# Divide-and-Conquer Recurrences (Draft)
# (Updated Thu Mar 4 09:05:35 PST 2021)

We will now discuss a completely different type of recurrences. These recurrences come up very often in the analysis of algorithms based on the divide-and-conquer method. In this method, the original instance is divided into $b$ smaller pieces, the solutions are computed independently for these $b$ pieces and then combined together to obtain the solution of the original instance. Such divide-and-conquer algorithms include: binary search, merge-sort, quick-sort, algorithms for integer multiplication (minimizing the number of bit operations), matrix multiplication, and many other.

**Assumption:** For simplicity, we will assume that $n$, the input size, is of the form $n = b^k$, where $b \geq 2$ is an integer. In most cases $b = 2$, but in some applications other values of $b$ will be used. This way, each time we divide $n$ by $b$, the result will be integer.

**Note:** For divide-and-conquer recurrences, we typically do not need the exact solution, but only the asymptotic solution. (Since these recurrences represent the running time, asymptotic solutions are sufficient. Plus, the exact solutions are often difficult to determine.)

**Merge-sort.** In Merge-Sort, we divide the sequence into equal halves, sort them recursively, and then merge them together. Merging two sorted sequences of length $n/2$ takes $n$ comparisons. So the recurrence is:

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

and, say, for $n = 1$ assume $T(1) = 1$.

As before, let $n$ be a power of 2, and applying the recurrence to $T(n/2)$, we have

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2[2T(n/4) + n/2] + n \\ &= 4T(n/4) + 2n. \end{aligned}$$

We can repeat this substitution again, and again, up to $\log n$ times:

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 4T(n/4) + 2n \\ &= 8T(n/8) + 3n \\ &\ldots \\ &= 2^j T(n/2^j) + jn \\ &\ldots \\ &= nT(1) + n \log n \\ &= \Theta(n \log n). \end{aligned}$$

**A more general case.** Let's solve a more general recurrence:

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

where $a \geq 1$, $b \geq 2$ are some integers, and we assume that the initial condition is $T(1) = 1$. We also assume that $n$ is a power of $b$. (We will later show that this does not affect the asymptotic solution.)

Again, we do repeated substitutions:

$$\begin{aligned}
T(n) &= aT(n/b) + n \\
&= a[aT(n/b^2) + n/b] + n \\
&= a^2T(n/b^2) + (a/b)n + n \\
&\dots \\
&= a^jT(n/b^j) + n[(a/b)^{j-1} + \dots + (a/b)^2 + (a/b) + 1] \\
&\dots \\
&= a^{\log_b n}T(1) + n \cdot \sum_{i=0}^{\log_b n - 1} (a/b)^i \\
&= n^{\log_b a} + n \cdot \sum_{i=0}^{\log_b n - 1} (a/b)^i
\end{aligned}$$

To estimate the second term and the whole expression, we have three cases.

**Case 1:** $a = b$. The first term is $n$. In the summation, we have $\log_b n$ terms and they are all equal $a/b = 1$, so the second term is $n \log_b n$. Thus we get $T(n) = \Theta(n \log n)$.

**Case 2:** $a < b$. The second term is now a geometric series with the ratio smaller than 1, so $\sum_{i=0}^{\log_b n-1} (a/b)^i = \Theta(1)$. The first term is $n^{\log_b a}$ with $\log_b a < 1$, so we get $T(n) = \Theta(n)$.

**Case 3:** $a > b$. Summing the geometric series in the second term, we get

$$\sum_{i=0}^{\log_b n - 1} (a/b)^i = \frac{(a/b)^{\log_b n} - 1}{(a/b) - 1} = \tfrac{b}{a-b}(a^{\log_b n}/b^{\log_b n} - 1) = \tfrac{b}{a-b}(n^{\log_b a}/n - 1)$$

So

$$T(n) = n^{\log_b a} + \tfrac{b}{a-b}(n^{\log_b a} - n) = \Theta(n^{\log_b a}).$$

This gives us the solution to the recurrence $T(n) = aT(n/b) + n$. The same proof works if we replace $n$ by $cn$, for some constant $n$. In fact, it can be generalized further, by allowing some additive term $cn^d$, for some $d \geq 0$. Thus we get:

**Theorem 1 ( Master Theorem )** *Let $a \geq 1$, $b > 1$, $c > 0$ and $d \geq 0$. If $T(n)$ satisfies the recurrence*

$$T(n) = aT(n/b) + cn^d,$$

*then*

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{for } a > b^d \\ \Theta(n^d \log n) & \text{for } a = b^d \\ \Theta(n^d) & \text{for } a < b^d \end{cases}$$

*Note:* Our proof, so far, assumes that $n$ is a power of $b$. Alternatively we could allow $n$ to be a real number, not necessarily integer. But in applications to algorithm analysis, $n$ can be any integer and divide-and-conquer recurrences involve rounding, for example in MergeSort, the recurrence would actually be $T(n) =$

$T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n$. We will show later that this rounding does not affect the asymptotic value of the solution.

**Example 1.** For the pseudo-code below, give the number of letters printed as a function of $n$, using the $\Theta$-notation.

```
procedure PrintAs($n$)
    if $n \le 2$ then
        print("A")
    else
        for $j \leftarrow 1$ to $n^2$
            do print("A")
        for $i \leftarrow 1$ to 5 do
            PrintAs($n/2$)
```

For any $n \ge 2$, we print $n^2$ "A"s and we make 5 recursive calls, each with parameter $n/2$. So the number of letters printed satisfies the following recurrence equation:

$$A(n) = 5A(n/2) + n^2$$

To estimate $A(n)$, we use the Master Theorem. In this recurrence, we have $a = 5$, $b = 2$ and $d = 2$, so $a > b^d$. Thus the solution is $A(n) = \Theta(n^{\log 5})$.

