Recursion Tree Method

Till now, we have learned how to write a recurrence equation of an algorithm and solve it using the iteration method. This chapter is going to be about solving the recurrence using recursion tree method.

In this method, we convert the recurrence into a tree and then we sum the costs of all the levels of the tree. This will be clear from the example given below.

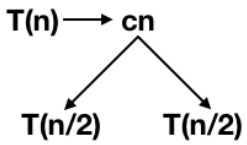
</>
Although this method uses the term 'tree' in this chapter, you will still be able to understand this chapter even without the knowledge of trees.

Let's take the recurrence equation from the previous chapter.

```
FOO1(A, left, right)
  if left < right
    mid = floor((left+right)/2)
    FOO1(A, left, mid)
    FOO1(a, mid+1, right)
    FOO2(A, left, mid, right)
```

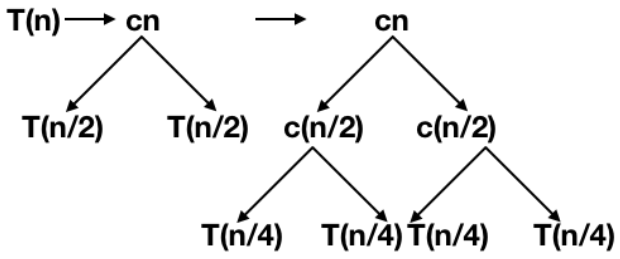
$$T(n) = cn + 2T\left(\frac{n}{2}\right)$$

Here, $T(n)$ is the running time of the entire algorithm and this running time is broken into two terms - cn and $2T\left(\frac{n}{2}\right)$. The statements for checking the condition (`if left < right`) and calculating the middle element (`mid = floor((left+right)/2)`) are taking constant time. However, the time taken by the `FOO2(A, left, mid, right)` function is linear i.e., $\Theta(n)$ (as stated in the previous chapter) and thus represented by cn, where c is a constant and n is the size of the array A. And then the problem is further broken down into two subproblems of $T\left(\frac{n}{2}\right)$ i.e., `FOO1(A, left, mid)` and `FOO1(a, mid+1, right)`.



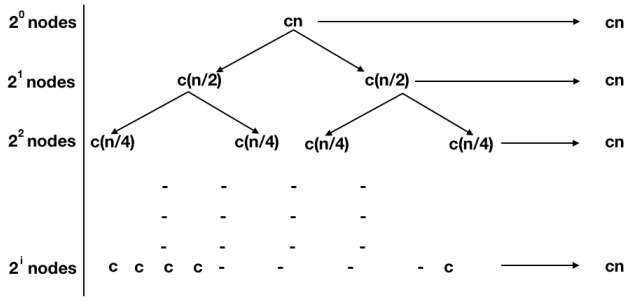
Here, we have represented $T(n)$ as two subproblems and the cost cn which is the total cost except these two subproblems.

Similarly, we can further break $T\left(\frac{n}{2}\right)$ as shown in the picture given below.



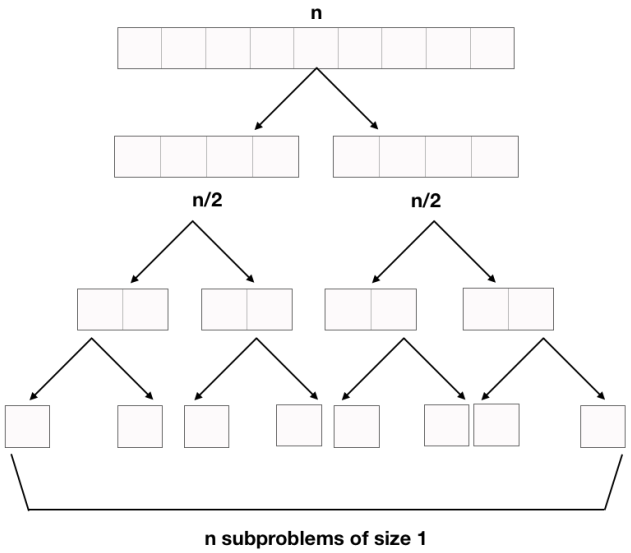
From the above picture, we can see that the cost of the top level is cn. Also, the next level has a cost of cn $\left(\frac{cn}{2} + \frac{cn}{2}\right)$. This is also true for the next level i.e., its cost is also $\left(\frac{cn}{4} + \frac{cn}{4} + \frac{cn}{4} + \frac{cn}{4}\right)$ i.e., cn.

