Lecture Notes on  
Divide-and-Conquer Recurrences

The following important class of recurrences often arise in the analysis of algorithms that are based on Divide-and-Conquer strategy.

\[
T(n) = \begin{cases} 
    a \frac{T(n)}{b} + c n^\alpha, & n > 1 \\
    d, & n = 1
\end{cases} \tag{1}
\]

Note that \(a, b, c, d\) and \(\alpha\) are constants (determined by the particular algorithm) and \(n\) is assumed to be an integer power of \(b\), \(n = b^k\).

**Theorem 1:** Let \(h = \log_b a\). The solution of the above class of recurrences is as follows:

\[
T(n) = \begin{cases} 
    A n^h + B n^\alpha = \Theta(n^\alpha), & h < \alpha \\
    A n^h + B n^\alpha = \Theta(n^h), & h > \alpha \\
    A n^h + B n^h \log n = \Theta(n^h \log n), & h = \alpha
\end{cases} \tag{2}
\]

where \(A\) and \(B\) are some constants for each case. (This theorem is a slight variation of a result called the Master Theorem in several textbooks on algorithms, including Cormen et al, and Goodrich-Tamassia.)

**Proof:** We will use the repeated substitution method (also known as iteration method) to derive the solution.

\[
T(n) = cn^\alpha + aT(n/b) = cn^\alpha + ac(n/b)^\alpha + a^2T(n/b^2) = cn^\alpha + ac(n/b)^\alpha + a^2c(n/b^2)^\alpha + a^3T(n/b^3).
\]

We factor out \(cn^\alpha\), use the equality \((b^i)^\alpha = (b^\alpha)^i\), and continue with the repeated substitution.

\[
T(n) = cn^\alpha[1 + a/b^\alpha + (a/b^\alpha)^2 + \cdots + (a/b^\alpha)^{k-1}] + a^kT(n/b^k)
\]

Recall that \(n = b^k\), thus \(k = \log_b n\). Then, \(T(n/b^k) = T(1) = d\), and

\[
a^k = a^\log_b n = n^\log_b a = n^h.
\]

(The reason for the equality, \(a^\log_b n = n^\log_b a\), is seen by taking the \(\log_b\) of both sides to get \(\log_b n \log_b a\).) Therefore,

\[
T(n) = cn^\alpha \sum_{i=0}^{k-1} (a/b^\alpha)^i + dn^h. \tag{3}
\]

We now consider two cases, depending on whether the ratio \(a/b^\alpha\) in the above summation is equal to 1.
1. $a/b^\alpha \neq 1$, which means $\log_b a \neq \alpha$. That is, $h \neq \alpha$.

In this case, the summation in Eq.(3) is a geometric series, thus:

$$T(n) = c n^\alpha \frac{(a/b^\alpha)^k - 1}{a/b^\alpha - 1} + d n^h.$$

Again we use $a^k = n^h$ (as derived above), and $(b^\alpha)^k = (b^k)^\alpha = n^\alpha$ to obtain the solution:

$$T(n) = c n^\alpha \frac{n^h/n^\alpha - 1}{a/b^\alpha - 1} + d n^h$$

$$= c \frac{n^h - n^\alpha}{a/b^\alpha - 1} + d n^h$$

$$= \left(\frac{c}{a/b^\alpha - 1} + d\right) n^h - \left(\frac{c}{a/b^\alpha - 1}\right) n^\alpha$$

$$= A n^h + B n^\alpha.$$

This proves the first two forms of the solution in Eq.(2).

**Note:** Although we derived the values of the constants $A$ and $B$ (in terms of the constants $a, b, c, d, \alpha$), these values need not be memorized. We will see in the examples below how the constants $A$ and $B$ may be directly computed in a trivial way.

2. $a/b^\alpha = 1$, which means $\log_b a = \alpha$. That is, $h = \alpha$.

In this case, $\sum_{i=0}^{k-1} (a/b^\alpha)^i = k = \log_b n$. Thus,

$$T(n) = c n^\alpha \log_b n + d n^h$$

$$= c n^h \log_b n + d n^h.$$

This proves the third line in Eq.(2), and completes the proof of the Theorem.