**Example 2.** For the pseudo-code below, give the number of letters printed as a function of $n$, using the $\Theta$-notation.

```
procedure PrintBs($n$)
    if $n \ge 3$ then
        print("B")
        PrintBs($n/3$)
```

For any $n \ge 3$, we print one "B" and we make one recursive call, each with parameter $n/3$. So the number of letters printed satisfies the following recurrence equation:

$$B(n) = B(n/3) + 1$$

To estimate $B(n)$, we use the Master Theorem. In this recurrence, we have $a = 1$, $b = 3$ and $d = 0$, so $a = b^d$. Thus the solution is $B(n) = \Theta(\log n)$.

**Example 3.** For the pseudo-code below, give the number of letters printed as a function of $n$, using the $\Theta$-notation.

```
procedure PrintCs($n$)
    if $n \ge 3$ then
        for $j \leftarrow 1$ to $4n$
            do print("C")
        PrintCs($n/3$)
        PrintCs($n/3$)
```

For any $n \geq 3$, we print $4n$ "C"s and we make two recursive calls, each with parameter $n/3$. So the number of letters printed satisfies the following recurrence equation:

$$C(n) = 2C(n/3) + 4n$$

To estimate $C(n)$, we use the Master Theorem. In this recurrence, we have $a = 2$, $b = 3$ and $d = 1$, so $a < b^d$. Thus the solution is $C(n) = \Theta(n)$.

**Extending Master Theorem to allow rounding.** We will now show that Master Theorem still holds if the values of $n$ in the recursive calls are rounded, either up or down. We will establish this in two lemmas.

The first lemma shows that for any function that arises in Master Theorem its asymptotic value does not change if we round $n$ to the nearest power of $b$, either down or up. It will help to introduce some notation: For any positive integer $n$, let $n_- = b^{\lfloor \log_b n \rfloor}$ be the largest power of $b$ that is at most $n$, and $n_+ = b^{\lceil \log_b n \rceil}$ be the smallest power of $b$ that is at least $n$.

**Lemma 1** *Let $f(n) = \Theta(n^p \log^q n)$, for some integers $p, q \geq 0$. Then (a) $f(n) = \Theta(f(n_-))$, and (b) $f(n) = \Theta(f(n_+))$.*

*Proof:* We will only show (b), as the proof of (a) is essentially the same. To prove (a) we need to show that $f(n) = O(f(n_+))$ and that $f(n) = \Omega(f(n_+))$.

By the assumption about $f(n)$, we have that

$$A_1 \cdot n^p \log^q n \leq f(n) \leq A_2 \cdot n^p \log^q n \tag{1}$$

for $n \geq n_0$, where $A_1$, $A_2$ and $n_0$ are some constants.

We show that $f(n) = O(f(n_+))$ first. Using the monotonicity of function $n^p \log^q n$ and inequality $n \leq n_+$, for any given $n \geq n_2$ we have

$$f(n) \leq A_2 \cdot n^p \log^q n \leq A_2 \cdot n_+^p \log^q n_+ = A_2 \cdot f(n_+),$$

so $f(n) = O(f(n_+))$.

We now give the second estimate, which is only a slightly more subtle. Using the monotonicity of function $n^p \log^q n$ and inequality $n \geq n_+/b$, we have

$$f(n) \geq A_1 \cdot n^p \log^q n \geq A_1 \cdot (n_+/b)^p \log^q(n_+/b) = A_1/b^p \cdot n_+^p(\log^q n_+ - \log^q b) \geq A_1' \cdot n_+^p \log^q n_+ = A_1' \cdot f(n_+),$$

for $n \geq n_0'$, where $A_1'$ and $n_0'$ are sufficiently large constants. So $f(n) = \Omega(f(n_+))$. $\square$

Rather than considering separately cases when we round down or up in Master Theorem, we will consider a more general statement that encompasses both cases. Let $a$ be the parameter from Master Theorem, and suppose that $a = a' + a''$, where $a', a''$ are non-negative integers. Let $T(n)$ satisfy recurrence:

$$T(n) = a'T(\lfloor n/b \rfloor) + a''T(\lceil n/b \rceil) + cn^d.$$

Note that if $n$ is a multiple of $b$ then this recurrence is exactly the same as in the statement of Master Theorem above. Therefore the solution of this recurrence for $n_-$ and $n_+$ is as stated in Master Theorem. We That this solution is also valid for any $n$ will now follow from Lemma 1 and Lemma 2 below, since if $f(n)$ is the solution in Master Theorem for the recurrence for $T(n)$, then these two lemmas give us that $T(n) \leq T(n_+) = f(n_+) = O(f(n))$ and $T(n) \geq T(n_-) = f(n_-) = \Omega(f(n))$. Thus $T(n) = \Theta(f(n))$.

**Lemma 2** *Let $T(n)$ satisfy the above recurrence. Then for each $n$ we have $T(n_-) \leq T(n) \leq T(n_+)$.*

*Proof:* As $n_- \leq n \leq n_+$, we just need to prove that $T(n)$ is a monotone function. In other words, we need to show that $T(n) \leq T(n+1)$ for any $n\ ge 1$. But this follows directly from the monotonicity of rounding and function $cn^d$. (A formal argument would involve induction with respect to the number of times $n$ gets divided by $b$ in the recurrence, that is with respect to $\lfloor \log_b n \rfloor$.) $\square$