[Recurrences]

Design and Analysis of Algorithms (CSc 523)

Samujjwal Bhandari
Central Department of Computer Science and Information Technology (CDCSIT)
Tribhuvan University, Kirtipur,
Kathmandu, Nepal.

Recurrence relations, in the computer algorithms, can model the complexity of the running time, particularly divide and conquer algorithms. Recurrence relation is an equation or an inequality that is defined in term of itself. Recursions are useful in computer science because it makes the expression of an algorithm easy and can be solved easily. There are many natural problems that are easily described through recursion.

Examples

a. $a_n = a_{n-1} + 1, a_1 = 1$; (what function?)

b. $a_n = 2a_{n-1}$, $a_1 = 1$; (what function?)

c. $a_n = na_{n-1}$, $a_1 = 1$; (what function?)

   a. $a_n = n$ (polynomial)

   b. $a_n = 2^n$ (exponential)

   c. $a_n = n!$ (factorial)

Solving Recurrence Relations

See an example below

$T(n) = 2T(n-1) + k$, where $T(0) = 1$

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

$= 4(2T(n-3) + k) + 3k = 2^3T(n-3) + (1+2+4)k$

$= 2^4T(n-4) + (2^0+2^1+2^2+2^3)k$

…

$= 2^pT(n-p) + (2^0+2^1+ \ldots+2^{p-2}+2^{p-1})k$

let $p = n$ then,

$T(n) = 2^n + (2^0+2^1+ \ldots+2^{n-2}+2^{n-1})k$ { $\Sigma 2^i = 2^{n+1} - 1$, $i = 0$ to n}

$= 2^n + 2^n - 1 = 2.\ 2^n - 1 = \Theta(2^n)$.

Above we solved the recurrence relation but what are the available methods to solve the relation is next.

Solving Methods

No general procedure for solving recurrence relations is known solving the recurrence relation is an art. However there are generally used method that can solve problems of specific type and characteristics.

Linear homogeneous recurrence relations of degree k with constant coefficients is a recurrence relation of the form

$a_n = c_1a_{n-1} + c_2a_{n-2} + \ldots + c_ka_{n-k}$

where $c_1, c_2, \ldots, c_k$ are real numbers, and c_k != 0.

A sequence satisfying a recurrence relation above uniquely defined by the recurrence relation and the k initial conditions:

$a_0 = c_0, a_1 = c_1, \ldots, a_{k-1} = c_{k-1}$ This kind of linear homogenous recurrence relation can be solved by constructing characteristic equation and solving it for answer (interested student can follow books on recurrence relation for more detail).

Now we try to solve the recurrence relation through different methods.

Note: we do not take care of floors, ceilings and boundary conditions in solving the relations since we are solving for asymptotic value and those conditions generates extra constants only.

Substitution Method

Given a recurrence relation

T(n) = 1, when n = 1

T(n) = 2T(n/2) + n, when n > 1

In this method to obtain lower or upper bound

1. We guess the solution
2. Prove that solution to be true by mathematical induction.

The question comes over guessing i.e. how to guess for the solution?

For the above given relation we can guess the upper bound solution is O(nlogn) [we have already come through such a relation so guessing is easy]. For the guess to be true

T(n) <= c*nlogn must be satisfied for some positive constant c.
Since we guessed solution as O(nlogn) we can assume
T(n/2) <= c*n/2log(n/2)

Then

T(n) <= 2(c*n/2log(n/2)) + n
= c*nlog(n/2) + n
= c*n(logn –log 2) + n
= c*nlogn + (1-c)*n
<= c*nlogn [this step holds as long as c>=1]

Now we have to prove that the solution holds for boundary condition. For this mathematical induction is required.
we have , T(1) = 1,

T(1) <= c*1log1
<=0 (false)

But since we can have some $n>=n_0$, we may choose boundary condition for n=2, 3…
Now we can find c large enough to hold the above relation with the help of extended boundary conditions T(2) and T(3).

Why n=2 and 3 for boundary condition? Because after n>3 relation doesn't depend on T(1).

