Chapter 04: Recurrences (Divide and Conquer).

The MergeSort algorithm.

\[
\text{Merge}(A, p, q, r) \begin{cases} 
  (2) & n_1 = q - p + 1; n_2 = r - q; \\
  (n_1 + 1) & \text{for } i = 1 \text{ to } n_1 \\
  (n_1) & L[i] = A[p + i - 1]; \\
  (n_2 + 1) & \text{for } j = 1 \text{ to } n_2 \\
  (n_2) & R[j] = A[q + j]; \\
  (2) & L[n_1 + 1] = \infty; R[n_2 + 1] = \infty; \\
  (2) & i = 1; j = 1; \\
  (r - p + 2) & \text{for } k = p \text{ to } r \\
  (3(r - p + 1)) & \text{if } L[i] \leq R[j] \{ \\
  & A[k] = L[i]; \\
  & i + +; \\
  \} \\
  \text{else} \{ \\
  & A[k] = R[j]; \\
  & j + +; \\
  \} 
\end{cases}
\]

\[
\text{MergeSort}(A, p, r) \begin{cases} 
  (1) & \text{if } p < r \{ \\
  (1) & q = \lfloor (p + r)/2 \rfloor; \\
  (1) & \text{MergeSort}(A(p, q)); \\
  (1) & \text{MergeSort}(A(q + 1, r)); \\
  (1) & \text{Merge}(A, p, q, r); \\
  \} 
\end{cases}
\]

The Merge problem size is \( n' = n_1 + n_2 = r - p + 1 \). The Merge count, \( T_{\text{merge}}(n') \), and the MergeSort count, \( T(n) \), are then

\[
T_{\text{merge}}(n') = 2n_1 + 2n_2 + 8 + (n' + 1) + 3n' = 6n' + 9 \\
T(n) = \begin{cases} 
  5 + 2T \left( \frac{n}{2} \right) + 6n + 9 = 2T \left( \frac{n}{2} \right) + 6n + 14, & n > 1 \\
  1, & n = 1, 
\end{cases}
\]

The base case \( n = 1 \) follows because only the “if \( p < r \)” test in the driver is executed when \( n - 1 \).
\[ T_{\text{merge}}(n') = 2n_1 + 2n_2 + 8 + (n' + 1) + 3n' = 6n' + 9 \]
\[ T(n) = \begin{cases} 
5 + 2T\left(\frac{n}{2}\right) + 6n + 9 = 2T\left(\frac{n}{2}\right) + 6n + 14, & n > 1 \\
1, & n = 1,
\end{cases} \]

**Note:** As stated, the expression for \( T(n) \) is approximate because the size of the two recursive calls to MergeSort may differ by one.

If \( n \) is even, then the expression is exact; both calls have arguments of size \( n/2 \). If \( n \) is odd, the first call has an argument which is one larger than the second call.
The expression for \( T(n) \) is called a recurrence.

\[
T(n) = 2T\left(\frac{n}{2}\right) + 6n + 14.
\]

As a first attempt to solve for \( T(n) \), we “unwind” the recurrence, restricting our attention to the subsequence \( n = 2^k \), for \( k = 0, 1, 2, \ldots \).

Why the restriction? Because the progression \( n \to n/2 \to n/2^2 \to \ldots \to 1 \) never encounters any fractions. Hence the recurrence is no longer approximate, provided \( n \to \infty \) along this particular sequence.

Note the pattern: For any \( 2^m > 1 \), we have

\[
T(2^m) = 2T(2^{m-1}) + 6 \cdot 2^m + 14.
\]

Starting with some \( n = 2^k \), we apply the pattern sequentially as follows.

\[
T(2^k) = 2T(2^{k-1}) + 6 \cdot 2^k + 14
\]
\[
= 2[2T(2^{k-2}) + 6 \cdot 2^{k-1} + 14] + 6 \cdot 2^k + 14 = 2^2T(2^{k-2}) + 6 \cdot 2 \cdot 2^k + 14(1 + 2)
\]
\[
= 2^2[2T(2^{k-3}) + 6 \cdot 2^{k-2} + 14] + 6 \cdot 2 \cdot 2^k + 14(1 + 2)
\]
\[
= 2^3T(2^{k-3}) + 6 \cdot 3 \cdot 2^k + 14(1 + 2 + 4)
\]
\[
= \ldots
\]
\[
= 2^kT(2^0) + 6 \cdot k \cdot 2^k + 14(1 + 2 + 4 + \ldots + 2^{k-1})
\]
\[
= 2^kT(1) + 6 \cdot k \cdot 2^k + 14 \sum_{j=0}^{k-1} 2^j = 2^kT(1) + 6 \cdot k \cdot 2^k + 14 \cdot \frac{2^k - 1}{2 - 1}
\]
\[
= [T(1) + 14]2^k + 6k2^k - 14.
\]

