Problem 1-1. (Constructing Recurrences)

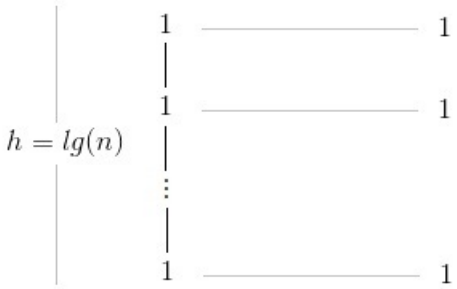
(a) Construct the recurrence for the binary search algorithm, and show that it is a $\Theta(lgn)$ algorithm via the iteration/tree method.

Solution:

$T(n) = T(n/2) + \Theta(1)$ [the $\Theta(1)$ part is for the compare operation]
with $T(1) = \Theta(1)$

$T(n) = T(n/2) + 1$
$= (T(n/4) + 1) + 1$
$= T\left(n/2^i\right)\left(\sum_{k=0}^{i-1} 1\right)$
$= T\left(n/2^{lg(n)}\right)\left(\sum_{k=0}^{lg(n)-1} 1\right)$
$= \Theta(1) * lg(n)$
$= \Theta(lg(n))$

Here's an image of the tree method:

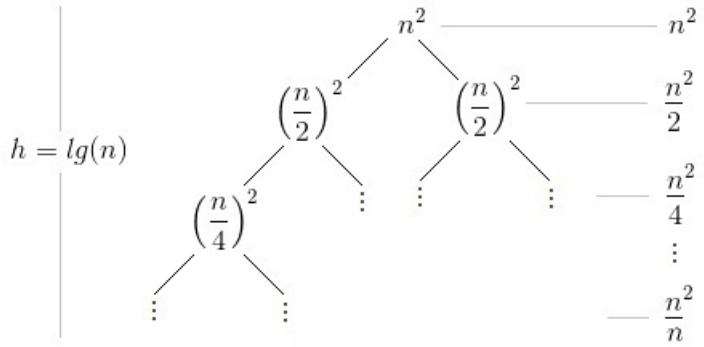


$$\Theta\left(\sum_{i=1}^{lg(n)} 1\right) = \Theta\left(lg(n)\right)$$

(b) Find the asymptotic runtime of the recurrence $T(n) = 2T(n/2) + n^2$ via the master theorem and via the tree/iteration method.

Solution: To solve the recurrence via the master theorem, apply case 3: a = b = 2, $log_b a = 1$, so $f(n) = n^2 = \Omega(n^{log_b a + \varepsilon} = n^{1+\varepsilon})$ for $\varepsilon = 0.1$ (any value between 0 and 1 will do). Then $T(n) = \Theta(f(n)) = \Theta(n^2)$.

To solve the recurrence via the iteration method, expand the expression repeatedly:
$T(n) = 2T(n/2) + n^2$
$= 4T(n/4) + (1 + \frac{1}{2})n^2$
$= 8T(n/8) + (1 + \frac{1}{2} + \frac{1}{4})n^2$
$= 2^i T(n/2^i) + (\sum_{k=0}^{i-1} \frac{1}{2^k})n^2$
$= 2^{lg(n)} T(n/2^{lg(n)}) + (\sum_{k=0}^{lg(n)-1} \frac{1}{2^k})n^2$
$= n\Theta(1) + (2 - \frac{2}{n})n^2$
$= \Theta(n^2)$
Here's an image of the tree method:



$$\Theta(n^2 + \frac{n^2}{2} + \frac{n^2}{4} + ... + \frac{n^2}{n}) = \Theta\left(\sum_{k=0}^{lg(n)} \frac{n^2}{2^k}\right) = \Theta\left(2n^2 - \frac{n^2}{n}\right) = \Theta\left(n^2\right)$$

Problem 1-2. (Master Theorem)

For each of the following, use the Master Theorem to determine the asymptotic growth of T(n), or determine that the Master Theorem does not apply. If it does not apply, describe why.

