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

$$T(n) = 2T(n/2) + n = 2[2T(n/4) + n/2] + n = 4T(n/4) + 2n.$$ 

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

$$T(n) = 2T(n/2) + n = 4T(n/4) + 2n = 8T(n/8) + 3n = 2^jT(n/2^j) + jn = nT(1) + n \log n = \Theta(n \log n).$$

**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 \(T(1) = 1\). (This does not matter for the asymptotic solution.)

Again, we do repeated substitutions:

\[
T(n) = aT(n/b) + n
\]
\[
= a[aT(n/b^2) + n/b] + n
\]
\[
= a^2T(n/b^2) + (a/b)n + n
\]
\[
\ldots
\]
\[
= a^jT(n/b^j) + n[(a/b)^{j-1} + \ldots + (a/b)^2 + (a/b) + 1]
\]
\[
\ldots
\]
\[
= a^{\log_b n}T(1) + n \cdot \sum_{i=0}^{\log_b n-1} (a/b)^i
\]
\[
= n^{\log_b a} + n \cdot \sum_{i=0}^{\log_b n-1} (a/b)^i
\]

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

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

So

\[
T(n) = n^{\log_b a} + \frac{b}{a-b} (n^{\log_b a} - n) = \Theta(n^{\log_b a}).
\]

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 \(c\). 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

\[
T(n) = \begin{cases} 
\Theta(n^{\log_b a}) & \text{for } a > b^d \\
\Theta(n^d \log n) & \text{for } a = b^d \\
\Theta(n^d) & \text{for } a < b^d 
\end{cases}
\]
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 \( n \leq 2 \) 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 \( n \geq 2 \), 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 \( A(n) = \Theta(n^{\log_2 5}) \).

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 \( n \geq 3 \) then
        print("B")
        PrintBs(n/3)

For any \( n \geq 3 \), 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 \( n \geq 3 \) 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 $C(n) = \Theta(n)$. 