Solving Recurrences arising in the context of Algorithms

There are several ways to solve recurrences which arise in the analysis of Algorithms. The three main, relatively simple methods are:

• Guessing the solution in asymptotic terms and then proving the guess correct using mathematical induction. The solution is often difficult to guess, and sometimes, the induction proof is difficult to get precisely.

• Drawing a recursion tree and totalling the costs of all the nodes. Here, the cost of the leaf nodes are taken to be $\theta(1)$ and the size of the subproblems at the leaves are required to be reasonably small. The cost of the internal nodes is determined according to the cost of dividing and combining solutions of subproblems solved by recursive calls. This term is present in the recurrence relation itself. This method is particularly difficult when different subproblems have different sizes.

• The Master method is applicable for recurrences where each problem is broken into a number of subproblems and the size of the subproblems are uniform, a constant fraction of the original problem and the constant is less than 1.

Examples

• Substitution method

1. Show that the solution of $T(n) = T(n - 1) + n$ is $O(n^2)$.
   Assume, $T(m) \leq cm^2, \forall m < n$.
   Thus, $T(n) = T(n - 1) + n \leq c(n - 1)^2 + n = cn^2 + (1 - 2c)n + c \leq cn^2$, if $c \geq 1$.

2. Show that the solution of $T(n) = T \left( \left\lceil \frac{n}{2} \right\rceil \right) + 1$ is $O(\log n)$.
   Assume inductively that,
   $T(m) \leq c \log m$, for some $c > 0, \forall m < n$. 


From the given recurrence, we get,
\[ T(n) \leq c \log \left(\left\lceil \frac{n}{2} \right\rceil\right) + 1. \]

When \( n \) is even, this is the same as,
\[ T(n) \leq c \log n - c + 1 \leq c \log n, \text{ for } c \geq 1. \]

When \( n \) is odd, this is the same as,
\[ T(n) \leq c \log(n + 1) + (1 - c) \]
\[ = c \log n + c \log \left(1 + \frac{1}{n}\right) - c + 1 \leq c \log n. \]

If we choose \( c \), such that \( c \geq\frac{1}{1 - \log(1 + \frac{1}{n})} \).

This is monotonically increasing for \( n \geq 2 \), and has value \( \frac{1}{1 - \log(\frac{3}{2})} \), at \( n = 2 \).

This number is greater than 1, and thus works whether \( n \) is even or odd. It follows that \( T(n) = O(\log n) \).

3. Show that the solution of
\[ T(n) = 2T \left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n \text{ is } \Theta(n \log n). \]
Assume inductively that \( T(m) \leq c_1 m \log m, \forall m < n. \)

It follows that,
\[ T(n) \leq 2c_1 \left\lfloor \frac{n}{2} \right\rfloor \log \left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n. \]

When \( n \) is even, this is the same as,
\[ T(n) \leq c_1 n \log n + (1 - c_1) n \leq c_1 n \log n, \text{ whenever } c_1 \geq 1. \]

When \( n \) is odd, this is the same as,
\[ T(n) \leq c_1 (n - 1)(\log(n - 1) - 1) + n \]
\[ = c_1 n \log n + c_1 n \log \left(1 - \frac{1}{n}\right) - c_1 \log(n - 1) + (1 - c_1) n + c \]
\[ \leq c_1 n \log n, \text{ whenever,} \]
\[ c_1 \geq \frac{1}{n + \log(n-1) - n \log(1 - \frac{1}{n}) - 1} < 1, \forall n \geq 2. \]

Thus, in either case,
\[ T(n) = O(n \log n). \]

Now, assume inductively, that \( T(m) \geq c_2 m \log m, \forall m < n. \)

It follows that,
\[ T(n) \geq 2c_2 \left\lfloor \frac{n}{2} \right\rfloor \log \left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n. \]

When \( n \) is even, this is the same as,
\[ T(n) \geq c_2 n \log n + (1 - c_2) n \geq c_2 n \log n, \text{ whenever } c_2 \leq 1. \]