\( T(1) \) is a constant, \( K_1 \), and \( k = \lg n \). So, in terms of \( n \),

\[
T(n) = (K_1 + 14)n + 6n \lg n - 14
\]
\[
\frac{T(n)}{n \lg n} = 6 + \frac{K_1 + 14}{\lg n} - \frac{14}{n \lg n} \to 6
\]
\[
5 \leq \frac{T(n)}{n \lg n} \leq 7, \text{ for } n = 2^k \text{ sufficiently large.}
\]
So, \( T(n) \in \Theta(n \lg n) \), provided \( n \) grows through the restricted sequence 1, 2, 4, 8, \ldots. Can we extend the result to the conventional sequence \( n = 1, 2, 3, \ldots \)?

If we maintain separate terms for the recursive calls, we have

\[
T(n) = T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(n - \left\lceil \frac{n}{2} \right\rceil\right) + 6n + 14
\]

Continuing with \( T(1) = K_1 \), we compute \( T(n) \) for some initial values.

\[
\begin{align*}
T(1) &= K_1 \\
T(2) &= T(1) + T(1) + 6(2) + 14 = 2K_1 + 26 \\
T(3) &= T(2) + T(1) + 6(3) + 14 = K_1 + 2K_1 + 26 + 32 = 3K_1 + 58 \\
T(4) &= T(2) + T(2) + 6(4) + 14 = 4K_1 + 52 + 38 = 4K_1 + 90.
\end{align*}
\]

We observe that \( T(n) \) is monotonically increasing, and we could rigorously establish this result by induction.
Now consider an arbitrary \( n \), situated in an interval \([2^k, 2^{k+1})\) for which the above partial result holds. That is,

\[
5 \leq \frac{T(2^k)}{2^k \lg 2^k} = \frac{T(2^k)}{2^k \cdot k} \leq 7
\]

\[
5 \leq \frac{T(2^{k+1})}{2^{k+1} \lg 2^{k+1}} = \frac{T(2^{k+1})}{2^{k+1} \cdot (k + 1)} \leq 7.
\]

Starting with the monotone property of \( T \), we reason that \( T(n) \leq T(2^{k+1}) \), for \( n \in [2^k, 2^{k+1}) \), and continue as follows.

\[
T(n) \leq T(2^{k+1}) \leq 7 \cdot [2^{k+1} \cdot (k + 1)] \leq 7 \cdot [2^{k+1}(2k)] = 7 \cdot 4 \cdot k2^k
\]

\[
= 28 \left[ x \lg x \right]_{x=2^k} \leq 28 \left[ x \lg x \right]_{x=n} = 28n \lg n,
\]

since \( x \lg x \) is also monotone increasing. So, \( T(n) \in O(n \lg n) \) without restriction.

For a lower bound,

\[
T(n) \geq T(2^k) \geq 5 \cdot [2^k \cdot k] \geq \frac{5}{2} \cdot 2^{k+1} \cdot k \geq \frac{5}{2} \cdot 2^{k+1} \cdot \frac{k + 1}{2}
\]

\[
= \frac{5}{4} \cdot (k + 1)2^{k+1} = \frac{5}{4} \left[ x \lg x \right]_{x=2^{k+1}} \geq \frac{5}{4} \left[ x \lg x \right]_{x=n} = \frac{5}{4} n \lg n.
\]

Thus, we have

\[
\frac{5}{4} \leq \frac{T(n)}{n \lg n} \leq 28, \text{ for all sufficiently large } n.
\]

So, \( T(n) \in \Omega(n \lg n) \). Together with the upper bound above, \( T(n) \in \Theta(n \lg n) \) without restriction.
Master Template for recurrences

Suppose $T(n) > 0$ is a function satisfying the following recurrence, and suppose $f(n) \geq 0$.

$$
T(n) = \begin{cases} 
aT\left(\frac{n}{b}\right) + f(n), & \text{if } n \geq b \\
T(n) \leq K, & \text{for } 1 \leq n < b,
\end{cases}
$$

where $a \geq 1, b > 1$, and $n/b$ can be either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then each of the following three cases provides an immediate solution for the recurrence.

