The Master Theorem

The Master Theorem provides us with a general method for analyzing recursive algorithms of a certain form. Suppose an algorithm solves a problem by dividing a problem of size n into some number, say $a \geq 1$, of subproblems, each of size $\frac{n}{b}$ where $b > 1$. Each of the subproblems is solved recursively and takes time $T(\frac{n}{b})$ to solve. The time to divide and combine the results of all the subproblems is $f(n)$. The recurrence for the total amount of time to solve the problem is

$$T(n) = a\,T(\tfrac{n}{b}) + f(n) \text{ where } a \geq 1 \text{ and } b > 1.$$

The Master Theorem gives us a method to solve this recurrence, that is, to find a tight asymptotic bound for $T(n)$, as a function of n.

The Master Theorem Suppose $T(n) = a\,T(\frac{n}{b}) + f(n)$ where $a \geq 1$ and $b > 1$.

Case 1: If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then

$$T(n) = \Theta(n^{\log_b a})$$

Case 2: If $f(n) = \Theta(n^{\log_b a})$, then

$$T(n) = \Theta(n^{\log_b a} \lg n)$$

Case 3: If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$ and

$a \cdot f(\frac{n}{b}) \leq c\,f(n)$ for some constant $c > 1$ and sufficiently large n, then

$$T(n) = \Theta(f(n))$$

NOTE: To show work for using the Master Theorem,

- demonstrate comparing $f(n)$ to $n^{\log_b a}$,
- show your calculation for logarithms,
- clearly indicate your choice of ε and c, as appropriate,
- identify which case of the Master Theorem applies,
- show the solution for $T(n)$.

```
foo(n)
   if (n == 1)
     return
   for i = 1 to n
   {
     for j = 1 to n
     {
         //some computations done here
     }
   }
   for i = 1 to 8
   {
     foo(n/2)
   }


bar(n)
   if (n == 1)
     return
   for i = 1 to n
   {
     for j = i+1 to n
     {
         //some computations done here
     }
   }
   for k = 1 to 9
     bar(n/3)




baz(a, n)      // a is an array of size n
   if (n == 1)
     return
   for i = 1 to n
   {
     for j = 1 to i
     {
       for k = 1 to j
       {
           //some computations done here
       }
     }
   }
```

```
m = n/2
baz(a[1..m], n/2)    // a[1..m] represents left half of array a
baz(a[m+1..n], n/2)  // a[m+1..n] represents right half of array a
```