

Question 1: Find the time complexity of the recurrence relation given by:

T(n) = 2T(n/2) + 4n

Solution: As discussed in the steps firstly we need to make a random guess towards our solution.

Let the solution be T(n) = O(nlogn).
The second step is to use mathematical induction to prove our guess to be correct.
So we have to prove that = T(n) <= c*n*logn, where c is a constant. We can assume that it is true for values smaller than n.
T(n) = 2T(n/2) + 4n
<= 2cn/2Log(n/2) + 4n
= cnLogn – cnLog2 + 4n
= cnLogn – cn + 4n
Ultimately T(n) <= cnLogn
Hence we can conclude the overall worst case time complexity to be O(nlogn).

Question 2: Determine the recurrence relation for the following series 1,7,31,127,511.
Solution : Let's analyze the sequence,
7 - 1 = 6
31 - 7 = 24 which can also be written as 4 * 6
127 - 31 = 96 which can be written as 4 * 24
511 - 127 = 372 which can be written as 4 *96
Now 2nd term 7 can be written as 4 * 1 + 3
Then 31 can be written as 4 * 7 + 3
. . . .and so on.
Hence the recurrence relation so forming is T(n) = 4 * (n - 1) + 3.

Recurrence tree method:

Recursion Tree is another method for solving the recurrence relations.
A recursion tree is a tree where each node represents the cost of a certain recursive sub-problem.
We sum up the values in each node to get the cost of the entire algorithm.

Steps to Solve Recurrence Relations Using Recursion Tree Method-

Step-01:
- Draw a recursion tree based on the given recurrence relation.

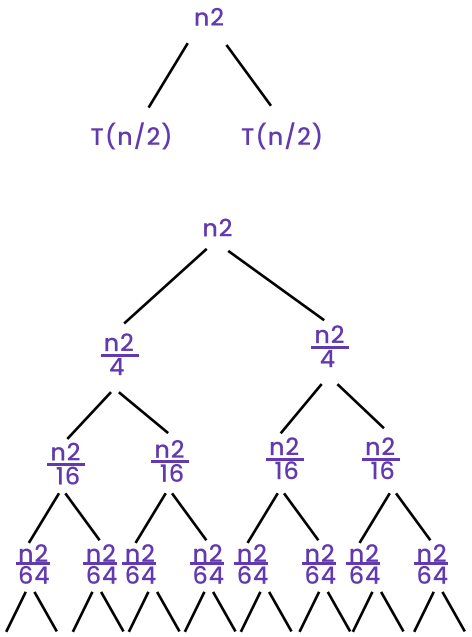
Step-02:
- Determine-
  - Cost of each level
  - Total number of levels in the recursion tree
  - Number of nodes in the last level
  - Cost of the last level

Step-03:
- Add the cost of all the levels of the recursion tree and simplify the expression so obtained in terms of asymptotic notation.

Question 1: Solve the following recurrence relation using recursion tree method-

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

Solution: The recurrence tree can be formed as:



@designer team please design these images as per PW standards.

T(n) = sum of time at each level
T(n) = n^2 + n^2/2 + n^2/4 + ... Log n (to the base 2) times.

Logn is the length/height of the recurrence tree so formed.

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

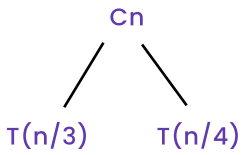
T(n) is an infinite GP with common ratio (r) = ½
Sum of infinite GP = a/(1-r) where a = first term and r = common ratio

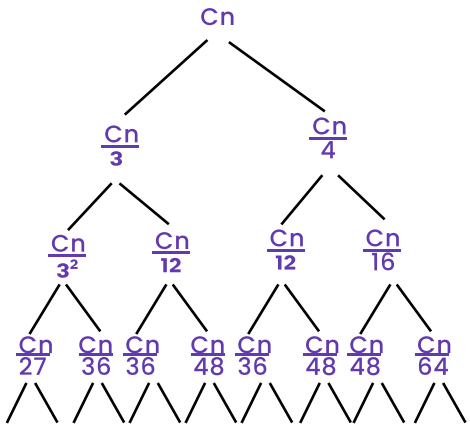
T(n) = n^2 / (1 - 1/2) = 2*n^2

Hence we can conclude the time to be T(n) = O(n^2) [considering upper bound for worst case complexity].