(a) $T(n) = 7T(n/2) + n^2$ (Strassen's method for matrix multiplication)
Solution: $\Theta(n^{2.81})$ => Case 1 a = 7, b = 2, f(n) = n^2 => $log_b a = 2.81$ => f(n) = $O(n^{log_b a - \varepsilon})$

(b) $T(n) = 3T(n/4) + nlog(n)$
Solution: $\Theta(nlog(n))$ => Case 3
a = 3, b = 4, f(n) = $nlog(n)$ => $log_b a = 0.79$ => f(n) = $\Omega(n^{log_b a + \varepsilon})$

(c) $T(n) = 4T(n-1) + 4^n$

Solution: $\Theta(n4^n)$ => Changing Variable + Case 2

Change n to $lg(m)$, so that m = 2^n.

T(n) = T($lg(m)$) = $4T(lg(m/2)) + m^2$

Let S(m) = T($lg(m)$) = T(n) => S(m) = 4S(m/2)+m^2

a = 4, b = 2, f(n) = n^2 => $log_b a = 2$ => f(n) = $\Theta(n^{log_b a})$

S(m) = $\Theta(m^2 lg(m))$ = T(n) = $\Theta(2^{2n} lg(2^n)) = \Theta(n4^n)$

(d) $T(n) = 3T(n/3) + n/lg(n)$

Solution: Cannot be solved. Case 1 doesn't apply since f(n) = n/$lg(n)$ is not polynomially smaller than $n^{log_3 3}$ for any $\varepsilon > 0$.