We obtain T(2) = 2T(1) + 2 = 4

T(3) = 2T(1) + 3 = 5 [3/2 = 1 as integer division)

If we put any c >=2 then,

T(2) <= c*2log2

<= 2c (true)

T(3) <= c*3log3

<= 3clog3 (true)

How to guess

Relate with similar previously done problem for e.g.

T(n) = 2T(n/2 + 3) + 3 + n

Here the problem is similar to the example previously done so we guess solution as O(nlogn).

Choose good lower bound and upper bound and converge the solution by reducing range. For e.g. for above relation since we see n on the relation we guess lower bound as $\Omega(n)$ and upper bound as $O(n^2)$. Then we can increase lower and decrease upper bound to get tighter bound $\Theta(nlogn)$.

Problem: Lower order term may defeat mathematical induction of substitution method.

Another Example

Consider the relation, $T(n) = 2T(n/2) + 1$

Guessing $T(n) = O(n)$

To show $T(n) <= c*n$

We can assume $T(n/2) <= c*n/2$ then,

$T(n) <= 2(c*n/2) + 1$

$<= c*n + 1$ is not $<= c*n$ for any c and n ,since we cannot subtract the 1.

Lets try to subtract the lower order term then,

Guess $T(n) = O(n)$

To show $T(n) <= c*n-b$ [since $c*n - b = O(n)$]

Assume $T(n/2) <= c*n/2 - b$ then,

$T(n) <= 2(c*n/2 - b) + 1$

$<= c*n - 2b + 1$

$<= c*n - b$,for $b >= 1$

Some time little modification on the variable can make the problem similar to other. For e.g. $T(n) = 2T(\sqrt{n}) + 1$

Take $m = \log n$ then $2^m = n$, i.e. $n^{1/2} = 2^{m/2}$ so

$T(2^m) = 2T(2^{m/2}) + 1$

Now rename $T(2^m)$ as $S(m)$ so the above relation can be written as

$S(m) = 2S(m/2) + 1$

This relation is simpler an we know solution to this, however you verify the result, so solution is S(m)= O(m) changing to T(n) we get T(logn)..

Example Factorial

Factorial(n)

if n < 1

then return 1

*else return n * Factorial(n-1)*

The above algorithm has recurrence relation as

$T(n) = 1$ n=0

$T(n) = T(n-1) + \Theta(1)$ n>0

Guess O(n) , Show T(n) <= c*n [proof left as an exercise]

Iteration Method

The iteration method does not require guessing but it need more experience in solving algebra. Here we are concerned with following steps.

1. Expand the relation so that summation dependent on n is obtained.
2. Bound the summation.

Take T(n) = 2T(n/2) + 1 , T(1) =1
Then we can expand the relation as

$$T(n) = 2T(n/2) + 1$$
$$= 2(2T(n/4) + 1) + 1 = 2^2T(n/2^2)+(1+2)$$
$$= 2^3T(n/2^3) + (1+2+4)$$
$$\ldots\ldots \text{ Up to } p^{th} \text{ term}$$
$$= 2^pT(n/2^p) + (2^0 + 2^1 + \ldots + 2^{p-1})$$

observe that $(2^0 + 2^1 + \ldots + 2^{p-1}) = 2^p - 1$

let $2^p = n$ then expanded relation becomes

$$T(n) = nT(1) + n-1$$
$$= \Theta(n)$$

Note : iterate until the boundary is met.

Example

Given the recurrence relation

$T(1) = \Theta(1)$ when $n = 1$

$T(n) = T(n/3) + \Theta(n)$ when $n>1$, solve using iteration method.

Here,

$$T(n) = T(n/3) + \Theta(n)$$
$$= T(n/9) + \Theta(n) + \Theta(n)$$
$$= T(n/3^3) + \Theta(n) + \Theta(n) + \Theta(n)$$
$$\ldots \text{ Up to } p^{th} \text{ term}$$
$$= T(n/3^p) + p\Theta(n)$$

let $n = 3^p$ then $p = \log_3 n$ so,

$$T(n) = T(1) + \log_3 n\Theta(n)$$
$$= \Theta(n \log_3 n)$$

This is Wrong

Right method

$$T(n) = T(n/3) + \Theta(n)$$
$$= T(n/9) + \Theta(n) + \Theta(n/3)$$
$$= T(n/3^3) + \Theta(n) + \Theta(n/3) + \Theta(n/9)$$
… Up to p^{th} term
$$= T(n/3^p) + \sum_{i=0}^{p} \theta(n/3^i)$$

let $n = 3^p$ then,

$$T(n) = T(1) + \Theta(n(\sum_{i=0}^{p} 1/3^i))$$
$$= 1 + \Theta(n\,(1/(1-1/3)) \qquad \{\sum_{i=0}^{p} a^i \approx 1/(1-a) \text{ when } 0 < a < 1\}$$
$$= \Theta(3n/2)$$
$$= \Theta(n).$$