Question 2: Find the time complexity of the following recurrence relation using recurrence tree method.
T(n) = T(n/3) + T(n/4) + kn where k = constant

Solution:





To get total time sum values at each level of the recurrence tree:
T(n) = cn + cn/3 + cn/4 + cn/9 + cn/12 ..... Logn time.
The height of the tree is logn which is proved already in above examples as well.
T(n) = cn[1 + ⅓ + ¼ + 1/9 + 1/12+ ...]
T(n) = cn[1 + 7/12 + (7/12)^2 + . . .]
This is again sum of infinite GP with common ratio = 7/12
And first term = 1

Sum of infinite GP = a/(1 - r)
T(n) = cn[1/(1 - 7/12)] = cn*12/5

Therefore T(n) = O(n) . . . [considering upper bounds, ignoring constants]

Master Theorem

Pre-requisites:

- Recursion and recurrence relation
- What is time complexity?

List of concepts involved:

- Master theorem

Introduction:

The most widely used method to analyze or compare various algorithms is by computing their time complexities. As the algorithm gets complex, the time complexity function calculation also complexifies. Recursive functions call themselves in their body. It's more difficult if we start calculating its time complexity function by other commonly used simpler methods like substitution methods or recurrence tree methods. Master's theorem method is the most useful and easy method to compute the time complexity function of recurrence relations.

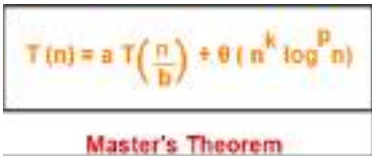
We can apply Master's Theorem for:

1. Dividing functions
2. Decreasing Functions

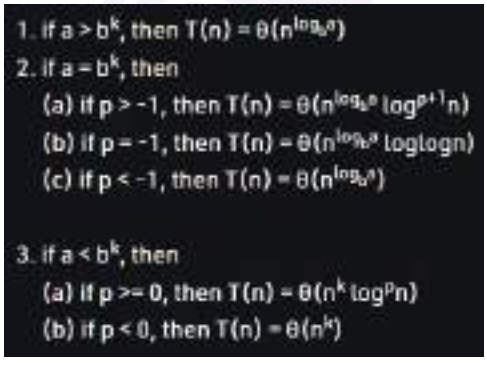
We will get into more detail what master theorem is and will see various examples to analyze the complexity of various functions.

Master theorem:

The master theorem is a method to calculate time complexities of recurrence relations of the following type:



where n = size of the problem
a = number of subproblems in the recursion and a >= 1
n/b = size of each subproblem
b > 1, k >= 0 and p is a real number.
Then,



Question 1: Given a recurrence relation:
T(n) = 3T(n/5) + 5n^2
Find the time complexity of this relation using the master theorem.

Solution:

From the given recurrence relation we can identify the variables a, b, p and k.
T(n) = 3T(n/5) + 5n^2

Here, a = 3 , b = 5 , k = 2 , p = 0
b^k = 5*5 = 25
a < b^k that corresponds to case 3.

Now since p >= 0
Therefore T(n) = θ(n^k * (logn)^p)
T(n) = θ(n^2) because (logn) ^ 0 = 1.

Question 2: Given a recurrence relation:
T(n) = 2T(2n/3) + 1
Find the time complexity of this relation using the master theorem.

Solution: Comparing the original equation of the master's theorem, the respective values of a ,b, p and k are 2 , 3/2, 0 , 0.

a > b^k because a = 2 and b^k = 3/2 ^ 0 = 1.
T(n) = θ(n ^ (logb a))
Therefore T(n) = θ(nlog1.5 2) [n raised to the power of log 2 at base 1.5]

Limitations of Master Theorem:

Relation function cannot be solved using Mater's Theorem if:

1. T(n) is a monotone function
2. a is not a constant
3. f(n) is not a polynomial

Examples where Master theorem can't be applied.

1.T(n) = cosx
2. T(n) = nT(n/3) + f(n)