(a) **weak glue**: If $\exists \epsilon > 0$ with $f(n) = O(n^{\log_b a - \epsilon})$, then $T(n) = \Theta(n^{\log_b a})$.

(b) **medium glue**: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \lg n)$.

(c) **strong glue**: If $\exists \epsilon > 0$ with $f(n) = \Omega(n^{\log_b a + \epsilon})$ and

- $\exists c$ with $0 < c < 1, af(n/b) \leq cf(n)$ for all sufficiently large $n$,
- then $T(n) = \Theta(f(n))$.

Observations.

1. We will call $f(n)$ the “glue function” as it represents the work required to prepare for the recursive calls and to assemble their returned results into a solution to the overall problem.

2. We will call the polynomial $g(n) = n^{\log_b a}$ the “reference function.” It represents the work done in establishing the tree of recursive calls. In the context of algorithm analysis, the recurrence represents a recursive algorithm that divides its input into $b$ parts and recursively considers $a$ of them.

3. The variations $g_-(n) = n^{(\log_b a) - \epsilon}$ and $g_+(n) = n^{(\log_b a) + \epsilon}$ are used in cases (a) and (c), respectively, of the template. We refer to these variations as “exponentially reduced” and “exponentially enhanced” versions of the reference.
Example.

For MergeSort: $T(n) = 2T(n/2) + f(n)$, where $f \in \Theta(n)$. This glue function then satisfies $K_1 n \leq f(n) \leq K_2 n$ for all sufficiently large $n$.

The reference function is $g(n) = n^{\log_2 2} = n^1 = n$.

Case (a) of the master template reads:

**weak glue:** If $\exists \epsilon > 0$ with $f(n) = O(n^{[\log_b a] - \epsilon})$, then $T(n) = \Theta(n^{\log_b a})$.

To check out case (a), let $g_-(n) = n^{1-\epsilon}$ be an exponentially reduced version of the reference.

For large $n$, we have

$$\frac{f(n)}{g_-(n)} \geq \frac{K_1 n}{n^{1-\epsilon}} = K_1 n^{\epsilon} \to \infty,$$

which implies $f(n) \neq O(n^{1-\epsilon})$ for any $\epsilon > 0$. Case (a) fails.

Case (b) of the master template reads:

**medium glue:** If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \lg n)$.

To check out case (b), we investigate the ratio of the glue function to the reference function:

$$0 < K_1 \leq \frac{K_1 n}{n} \leq \frac{f(n)}{g(n)} \leq \frac{K_2 n}{n} = K_2 < \infty,$$

which implies $f \in \Theta(g)$. Case (b) succeeds, and therefore $T(n) = \Theta(n \lg n)$. 

7
Case (c) of the master template reads:

**strong glue:** If \( \exists \epsilon > 0 \) with \( f(n) = \Omega(n^{[\log_b a]+\epsilon}) \) and 
\( \exists c \) with \( 0 < c < 1 \), \( af(n/b) \leq cf(n) \) for all sufficiently large \( n \),
then \( T(n) = \Theta(f(n)) \).

In the problem at hand, if \( g_+ = n^{1+\epsilon} \) any exponentially enhanced reference, we have

\[
\frac{f(n)}{g_+(n)} \leq \frac{K_2 n}{n^{1+\epsilon}} = \frac{K_2}{n^\epsilon} \to 0.
\]

For any \( \epsilon > 0 \), there is no possible positive lower bound under \( f/g_+ \). Therefore \( f \not\in \Omega(g_+) \) and case (c) fails.

In general, you can check out the cases in the master template in any order. If you find one that works, then you can ignore all of the others.