Next, we will provide several examples that use the solution form established by this theorem.
Example 1: Finding MAX by divide-and-conquer

We discussed in class how to apply divide-and-conquer to find the maximum element in an array of size $n$. Assuming that $n = 2^k$, the number of key comparisons is given by the following recurrence.

$$f(n) = \begin{cases} 
0, & n = 1, \\
2f(n/2) + 1, & n \geq 2.
\end{cases}$$

The solution form may be obtained immediately from Theorem 1:

$$a = 2, \ b = 2, \ \alpha = 0, \ h = \log_b a = 1.$$ 

Since $h \neq \alpha$, the solution is of the form:

$$f(n) = An^h + Bn^\alpha = An + B.$$ 

Thus, $f(n) = \Theta(n)$. If we are interested in finding the exact solution, we may readily compute the constants $A$ and $B$. We will discuss two easy methods for finding the constants directly from the recurrence.

1. **Find the two constants by substituting two values in the recurrence.**
   (This method is applicable because we know from Theorem 1 that the solution form $f(n) = An + B$ is correct.)
   - $n = 1$: $f(1) = 0$ (from the recurrence)
     $$= A + B \quad \text{(from the solution form)}$$
   - $n = 2$: $f(2) = 2f(1) + 1 = 1$ (from the recurrence)
     $$= 2A + B \quad \text{(from the solution form)}$$

Solving the two equations $A + B = 0$ and $2A + B = 1$ finds the constants $A = 1$ and $B = -1$. Thus, $f(n) = n - 1$.

2. **Apply induction to verify the correctness of the solution and find the constants.**
   This method is useful in situations when we need to verify the correctness of the solution (i.e., if the solution form is a guess that needs to be verified).

For the induction base, $n = 1$, $f(1) = 0 = A + B$.

For $n \geq 2$, suppose that the solution is correct for all $m < n$. That is, suppose that $f(m) = Am + B$, $\forall m < n$.

Then, we will prove the solution is also correct for $m = n$. That is, we will prove that $f(n) = An + B$.

$$f(n) = 2f(n/2) + 1 \quad \text{from the recurrence for } n \geq 2$$
$$= 2[An/2 + B] + 1, \quad \text{from the hypothesis for } m = n/2$$
$$= An + 2B + 1$$
$$= An + B, \quad \text{if } 2B + 1 = B.$$ 

Solving the two relations (equations)

$$A + B = 0 \quad \text{(to satisfy the base)}$$
$$2B + 1 = B \quad \text{(to satisfy the induction step)}$$

finds the constants $A = 1$ and $B = -1$ (and completes the inductive proof). Therefore, $f(n) = n - 1$. 

3
Example 2: Binary Search

We discussed this algorithm in class. Assuming that \( n = 2^k \), the worst-case number of key comparisons is given by the following recurrence.

\[
f(n) = \begin{cases} 
1, & n = 1, \\
f(n/2) + 1, & n \geq 2.
\end{cases}
\]

We may find the solution from Theorem 1.

\[a = 1, \quad b = 2, \quad \alpha = 0, \quad h = \log_b a = 0.\]

Since \( h = \alpha \), Theorem 1 tells us the solution has the form:

\[
f(n) = A n^h + B n^h \log n = A + B \log n.
\]

Thus, \( f(n) = \Theta(\log n) \). The exact solution is found by computing the constants \( A \) and \( B \). Since the correctness of the solution form is already known from Theorem 1, we may simply plug in two values for \( n \) to find two relations.

- \( n = 1 \) : \( f(1) = 1 \) (from the recurrence)
  \[= A + B \log 1 = A. \] (from the solution form)

- \( n = 2 \) : \( f(2) = f(1) + 1 = 2 \) (from the recurrence)
  \[= A + B \log 2 = A + B. \] (from the solution form)

From the two relations

\[
A = 1 \\
A + B = 2
\]

we find the constants: \( A = 1 \) and \( B = 1 \). Therefore,

\[
f(n) = 1 + \log n.
\]