To generalize, we can also say that at a level i, there are 2^i nodes and each of them is making a contribution of $\frac{cn}{2^i}$ to the cost and thus a total of $\left(2^i\right)\left(\frac{cn}{2^i}\right) = cn$.



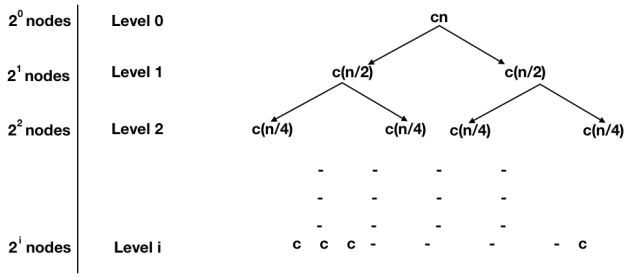
Now, we know that each level is taking cn time and if we calculate the total number of levels then we can easily know the total time taken by the algorithm.

From the previous chapter, we know that our base case is when the `left` is equal to the `right`. It means that we will stop breaking our problem into subproblems when the size will become 1, so there will be n (size of the problem) such subproblems which won't be further broken down into subproblems.



Since the size of our problem is n, so there will be n nodes without further breaking down into subproblems (or leaves). If the last level of the tree is level i, then

$$2^i = n$$
$$=> \lg(n) = i$$



Thus, the total number of levels are $1 + \lg(n)$ (counting of the levels is starting from 0).

Now, we know that there are a total of $1 + \lg(n)$ levels and each level is making a cost contribution of cn, so the total time should be

$$(l + \lg(n))cn$$

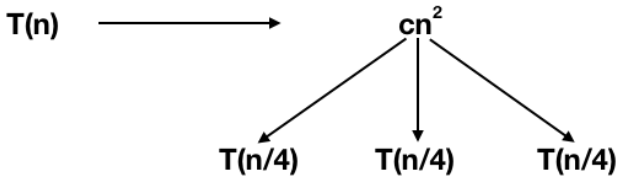
$$= c(n\lg(n) + cn)$$

By ignoring the lower order terms and the constant, we can get the order of the growth to be of $\Theta(n\lg(n))$.

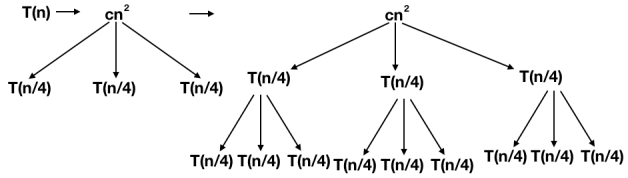
Let's take a look at one more example.

$$T(n) = 3T\left(\frac{n}{4} + cn^2\right)$$

Here, $T(n)$ is made up of two terms - cn^2 and $3T\left(\frac{n}{4}\right)$. Similar to the previous example, cn^2 simply represents the cost and $3T\left(\frac{n}{4}\right)$ can be viewed as breaking of the main problem into 3 subproblems with each of size $\frac{n}{4}$.



We can further break it as



By looking at the pattern, we can say that any level i has 3^i nodes and each node is making a contribution of $c\left(\frac{n}{4^i}\right)^2$ and thus a total cost of $3^i\frac{cn^2}{16^i} = \left(\frac{3}{16}\right)^i cn^2$.

Now, we know the contribution of each level, so the calculation of the total number of levels will lead us to the total cost of the algorithm.

We know that there won't be any division of the problem at the last level, so the size of each subproblem at the last level will be 1.

$$\text{Size of subproblem at level } i = \frac{n}{4^i}$$

$$\text{At the last level, size of the subproblem is } 1$$

$$=> 1 = \frac{n}{4^i}$$

$$=> n = 4^i$$

$$=> \log_4 n = i \text{ (Last level)}$$

Now, we know that the last level is $\log_4 n$ and the number of nodes at any level is 3^i, so the number of nodes at the last level can be written as $3^{\log_4 n} = n^{\log_4 3}$.

So at the last level, there are $n^{\log_4 3}$ nodes and each is making a contribution of $T(1)$. Thus, the total contribution of the last level = $\Theta(n\log_4 3)$

So, the total running time of the algorithm $T(n)$ can be written as

$$T(n) = cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \ldots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3})$$

$$= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{log_4 3})$$

The above equation is a GP (geometric progression) with the first term (a) = cn^2 and the common ratio (r) = $\frac{3}{16}$

One thing we can do here is:

$$= \sum_{i=0}^{\log_4 n-1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{log_4 3}) \leq \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{log_4 3})$$

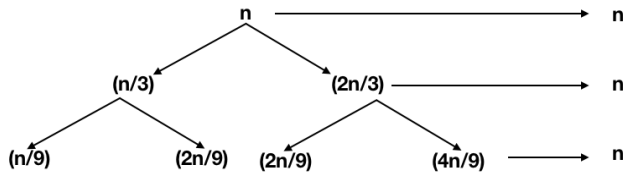
We have converted our problem from a finite GP to an infinite one and we can still get the upper bound (O), if not the tight bound (Θ).