The homework asks you to prove that the three cases of the master template are mutually exclusive. That is, if one of the cases succeeds, the other two must fail.
Example. Suppose

\[ T(n) = 2T(n/3) + \sqrt{n}. \]

We have \( f(n) = n^{1/2} \), and \( g(n) = n^{\log_3 2}. \)

Note that \( 0.5 = \log_3 3^{1/2} = \log_3 1.732 < \log_3 2. \)

Choose \( \epsilon \) as shown above. Then

\[ \frac{f(n)}{g(n)} = \frac{n^{1/2}}{n^{(\log_3 2)-\epsilon}} \to 0. \]

Thus \( f \in o(g) \), which implies \( f \in O(g) \), which enables case (a), which gives \( T(n) = \Theta(n^{\log_3 2}). \)
Example. $T(n) = 49T(n/25) + n^{3/2} \lg n$. Here $a = 49$, $b = 25$, so $\log_{25} 49 = 1.209$. The reference is $g(n) = n^{1.209}$. The glue is $f(n) = n^{1.5} \lg n$.

$$\frac{f(n)}{g_+(n)} = \frac{n^{1.5} \log_2 n}{n^{1.209+\epsilon}} = n^{0.291-\epsilon} \log_2 n \to \infty$$

as $n \to \infty$ for any $0 < \epsilon < 0.291$. That is, $f(n) = \omega(n^{1.209+\epsilon})$, which implies $f(n) = \Omega(n^{1.209+\epsilon})$. This leads to a possible application of case (c) of the template.

There is, however, a regularity condition that must be satisfied: $af(n/b) \leq cf(n)$ for some constant $0 < c < 1$ and all sufficiently large $n$. In the problem at hand,

$$af(n/b) = 49f(n/25) = 49 \cdot \left(\frac{n}{25}\right)^{3/2} \cdot \lg \frac{n}{25}$$

$$= \frac{49n^{3/2}}{(25)^{3/2}} \cdot [(\lg n) - (\lg 25)] \leq \frac{49}{125} n^{3/2} \lg n = \frac{49}{125} f(n).$$

Consequently, the value $c = 49/125$ satisfies the regularity condition. Then $T(n) = \Theta(f(n)) = \Theta(n^{3/2} \lg n)$ via case (c) of the template.
Maximum subarray problem: given a vector of real numbers, find a contiguous segment with the largest sum. Output the lower and upper indices of the chosen segment and the resulting optimal sum.

Observations:

1. If the array contains only nonnegative numbers, choose the entire array.
2. If the array contains only negative numbers, choose the negative number closest to zero in an array of length 1.
3. If the array contains both positive and negative numbers, divide it in half. The optimal segment is in the lower array, in the upper array, or it crosses the boundary between the two.

\[(\text{low}, \text{high}, \text{sum}) = \text{MaxSubArray}(A, 1, \text{length}(A));\]

MaxSubArray(A, low, high) {
    if (low = high)
        return (low, low, A[low]);
    mid = (low + high)/2;
    (leftLow, leftHigh, leftSum) = MaxSubArray(A, low, mid);
    (rightLow, rightHigh, rightSum) = MaxSubArray(A, mid + 1, high);
    (crossLow, crossRight, crossSum) = CrossSubArray(A, low, mid, high);
    if (leftSum \geq rightSum) and (leftSum \geq crossSum)
        return (leftLow, leftHigh, leftSum);
    if (rightSum \geq leftSum) and (rightSum \geq crossSum)
        return (rightLow, rightHigh, rightSum);
    else
        return (crossLow, crossHigh, crossSum);
}

minimum count = 2.

maximum count = 8 + work in CrossSubArray + work in two recursive calls.
CrossSubArray(A, low, mid, high) {
    leftSum = −∞;
    sum = 0;
    for k = mid downto low {
        sum = sum + A[k];
        if (sum > leftSum) {
            leftSum = sum;
            maxLeft = k;
        }
    }
    rightSum = −∞;
    sum = 0;
    for k = mid + 1 to high {
        sum = sum + A[k];
        if (sum > rightSum) {
            rightSum = sum;
            maxRight = k;
        }
    }
    return (maxLeft, maxRight, leftSum + rightSum);
}

count = 7 + (3 to 5) · n’, where n’ = high − low +1 is the length of the subproblem under analysis.

\[
T(n’) = 8 + 7 + (3 \text{ to } 5)n’ + 2T(n’/2) = 2T(n’/2) + (3 \text{ to } 5)n + 15 = 2T(n’/2) + f(n)
\]
\[
T(n) = \begin{cases} 
2T(n/2) + f(n), & n > 1 \\
2, & n = 1
\end{cases}
\]
Check the master template:

Suppose \( T(n) > 0 \) is a nondecreasing function satisfying the following recurrence, and suppose \( f(n) \geq 0 \). Also,

\[
T(n) = aT\left(\frac{n}{b}\right) + f(n), \quad \text{if } n \geq b
\]
\[
T(n) \leq K, \quad \text{for } 1 \leq n < b,
\]

where \( a \geq 1, b > 1, \) and \( n/b \) can be either \( \lfloor n/b \rfloor \) or \( \lceil n/b \rceil \). Then each of the following three cases provides an immediate solution for the recurrence.