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

Solution: Cannot be solved => a is not constant

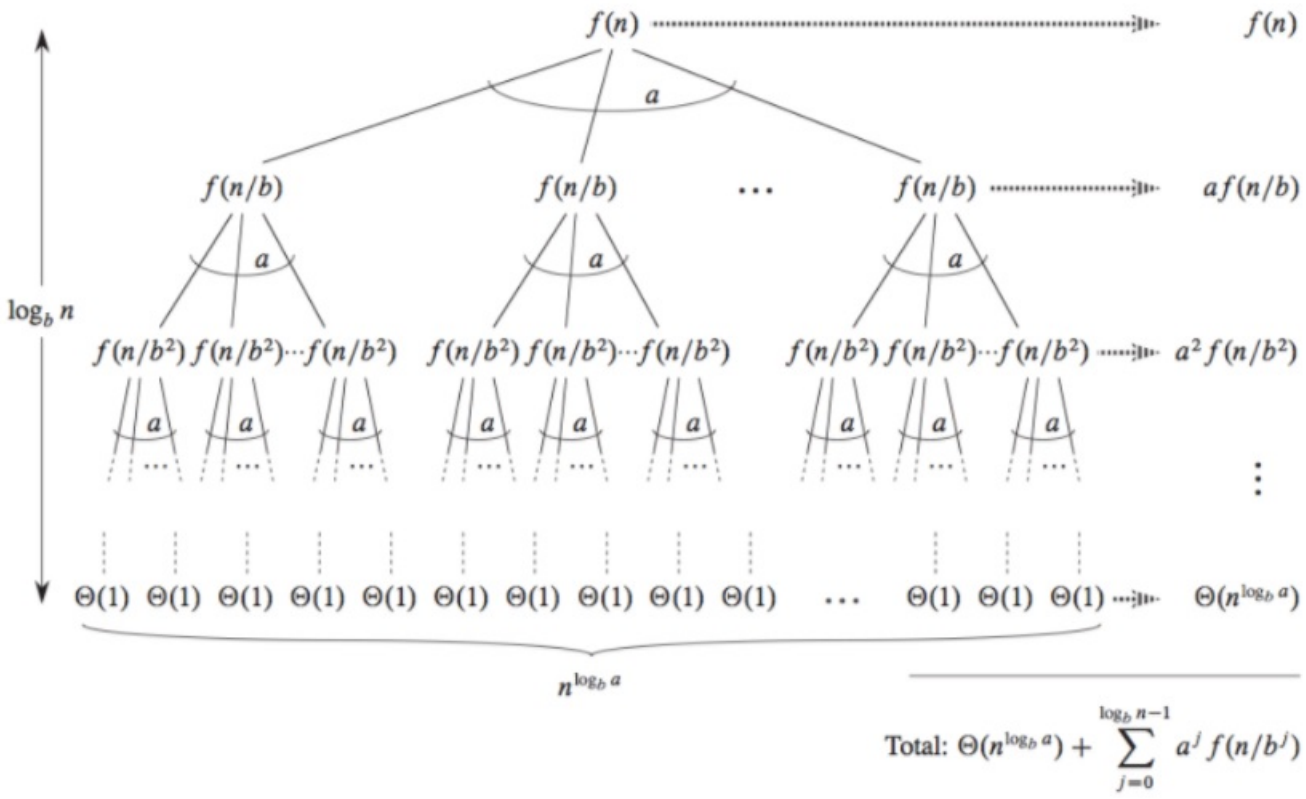
(f) $T(n) = T(n/2) + n(2 - cos(n))$

Solution: Cannot be solved => Case 3 would apply, but there is a regularity violation.

Problem 1-3. (Understanding the Master Method)

The LIGHT side vs. the DARK side (learning to interpret the master theorem). This question is meant for the students to gain a better intuition behind the Master Theorem's 3 cases.

The following questions refer to the generalized recursion tree:



(a) At each level j = $0, 1, 2, 3...log_b(n)$, how many subproblems are there?

Solution: a^j

At each level j, we divide each problem in a subsequent subproblems.

This is why (a) is called the Rate of Subproblem Proliferation (RSP)

(b) At each level j = $0, 1, 2, 3...log_b(n)$, what is the size of each subproblem?

Solution: $n/(b^j)$

At each level j, we shrink the subproblem size by a factor of b.

Since each problem of size n take $O(n^d)$ amount of work, the amount of work for each subproblem is shrinking by a factor of b^d

This is why (b^d) is called the Rate of Work Shrinkage (RWS)

(c) What is the total amount of work done at a given level j?

Solution: We start by noting that at level j we have a^j subproblems, and each subproblem take $(n/b^j)^d$ amount of work (we raise to the d power since the recurrence says that's the amount of work that each subproblem of size n takes).

Thus, we have $c * (a^j)(n/b^j)^d$ amount of work at level j.

After grouping some terms, we get $c * n^d * (a/b^d)^j$ total operations for the entire algorithm.

When we compare the rate of Subproblem Proliferation (dark side) to the Rate of Work Shrinkage (light side) there are three options:

1) RSP = RWS (the force is finally balanced!)

2) RSP < RWS

3) RSP > RWS

(d) For each of these situations, explain where most of the work is going to be done and why (in the root level, at the leaves, or the same amount per level)

Solution: If RSP = RWS, work done at each level will remain constant.

If RSP < RWS, there is less work done at each level, so most work is done at the root

If RSP > RWS there is more work done at each level, so most of the work will be done at the leaves

Problem 1-4. (Substitution and bounding)

(a) Prove that $T(n) = T(3n/4) + T(n/4) + n^2 = \Theta(n^2)$ using the substitution method.

Solution:

First, observe that T(n) is $\Omega(n^2)$ because it's at least n^2, because there's an n^2 in the recurrence. Then we only need to prove that T(n) is $O(n^2)$.

Observe that we can't easily upper bound this quantity as before: Applying the master method to $S(n) = 2T(3n/4) + n^2$ won't work because $log_{4/3}(2) > 2$.

So we do need to use the substitution method. We seek to prove by induction on n that for a fixed c, $T(n) \leq cn^2$ for all values of n.

We'll need to find a c such that our inductive hypothesis is strong enough for the inductive step. The inductive step looks like this:

Our inductive hypothesis is that for all values of k less than n, $T(k) \leq ck^2$. Now for the value n:

$T(n) = n^2 + T(3n/4) + T(n/4)$. By our inductive hypothesis, since 3n/4 and n/4 are both less than n, in particular $T(3n/4) \leq c(3n/4)^2 = 9cn^2/16$, and $T(n/4) \leq c(n/4)^2 = cn^2/16$.