When \( n \) is odd, this is the same as,
\[ T(n) \geq c_2 (n - 1)(\log(n - 1) - 1) + n \]
\[ = c_2 n \log n + c_2 n \log \left(1 - \frac{1}{n}\right) - c_2 \log(n - 1) + (1 - c_2) n + c \]
\[ \geq c_2 n \log n, \text{ whenever,} \]
\[ c_2 \leq \frac{1}{n+\log(n-1)-n\log\left(1-\frac{1}{n}\right)-1} < 1, \forall n \geq 2. \]

Thus, in either case,
\[ T(n) = \Omega(n \log n). \]
We can conclude that \( T(n) = \theta(n \log n). \)

4. Show that the solution to \( T(n) = 2T\left(\left\lceil \frac{n}{2} \right\rceil + 17\right) + n \) is \( O(n \log n) \).

Assume inductively, that
\[ T(m) \leq cm \log m, \forall m < n. \]
If follows that,
\[
T(n) \leq 2c \left( \left\lceil \frac{n}{2} \right\rceil + 17 \right) \log \left( \left\lceil \frac{n}{2} \right\rceil + 17 \right) + n \\
= 2c \left( \frac{n+34}{2} \right) \log \left( \frac{n+34}{2} \right) + n.
\]
When \( n \) is even, this is the same as,
\[
T(n) \leq c(n + 34) \log(n + 34) - c(n + 34) + n \\
= cn \log n + cn \log \left(1 + \frac{34}{n}\right) + 34c \log(n + 34) + (1 - c)n - 34c \\
\leq cn \log n, \text{ whenever,}
\]
\[
c \geq \frac{n}{n - 34 \log(n + 34) - n \log\left(1 + \frac{34}{n}\right) + 34}.
\]
Clearly, when \( n \) is sufficiently large, the denominator in the above expression for \( c \) is positive, and the actual expression for \( c \), heads to a constant limit and is bounded above, for all \( n \) after the denominator becomes positive. Any value of \( c \) greater than that upper bound, works in this case.

When \( n \) is odd, this is the same as,
\[
T(n) \leq c(n + 33) \log(n + 33) - c(n + 33) + n \\
= cn \log n + cn \log \left(1 + \frac{33}{n}\right) + 33c \log(n + 33) + (1 - c)n - 33c \\
\leq cn \log n, \text{ whenever,}
\]
\[
c \geq \frac{n}{n - 33 \log(n + 33) - n \log\left(1 + \frac{33}{n}\right) + 33}.
\]
Analysis very similar to the one above shows that there exists a suitable constant and the base case is proved for a sufficiently large value of \( n \).

Thus in any case, the running time is \( T(n) = O(n \log n) \).

- Solution to recurrences in the assignment.

1. \( T(n) = 3T\left(\frac{3n}{2}\right) + \frac{1}{n} \).
   
   By the master method, we see that \( \frac{1}{n} = \Theta(n^{\log_3 3 - \epsilon}) \), for \( \epsilon = \log_3 3 + 1 \).
   
   Thus, it comes under case 1 of the Master theorem and the solution is, \( T(n) = \Theta \left( n^\left( \log_3 3 \right) \right). \)

3
By the recursion tree method, we say the cost at the root is $\frac{1}{n}$, the cost at the next level is $3\frac{1}{n}$, and at the next $3^2\frac{1}{n^2}$ and so on. Thus the cost at level $i$ (where the root is taken to be level 0), is $\frac{4}{n^i}$. We terminate the recursion when the size of the subproblem becomes, say 3. That is at a depth $i$ of the tree, where $(\frac{4}{3})^i n \leq 3$. This is the smallest value of $i$, such that $i \geq \log \frac{n - \log 3}{\log 4 - \log 3}$. That is, $\lceil \log \frac{n - \log 3}{\log 4 - \log 3} \rceil$. The total cost due to internal nodes, is thus $\sum_{i=0}^{\lceil \log \frac{n - \log 3}{\log 4 - \log 3} \rceil} \frac{4}{n^i} = \frac{1}{3\log 4} \left(4\frac{\log n - \log 3}{\log 4 - \log 3}\right)$. The number of leaves of the recursion tree is $3^\lceil \log \frac{n - \log 3}{\log 4 - \log 3} \rceil$. Asymptotically, the cost at the internal nodes is a polynomial of degree $\log \frac{4}{3} - 1$, while the cost at the leaves is a polynomial of degree $\log \frac{4}{3}$. The two actually are the same value. Thus the solution of the recurrence is $T(n) = \theta \left(n^{\frac{\log 4}{3}}\right)$.

