Master Theorem

Suppose that we have a recurrence relation that looks like

$$f(n) = 2f(n/4) + n^{1/2}$$

We would like to come up with a Big-O expression for f(n)
Assumptions:

- We will assume that f(n) is constant for small values of n. This should contribute a constant factor to the final value.
- We will assume that the fractional part of n/4 will not matter.

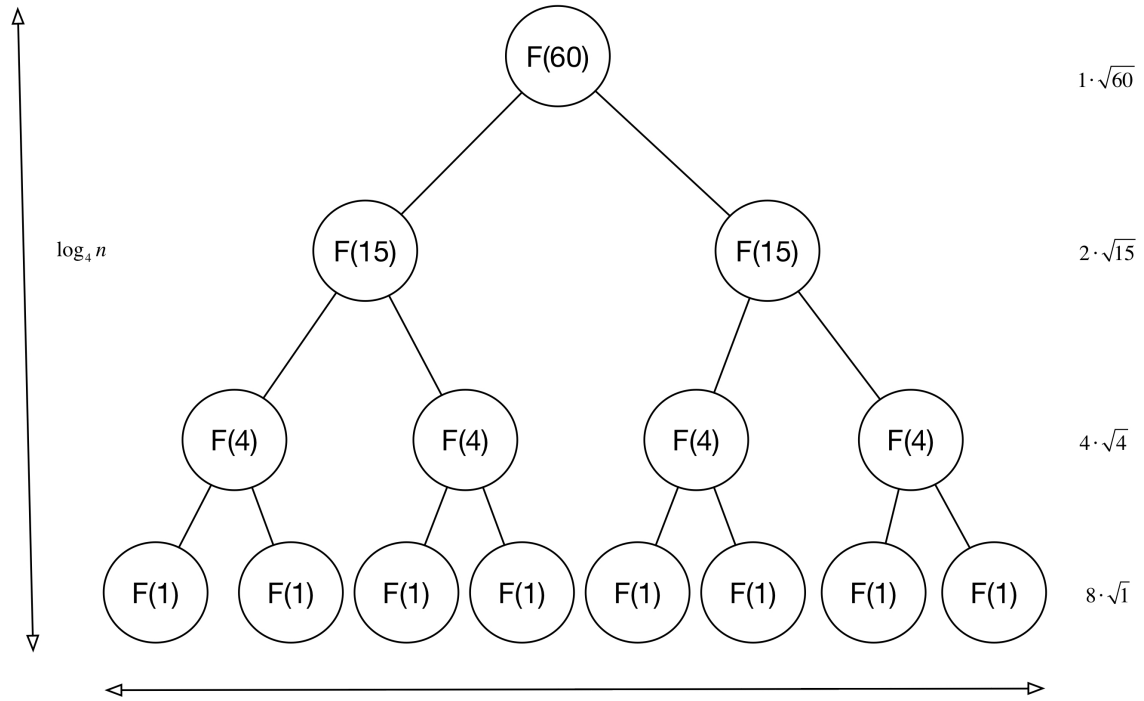
The height will be related to the division and is $\log_4 n$
The number of calls on the final layer will be determined by the height and branching factor $2^{\log_4 n}$. We can rewrite this by changing the base on the log

$$\log_4 n = \frac{\log_2 n}{\log_2 4} = \frac{\log_2 n}{2}$$

And then plugging this into the exponent

$$2^{\log_4 n} = 2^{\log_2 n/2} = \left(2^{\log_2 n}\right)^{1/2} = n^{1/2}$$

Lets take a look at the pattern for F(60). We will ignore the fact that this is not a power of 4 and we may get fractional parts when we divide by 4.



$$2^{\log_4 n} = 2^{\log_2 n/2} = \left(2^{\log_2 n}\right)^{1/2} = n^{1/2}$$

We have two sources of work… we have the work that is associated with the recursion and we have the non-recursive work.

If we look at each level we see that the total non-recursive works is given by

$$1 \cdot \sqrt{60} + 2 \cdot \sqrt{15} + 4 \cdot \sqrt{4} + 8 \cdot \sqrt{1}$$

To find the total work we sum up the cost of the recursion and add in the work from each layer.

Generalizing

We want to come up with a general notion for how much is being done in a general form. (Note: There is a more complicated version of the Master's Theorem that allows us to handle situations where the non-recursive work is an arbitrary function, but we will not study it in this class.)

$f(n) = af(n/b) + cn^d$

$a \geq 1$

b is an integer greater than 1

c is positive and real

d is non-negative and real

We want to compare two things… How much recursive work is there is at the last layer verses how much non-recursive work there is at the first layer.

- Non-recursive work on the first layer is n^d
- Recursive work at the last layer is $a^{\log_b n} = \left(b^{\log_b a}\right)^{\log_b n} = \left(b^{\log_b n}\right)^{\log_b a} = n^{\log_b a}$ (We used $a = b^{\log_b a}$.)

We have three cases based on comparing:

	Last layer	First Layer
	$n^{\log_b a}$	n^d
Or equivalently	a	b^d

Case 1) First layer dominates: $n^{\log_b a} < n^d$ or ($a < b^d$). Bound the function by the first layer.
Case 2) The first and last layers are equivalent: Bound the function by work on the first layer times the number of layers.
Case 3) Last layer dominates: Bound the function by the recursive work on the last layer.

$$f(n) = af(n/b) + cn^d$$

$a \geq 1$
b is an integer greater than 1
c is positive and real
d is non-negative and real

$$f(n) = \begin{cases} O\left(n^d\right) & a < b^d \\ O\left(n^d \log n\right) & a = b^d \\ O\left(n^{\log_b a}\right) & a > b^d \end{cases}$$

Example Applications:

$F(n) = 2F(n/4) + n^{1/2}$

a=2, b=4, c=1, d=1/2

Compare 2 with $(4)^{1/2} = 2$. This is the second case and $F(n) = O\left(n^{1/2} \log n\right)$

Binary Search:

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

a=1, b=2, c=1, d=0

Compare 1 with $(2)^0 = 1$. This is the second case and $B(n) = O(\log n)$

K – Select Best Case:

$K(n) = K(n/2) + n$

a=1, b=2, c=1, d=1

Compare 1 with $(2)^1 = 2$. This is the first case and $K(n) = O(n)$

Merge Sort :

$M(n) = 2M(n/2) + n$

a=2, b=2, c=1, d=1

Compare 2 with $(2)^1 = 2$. This is the second case and $M(n) = O(n)$

A made up example:

$Q(n) = 3Q(n/2) + n$

a=3, b=2, c=1, d=1

Compare 3 with $(2)^1 = 2$. This is the third case and

$Q(n) = O\left(n^{\log_2 3}\right) = O\left(n^{1.584962500721156}\right)$