(a) If \( \exists \epsilon > 0 \) with \( f(n) = O(n^{\log_b a - \epsilon}) \), then \( T(n) = \Theta(n^{\log_b a}) \).

(b) If \( f(n) = \Theta(n^{\log_b a}) \), then \( T(n) = \Theta(n^{\log_b a} \cdot \lg n) \).

(c) If \( \exists \epsilon > 0 \) with \( f(n) = \Omega(n^{\log_b a + \epsilon}) \) and \( \exists c \) with \( 0 < c < 1, af(n/b) \leq cf(n) \) for all sufficiently large \( n \), then \( T(n) = \Theta(f(n)) \).

For MaximumSubarray: \( T(n) = 2T(n/2) + f(n) \), with \( f(n) \in \Theta(n) \); reference function is \( g(n) = n^{\log_2 2} = n \).

So, \( f \in \Theta(g) \), which enables case (b) of the Master Template. \( T(n) = \Theta(n \lg n) \).
Matrix multiplication: Given two \( n \times n \) matrices, \([a_{ij}]\) and \([b_{ij}]\), their product is \([c_{ij}]\), where

\[
c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}.
\]

MatrixMultiply(a, b) {
for \( i = 1 \) to \( n \) \( n + 1 \)
for \( j = 1 \) to \( n \) \( n(n + 1) \)
for \( k = 1 \) to \( n \) \( n^2(n + 1) \)
\( c[i][j] = a[i][k] \ast b[k][j] \);
\( n^3 \)
}

Have \( T(n) = (n + 1) + n(n + 1) + n^2(n + 1) + n^3 = 2n^3 + 2n^2 + 2n + 1 = \Theta(n^3) \).

Recursive approach: for \( n = 2^k \), divide factors into four pieces.

\[
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix} = \begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix} \cdot \begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
\]

\[
C_{11} = A_{11}B_{11} + A_{12}B_{21} \\
C_{12} = A_{11}B_{12} + A_{12}B_{22} \\
C_{21} = A_{21}B_{11} + A_{22}B_{21} \\
C_{22} = A_{21}B_{12} + A_{22}B_{22}
\]

But, can isolate components by passing row and column boundaries to recursive calls, and merge results of recursive calls by doing \( 4(n/2)(n/2) = n^2 \) additions.

Then, \( T(n) = 8T(n/2) + n^2 \). Glue is \( n^2 \); reference is \( n^{\log_3 8} = n^3 \).

Then \( n^2/n^3 = 1/n \rightarrow 0 \), implying \( n^2 = o(n^3) \), which in turn implies \( n^2 = O(n^3) \), which enables case (a). Therefore \( T(n) = \Theta(n^3) \).
1. Notice that addition is much less expensive than multiplication. Given two \( n \times n \) matrices, \([a_{ij}]\) and \([b_{ij}]\), their sum is \([c_{ij}]\), where

\[
c_{ij} = a_{ij} + b_{ij}.
\]

MatrixAdd(a, b) {
    for \( i = 1 \) to \( n \) \( n+1 \)
    for \( j = 1 \) to \( n \) \( n+1 \)
        \( c[i][j] = a[i][j] + b[i][j]; \)
    }

\[
T(n) = (n + 1) + n(n + 1) + n^2 = 2n^2 + 2n + 1 = \Theta(n^2).
\]

2. Very clever observation for case of \( 2 \times 2 \) matrices: 8 multiplications, 4 additions.

\[
A = \begin{bmatrix}
a & b \\
c & d
\end{bmatrix} \quad B = \begin{bmatrix}
e & f \\
g & h
\end{bmatrix} \quad AB = \begin{bmatrix}
ae + bg & af + bh \\
ce + dg & cf + dh
\end{bmatrix}
\]

becomes 7 multiplication and (about) 18 additions/subtractions.

\[
AB = \begin{bmatrix}
-(a + b)h + d(g - e) + (a + d)(e + h) & a(f - h) + (a + b)h \\
+(b - d)(g + h) & a(f - h) - (c + d)e + (a + d)(e + h) \\
(c + d)e + d(g - e) & -(a - c)((e + f)
\end{bmatrix}
\]

7 multiplications are:

\[
(a + b)h \\
d(g - e) \\
(a + d)(e + h) \\
(b - d)(g + h) \\
a(f - h) \\
(c + d)e \\
(a - c)(e + f)
\]
Strassen’s Method, for \( n = 2^k \).