2. $T(n) = T(\sqrt{n}) + \log n$.

This can be rewritten as,

$T \left(n^{\frac{1}{2^k}}\right) = T \left(n^{\frac{1}{2^{k+1}}}\right) + \log \left(n^{\frac{1}{2^k}}\right)$.

and similarly,

$T \left(n^{\frac{1}{2^{k+1}}}\right) = T \left(n^{\frac{1}{2^{k+2}}}\right) + \log \left(n^{\frac{1}{2^{k+1}}}\right)$.

$T \left(n^{\frac{1}{2^{k+2}}}\right) = T \left(n^{\frac{1}{2^{k+3}}}\right) + \log \left(n^{\frac{1}{2^{k+2}}}\right)$.

$\ldots$

$T \left(n^{\frac{1}{2^{k-1}}}\right) = T \left(n^{\frac{1}{2^k}}\right) + \log \left(n^{\frac{1}{2^{k-1}}\frac{1}{2^{k-2}}}\right)$.

Adding these equations up, we get,

$T(n) = \sum_{i=0}^{k-1} \log \left(n^{\frac{1}{2^i}}\right) + T \left(n^{\frac{1}{2^k}}\right)$

$= \theta(\log n)$.

Not here, that the number of levels in the recursion tree is around $\theta(\log \log n)$, and that there is only one leaf in the tree. However, summing the cost of the internal nodes, yields an geometric series with decreasing terms, hence, the order of grpwth is the same as the sum of the extended infinite geometric series.

3. $T(n) = T(n - 1) + 19$. 

4
Similarly,
\[
T(n - 1) = T(n - 2) + 19
\]
\[
\vdots
\]
\[
T(2) = T(1) + 19.
\]
Adding these equations up, we get,
\[
T(n) = 19(n - 1) + T(1) = \theta(n).
\]
4. \(T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + \sqrt{n}.
\
Observe, that the recursion tree corresponding to this recurrence is a **full binary tree**. As derived in class, in the context of binary heaps, we know that every full binary tree has exactly one leaf more than its number of internal nodes. Let us take the boundary condition where we recurse no further and look for a direct solution to be a problem of size 1. Note that the size of any internal node is the sum of the size of its two children and so on. The fact that the root represents a problem of size \(n\), immediately indicates that the recursion tree has \(n\) leaves. Note this **analysis** presents a much easier way to solve the problem by recursion tree method, than grappling with the uneven way in which the tree shapes. It follows, that the number of internal nodes is also approximately \(n\).
Here, we have the total cost at an internal level to be the sum of the square-roots of the subproblem sizes. Not that this sum is always upperbounded by the case in which the subproblems have uniform size. Thus, the cost at level \(i\) is at most
\[
2^i \sqrt{\frac{n}{2^i}} = 2^i \sqrt{n}.
\]
If we take this sum upto level \(\log_3 n\), we obtain
\[
\sqrt{n} \sum_{i=0}^{\log_3 n} 2^i = n \left(\frac{1}{2} + \frac{1}{2} \log_3 2\right).
\]
This is clearly, smaller asymptotically, than the number of leaves. If we took a similar sum upto the deepest level, we get,
\[
\sqrt{n} \sum_{i=0}^{\log_3 n} 2^i = n \left(\frac{1}{2} + \frac{1}{2} \log_3 2\right),
\]
which is clearly \(\omega(n)\), and we know that the number of leaves is roughly \(n\). Thus a guess is that the solution to the recurrence is at least \(\theta(n)\), and it could be \(n \log n\).
The cost at the deepest level with internal nodes is at most \(\frac{n}{2} \sqrt{2} = \frac{n}{\sqrt{2}}\). The sum of another \(\frac{n}{4}\) internal nodes could be bounded by
\( n^{2} \sqrt{3} = \frac{n}{2} \). This linear upper bound on the cost of a level, along with the fact that there are \( \theta(\log n) \) levels indicates an upper bound of not more than \( \theta(n \log n) \).