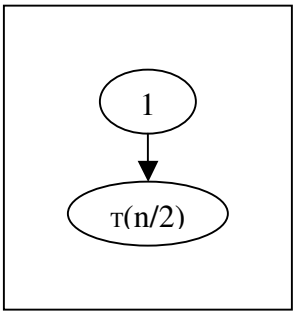
Recursion Tree Method

This method is just the modification to the iteration method for pictorial representation.

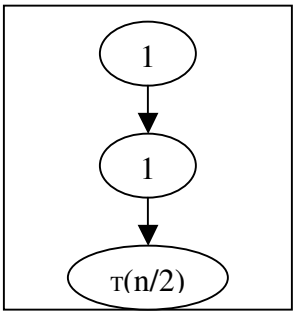
Given $T(n) = 1$

$T(n) = T(n/2) + 1$

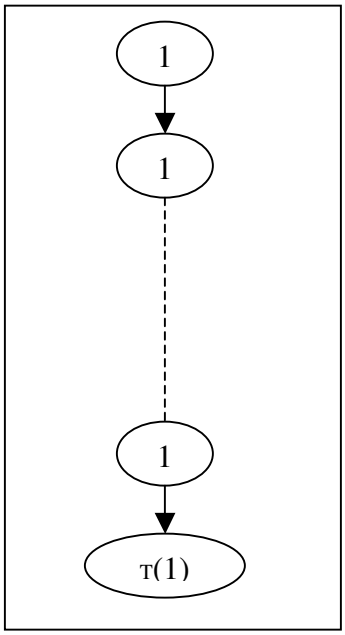
A



B



C

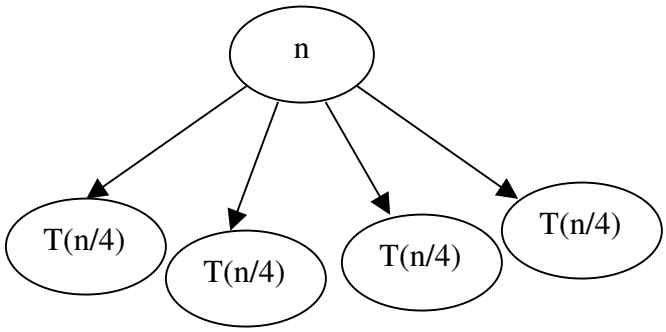


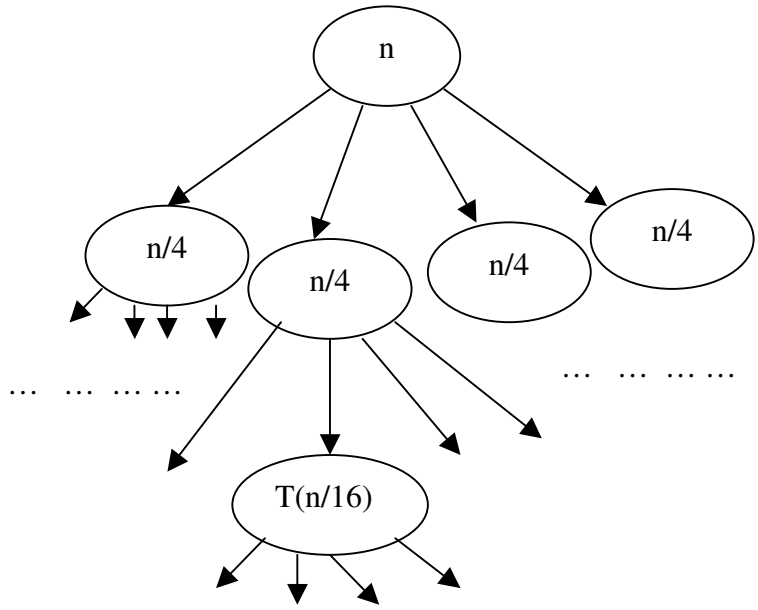
Here the height is log n so $T(n) = O(\log n)$

Example: Recursion Tree

Given $T(n) = 1$

$T(n) = 4T(n/4) + n$, solve by appealing to recurrence tree.