Step 1: Divide the input matrix into four \((n/2) \times (n/2)\) matrices as above. \(\Theta(1)\), that is, constant time via index calculations.

Step 2: Compute 10 matrices \(S_1, \ldots, S_{10}\), each \((n/2) \times (n/2)\), in \(\Theta(n^2)\) time. These correspond to the factors in the 7 multiplications above:

\[
(a + b), (g - e), (a + d), (e + h), (b - d), (g + h), (f - h), (c + d), (a - c), (e + f)
\]

Step 3: Recursively compute 7 matrix products: \(P_1, \ldots, P_7\), each of size \((n/2) \times (n/2)\). These correspond to the 7 multiplications themselves:

\[
(a + b)h, d(g - e), (a + d)(e + h), (b - d)(g + h), a(f - h), (c + d)e, (a - c)(e + f)
\]

Step 4: Create the components \(C_{11}, C_{12}, C_{21}, C_{22}\) by adding and subtracting various combinations of the matrix products in \(\Theta(n^2)\) time.

\[
T(n) = 7T(n/2) + \Theta(n^2)
\]

Reference function is \(n^{\log_2 7} = n^{2.8074}\). Glue function is \(f(n) = \Theta(n^2)\).

\[
\frac{f(n)}{n^{2.8074-\epsilon}} \leq \frac{K_2n^2}{n^{2.8074-\epsilon}} = \frac{K_2}{n^{0.8074-\epsilon}} \to 0, \text{ for } 0 < \epsilon < 0.8074.
\]

\[
f(n) = o(n^{2.8074-\epsilon}), \text{ which implies } f(n) = O(n^{2.8074-\epsilon}), \text{ which enables case(a)}.
\]

\[
T(n) = \Theta(n^{2.8074}).
\]

Observation: For matrices which are not of size equal to a power of 2, pad with zeros to the next power of two. Desired product appears in the upper left corner of the larger product.
Alternatives to the master template.

(a) Induction (also called back-substitution)

Step 1: Guess the form of the solution, normally by examining small cases.
Step 2: Prove the form correct via induction.

Example:

\[
T(n) = \begin{cases} 
2T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + n, & n \geq 2 \\
1, & n = 1 
\end{cases}
\]

To facilitate our guess, consider just \( n = 2^k \), rather than a general integer \( n \).
In this case, the round-down operation has no effect.

\[
\begin{align*}
T(1) &= 1 \\
T(2) &= 2T(1) + 2 = 4 = 2^1(1 + 1) \\
T(4) &= 2T(2) + 4 = 12 = 2^2(2 + 1) \\
T(8) &= 2T(4) + 8 = 32 = 2^3(3 + 1) \\
T(16) &= 2T(8) + 16 = 80 = 2^4(4 + 1)
\end{align*}
\]

It appears \( T(2^k) = 2^k(k+1) \). In terms of \( n = 2^k \), we have \( T(n) = n(1 + \lg n) = n \lg(2n) \) when \( n \) is a power of 2.

We now guess that \( T(n) \leq cn \lg(2n) \), for \( n \geq 2 \) (to avoid a right-hand-side zero) and an appropriately chosen \( c > 1 \).

We start an induction proof by working out the first few cases, say \( n = 2, 3, 4 \), and choosing a \( c > 1 \) that works for all of those cases.

Then, for larger \( n \), we argue

\[
T(n) = 2T(\lfloor n/2 \rfloor) + n \leq 2c \lfloor n/2 \rfloor \lg(2 \lfloor n/2 \rfloor) + n \leq 2c \left( \frac{n}{2} \right) \lg \left( 2 \cdot \frac{n}{2} \right) + n
\]

\[
= cn[(\lg 2n) - 1] + n = cn \lg(2n) + n(1 - c) \leq cn \lg(2n),
\]

the last since \( 1 - c < 0 \).