Thus, we know that \( T(n) = \Omega(n) \), given by the number of leaves, and that \( T(n) = O(n \log n) \), given by the above analysis. We can try and prove one of these using the substitution method.

Let us try and prove a lower bound of \( O(n \log n) \). Thus, we guess that the solution to the recurrence has been shown to be \( T(m) \geq cm \log m, \forall m < n \). Substituting this into the recurrence, yields,

\[
T(n) \geq cn \log n + \frac{2cn}{3} - cn \log 3 + \sqrt{n}.
\]

\( c \leq \frac{\sqrt{n}}{n(\log 3 - \frac{2}{3})} \). However, there is no constant \( c > 0 \) satisfying this condition, so the expression is \( o(1) \) and tends to zero as \( n \to \infty \).

Let us now try and prove an upper bound of \( n \) on the solution to the recurrence. Thus, we assume that \( T(m) \leq cm \log m, \forall m < n \). Substituting this into the recurrence, we get,

\[
T(n) \leq cn \log n + \frac{2cn}{3} + \sqrt{n} = cn + \sqrt{n}.
\]

This does not work. Let us try and strengthen the induction hypothesis by subtracting out a lower-order term. Thus, we assume, \( T(m) \leq c_1 m - c_2 \sqrt{m}, \forall m < n \). Substituting this into the recurrence yields,

\[
T(n) \leq c_1 n - c_2 \sqrt{n} + \frac{2c_1 n}{3} - c_2 \sqrt{\frac{2n}{3}} + \sqrt{n}.
\]

\( \leq c_1 n - c_2 \sqrt{n} \), whenever,

\[
1 + c_2 \left( \frac{1}{\sqrt{3}} - \frac{2}{\sqrt{3}} \right) < 0.
\]

This is true when \( c_2 > \frac{\sqrt{3}}{1 + \sqrt{2} - \sqrt{3}} \).

Hence, the solution to the given recurrence relation is \( T(n) = \theta(n) \).

NOTE: This is an important and difficult example. It shows the kind of resourcefulness one needs to show when trying to use the recursion tree method. Also, the power of the substitution method is demonstrated, to verify an exact asymptotic bound, when the recursion tree method did not provide a precise answer. This is also an example,
where a direct proof using the substitution method fails. One is obliged to subtract a lower order term, for the method to work.

- The Master method provides solution to recurrences of the form

\[ T(n) = aT\left( \frac{n}{b} \right) + f(n), \]

where \( a \geq 1 \) is an integer and \( b > 1 \) is some arbitrary real number. The solution falls into three cases.

1. If \( f(n) = O\left(n^{\log_b a - \epsilon}\right) \), for some constant \( \epsilon > 0 \), then the solution is \( T(n) = \Theta(n^{\log_b a}) \).
2. If \( f(n) = \Theta(n^{\log_b a}) \), then the solution is \( T(n) = \Theta(f(n) \log n) = \Theta(n^{\log_b a \log n}) \).
3. If \( f(n) = \Omega(n^{\log_b a + \epsilon}) \), for some constant \( \epsilon > 0 \), and if \( af\left(\frac{n}{b}\right) \leq cf(n) \), for some positive constant \( c < 1 \), and all sufficiently large \( n \), then \( T(n) = \Theta(f(n)) \).

Note, the application of this method is quite routine and I am not covering any solved examples, here. For the other two methods, some solutions have been provided, and a few difficult examples are included. You will, however, need to practise more examples on your own, as solving recurrence relations cannot be learned by merely looking at solved examples.