Using the summation formula of an infinite GP ($\frac{a}{1-r}$),

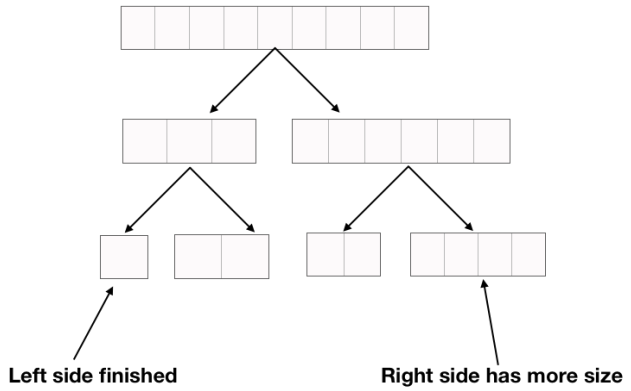
$$T(n) = \frac{1}{1-\frac{3}{16}} cn^2 + \Theta(n^{\log_4 3}) = O(n^2)$$

Let's look at one more example

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + cn$$

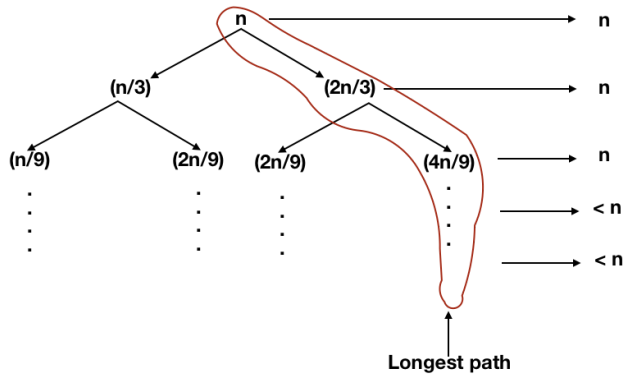


Here, the problem is divided into two subproblems of size $\frac{n}{3}$ and $\frac{2n}{3}$. Since the size of the problem on the left side is smaller than that of the right side, so the breaking into subproblems of the left side will finish earlier than that of the right one. Thus, we can focus on the rightmost subproblem at each level to get the upper bound of the algorithm because these subproblems are going to last longer.



One thing to be noticed from the above picture is that initially, the total cost of each level is cn but as the subproblems on the left side started finishing, it is becoming less than cn. So, if we make our calculations using cn, then the real time taken by the algorithm will be of course lesser but it will be an upper bound of our algorithm.

Thus, we have decided that we are going to calculate the upper bound by considering the cost of each level to be cn and considering only the longest path i.e., the number of levels (or nodes) in the longest path.



Looking at the longest path, we can see that the size of the subproblems is decreasing as $n \to \left(\frac{2n}{3}\right) n \to \left(\frac{2n}{3}\right)^2 n \to \ldots \to 1$. So, at any level i, the size of the subproblem can be written as $\left(\frac{2}{3}\right)^i n$. Thus at the last level,

$$1 = \left(\frac{2}{3}\right)^i n$$

$$=> n = \left(\frac{3}{2}\right)^i$$

$$=> i = \log_{\frac{3}{2}} n$$

Now, we know that there are $1 + \log_{\frac{3}{2}} n$ levels in the longest path and each level will take at most cn time i.e.,

$$T(n) < \underbrace{cn + cn + \ldots + cn}_{1+\log_{\frac{3}{2}} n \text{ times}} = cn\left(1 + \log_{\frac{3}{2}} n\right) = cn + cn\frac{\lg n}{\lg \frac{3}{2}}$$

Ignoring the lower order and constant terms, $T(n) = O(n \lg n)$

Till now, we have studied two methods to solve a recurrence equation. The third and last method which we are going to learn is the Master's Method (/course/algorithms-masters-theorem/). This makes the analysis of an algorithm much easier and directly gives us the result for 3 most common cases of recurrence equations. So, let's visit the next chapter and learn about the Master's Method.

> Don't watch the clock; do what it does. Keep going.
>
> - Sam Levenson

PREV (/course/algorithms-lets-iterate/) (/course/algorithms-masters-theorem/) NEXT