Also, since \( n \lg(2n) = n(1 + \lg n) \leq 2n \lg n \), for \( n \geq 2 \), we have \( T(n) \leq 2cn \lg n \).
We conclude \( T(n) = O(n \lg n) \).
Consider the recurrence: $T(n) = 3T(\lfloor n/4 \rfloor) + n^2$. Our goal is to obtain a general form for the solution that we can verify via induction as in part (a) above. Working with the sequence $n = 4^k$ for $k = 0, 1, \ldots$ is convenient because the round-down operation does not introduce any fractions. However, instead of developing the recurrence algebraically, we draw a tree that illustrates the various subproblems considered.

Each node makes three recursive calls to its children. The recursive calls themselves constitute a constant amount of work — establishing the stack frame and copying argument pointers — that does not grow with the size of the problem. However, each node expends $m^2$ work, where $m$ is the size of the node’s input, in processing the recursive returns into a package to return to its caller. This latter processing does grow with the size of the problem.

In the tree above, the right edge totals the processing associated with the nodes on a given level. To obtain a grand total over all nodes, we sum the sub-totals associated with each level:

$$
T(n) = \sum_{j=0}^{\log_4 n} 3^j \left( \frac{n}{4^j} \right)^2 = n^2 \sum_{j=0}^{\log_4 n} \left( \frac{3}{16} \right)^j \leq n^2 \sum_{j=0}^{\infty} \left( \frac{3}{16} \right)^j = \frac{n^2}{1 - 3/16} = \frac{16}{13} \cdot n^2 = \Theta(n^2).
$$

We can then conjecture $T(n) \leq cn^2$ for some appropriate constant $c$ and continue with a proof by induction.
The master template for recurrences.

1. Statement of the theorem.

Suppose $T(n) > 0$ is a nondecreasing function satisfying the following recurrence, and suppose $f(n) \geq 0$. Also,

$$
T(n) = aT\left(\frac{n}{b}\right) + f(n), \quad \text{if } n \geq b \\
T(n) \leq K, \quad \text{for } 1 \leq n < b,
$$

where $a \geq 1, b > 1$, and $n/b$ can be either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then each of the following three cases provides an immediate solution for the recurrence.

(a) If $\exists \epsilon > 0$ with $f(n) = O(n^{\log_b a - \epsilon})$, then $T(n) = \Theta(n^{\log_b a})$.

(b) If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \lg n)$.

(c) If $\exists \epsilon > 0$ with $f(n) = \Omega(n^{\log_b a + \epsilon})$ and $\exists c$ with $0 < c < 1$, $af(n/b) \leq cf(n)$ for all sufficiently large $n$, then $T(n) = \Theta(f(n))$. 

2. **Proof of Case (a):** If $\exists \epsilon > 0$ with $f(n) = O(n^{\log_b a - \epsilon})$, then $T(n) = \Theta(n^{\log_b a})$.

We are interested in the asymptotic growth of $T(n)$ as $n \to \infty$. To this end, we first consider a special case where $n$ approaches $\infty$ through a constrained sequence: $n = b^k$, for $k = 0, 1, 2, \ldots$. Along this sequence $k = \log_b n$ and the general recurrence pattern is, for any exponent $t > 0$,

$$T(b^t) = aT(b^{t-1}) + f(b^t),$$

while the base case corresponds to $t = 0$: $T(b^0) = T(1)$, some constant.

We expand the recurrence, starting with the generic $n = b^k$.

$$T(b^k) = aT(b^{k-1}) + f(b^k)$$
$$= a \left[ aT(b^{k-2}) + f(b^{k-1}) \right] + f(b^k) = a^2T(b^{k-2}) + af(b^{k-1}) + f(b^k)$$
$$= a^2 \left[ aT(b^{k-3}) + f(b^{k-2}) \right] + af(b^{k-1}) + f(b^k)$$
$$= a^3T(b^{k-3}) + a^2f(b^{k-2}) + af(b^{k-1}) + f(b^k)$$
$$\vdots$$
$$= a^kT(b^0) + a^{k-1}f(b^1) + a^{k-2}f(b^2) + \ldots + a^0 f(b^k)$$

That is,

$$T(b^k) = a^kT(1) + \sum_{j=0}^{k-1} a^j f(b^{k-j}) = a^kT(1) + \sum_{j=1}^{k} a^{k-j} f(b^j). \quad (1)$$
Since $f$ is a positive function, we immediately have

$$T(n) \geq T(1) \cdot a^k = T(1) \cdot a^{\log_b n}.$$ 

Considering the following useful rearrangement,

$$n = b^{\log_b n}, \quad n^{\log_b a} = (b^{\log_b n})^{\log_b a} = b^{(\log_b n)(\log_b a)} = (b^{\log_b a})^{\log_b n} = a^{\log_b n}, \quad (2)$$

we now have $T(n) \geq T(1) \cdot n^{\log_b a}$ and therefore $T(n) = \Omega(n^{\log_b a})$ along the constrained sequence $n = b^k$.