Continue like this ……

Add the terms at each level then you will obtain

n+ n +n + … + n up to height of the tree (logn time)
=nlogn

so T(n) = O(nlogn)

Master Method

Master method is used to solve the recurrences of the form

T(n) = aT(n/b) + f(n),

where a>=1 and b>1 are constants and f(n) is asymptotically positive function. This method is very handy because most of the recurrence relations we encounter in the analysis of algorithms are of the above kinds and it is very easy to use this method.
The above recurrence divides problem of size n into a sub problems each of size n/b where a and b are positive constants. The cost of dividing and combining the problem is given by f(n).

Master Theorem: let a>=1 and b>1 be constants, let f(n) be a function, and let T(n) be defined on the nonnegative integers by the recurrence

T(n) = aT(n/b) + f(n),

Where n/b is either $\lceil n/b \rceil$ or $\lfloor n/b \rfloor$. Then T(n) can be bounded asymptotically as follows.

1. If $f(n) = O(n^{\log_b a-\varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.
3. If $f(n) = \Omega(n^{\log_b a+\varepsilon})$ for some constant $\varepsilon > 0$, and if $a\ f(n/b) \leq c\ f(n)$ for some constant $c<1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

In each of the three cases we are comparing $n^{\log_b a}$ and f(n). Solution of the recurrence is derived from the larger of the two. The function that is smaller or greater must be polynomially smaller or greater by the factor of n^{ε} for case 1 and 3. Additional in case 3 is that regularity condition for polynomial $a\ f(n/b) \leq c\ f(n)$.

Examples

1. $T(n) = 9T(n/3) + n$

   $a=9, b=3, f(n) = n$

   $n^{\log_3 9} = n^2$

   $f(n) = n = O(n^{\log_3 9-\varepsilon})$, where $\varepsilon = 1$, **Case 1**

   So, $T(n) = \Theta(n^2)$

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

   $a=2, b=2, f(n) = n$

   $n^{\log_2 2} = n$

   $f(n) = n = \Theta(n^{\log_3 9})$, **Case 2**

   So, $T(n) = \Theta(n \log n)$

3. $T(n) = 3T(n/4) + n\log n$

$a=3, b=4, f(n) = n\log n$

$n^{\log_4 3} = O(n^{0.793})$

$f(n) = \Omega(n^{\log_4 3+\varepsilon})$, ε = (nearly)0.2,

For large n, $af(n/b) = 3(n/4)\log(n/4) \leq 3/4 n\log n = cf(n)$, $c=3/4$, **Case 3**

So, $T(n) = \Theta(n\log n)$

Exercises

1.

4.1.2: Show that the solution of $T(n) = 2T(\lfloor n/2 \rfloor) + n$ is $\Omega(n\log n)$. Conclude that solution is $\Theta(n\log n)$.

4.1.5: Show that the solution to $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$ is $O(n\log n)$.

2. Write recursive Fibonacci number algorithm derive recurrence relation for it and solve by substitution method.

3.

4.2.2: Argue that the solution to the recurrence $T(n) = T(n/3) + T(2n/3) + n$ is $\Omega(n\log n)$ by appealing to a recursion tree.

4.2.4: Use iteration to solve the recurrence $T(n) = T(n-a) + T(a) + n$, where $a >= 1$ is a constant.

4.

4.3.1: Use the master method to give tight asymptotic bounds for the following recurrences.

4.3.2: The running time of an algorithm A is described by the recurrence $T(n) = 7T(n/2) + n^2$. A competing algorithm A' has a running time of $T'(n) = aT'(n/4) + n^2$. What is the largest integer value for a such that A' is asymptotically faster than A?