Then $T(n) \leq n^2 + 9cn^2/16 + cn^2/16$. This in turn will be less than cn^2 if c is 16 (or anything larger than 8/3), so there is a c for which our inductive step will succeed.

(Formally, we'll also need a base case, but T(1) is some constant, which we can call d. Then, we can let the constant for our induction be max(2d, 16). Then our base case holds, and our inductive step holds just as it did before.)

(b) Prove that $T(n) = T(n/3) + T(2n/3) + n^2 = \Theta(n^2)$ by bounding it above and below by $\Theta(n^2)$ functions.

Solution: We can't apply the Master Theorem directly because the subproblem sizes are different. However, observe that T(n) is lower bounded by $S(n) =$

$S(n/3)+S(n/3)+n^2$. Here, we can apply the Master Theorem with $log_3(2) < 2$, so $S(n) = \Theta(n^2)$.

Also observe that T(n) is upper bounded by $R(n) = R(2n/3)+S(2n/3)+n^2$. Here, we can apply the Master Theorem with $log_{3/2}(2) < 2$, so $R(n) = \Theta(n^2)$.

Since T(n) is between two $\Theta(n^2)$ algorithms, T(n) is in $\Theta(n^2)$ itself.

(c) Find the Θ-class of T(n) = T(n/2) + T(n/4) + n first by establishing upper and lower bounds, then prove the correctness of your guess by the substitution method.

Solution: Establishing bounds for the guess:

T(n) is upper bounded by $R(n) = 2R(n/2)+n = \Theta(nlg(n))$
T(n) is lower bounded by $S(n) = 2S(n/4)+n = \Theta(n)$,
so T(n) is between $\Theta(n)$ and $\Theta(nlg(n))$. In other words, T(n) is $\Omega(n)$ and $O(nlg(n))$.
We'll have its Θ-class if we can establish that it's either $\Omega(nlg(n))$ or $O(n)$.

(We don't actually need to use the Master Method for the lower bound- it's clear that T(n) is $\Omega(n)$, because T(n) is at least n, because there's an n in the recurrence already.)

Let's try to prove that T(n) is $\Omega(nlg(n))$: (incorrect guess) That is, we want to show that for some fixed constant C, $\Omega(nlg(n))$
$T(n) \geq Cnlg(n)$

Our inductive hypothesis will assume that: $//T(n/2) \geq C\frac{n}{2}lg\left(\frac{n}{2}\right)$
$T(n/4) \geq C\frac{n}{4}lg\left(\frac{n}{4}\right)$

Then, we try to prove our lower bound: $T(n) = T(n/2)+T(n/4)+n$
$\geq C\frac{n}{2}lg\left(\frac{n}{2}\right)+C\frac{n}{4}lg\left(\frac{n}{4}\right)+n$
$= C\frac{n}{2}(lg(n)-1)+C\frac{n}{4}(lg(n)-2)+n$
$= C\frac{n}{2}lg(n)-C\frac{n}{2}+C\frac{n}{4}lg(n)-C\frac{n}{2}+n$
$= \frac{3}{4}Cnlg(n)-(C-1)n$
$< Cnlg(n)$

It doesn't work (for any value of C)!

Let's try to prove that T(n) is $O(n)$: (correct guess)

$O(n)$
$T(n) \leq Cn$
assume :
$T(n/2) \leq C\frac{n}{2}$
$T(n/4) \leq C\frac{n}{4}$

$T(n) = T(n/2)+T(n/4)+n$
$\leq C\frac{n}{2}+C\frac{n}{4}+n$
$= \left(C\frac{1}{2}+C\frac{1}{4}+1\right)n$

$= \left(\frac{3}{4}C + 1\right) n$
$\leq Cn(for : C \geq 4)$

(As before, we'll also need a base case, but T(1) is some constant, which we can call d. Then, we can let the constant for our induction be max(2d, C). Then our base case holds, and our inductive step holds just as it did before.)