Note that we have **not used the hypothesis**: $\exists \epsilon > 0 \text{ with } f(n) = O(n^{[\log_b a] - \epsilon})$.

Is this partial result true for all three cases?

Yes. In case (b), the result is $T(n) = \Omega(n^{\log_b a} \cdot \lg n)$, a tighter lower bound.

In case (c), the result is $T(n) = \Omega(f) \geq \Omega(n^{(\log_b a)+\epsilon})$, a yet tighter lower bound.
Returning to Equation (1), we now pursue an upper bound. Since \( T(1) \) is just another constant, say \( K_1 \), we start with

\[
T(b^k) = K_1 a^k + \sum_{j=1}^{k} a^{k-j} f(b^j).
\]

Since we know that \( f(n) = O(n^{[\log_b a] - \epsilon}) \), we can bound the \( f(b^j) \) terms when \( j \) is large. Specifically, \( \exists k_1, K_2 \) with \( n \geq b^{k_1} \) implying \( f(n) \leq K_2 n^{[\log_b a] - \epsilon} \). To use this constraint, we break the sum above into two components:

\[
T(b^k) = K_1 a^k + \sum_{j=1}^{k_1} a^{k-j} f(b^j) + \sum_{j=k_1+1}^{k} a^{k-j} f(b^j),
\]

for \( k \geq k_1 \).
Now let $K_3 = \max_{1 \leq j \leq k_1} f(b^j)$, a constant that allows us to combine the first sum with the $K_1a^k$ term. Specifically, since $a \geq 1$, we have

$$T(b^k) \leq K_1a^k + K_3\sum_{j=1}^{k_1} a^{k-j} + \sum_{j=k_1+1}^{k} a^{k-j} f(b^j)$$

$$\leq K_1a^k + K_3a^k \sum_{j=1}^{k_1} a^{-j} + \sum_{j=k_1+1}^{k} a^{k-j} f(b^j)$$

$$\leq K_1a^k + K_3a^k \sum_{j=1}^{k_1} (1) + \sum_{j=k_1+1}^{k} a^{k-j} f(b^j)$$

$$= (K_1 + k_1K_3)a^k + \sum_{j=k_1+1}^{k} a^{k-j} f(b^j).$$

In the remaining sum, $j > k_1$, which implies $b^j > b^{k_1}$ and therefore

$$f(b^j) \leq K_2 (b^j)^{\log_b a - \epsilon} = K_2 b^{-j\epsilon} \cdot b^{j\log_b a} = K_2 \left( \frac{1}{b^\epsilon} \right)^j \cdot (b^{\log_b a})^j = K_2 \left( \frac{1}{b^\epsilon} \right)^j \cdot a^j$$

$$T(b^k) \leq (K_1 + k_1K_3)a^k + \sum_{j=k_1+1}^{k} a^{k-j} K_2a^j \left( \frac{1}{b^\epsilon} \right)^j$$

$$= (K_1 + k_1K_3)a^k + K_2a^k \sum_{j=k_1+1}^{\infty} \left( \frac{1}{b^\epsilon} \right)^j$$

$$\leq (K_1 + k_1K_3)a^k + K_2a^k \sum_{j=0}^{\infty} \left( \frac{1}{b^\epsilon} \right)^j$$

$$= (K_1 + k_1K_3)a^k + \frac{K_2a^k}{1 - (1/b^\epsilon)}.$$

The last reduction obtains because $b > 1, \epsilon > 0$, which implies that $(1/b^\epsilon) < 1$ and the infinite geometric series converges.
Therefore, we have

\[
T(b^k) \leq \left( K_1 + k_1K_3 + \frac{K_2}{1 - (1/b^\epsilon)} \right) a^k = \left( K_1 + k_1K_3 + \frac{K_2}{1 - (1/b^\epsilon)} \right) a^{\log_b n}
\]

\[
= \left( K_1 + k_1K_3 + \frac{K_2}{1 - (1/b^\epsilon)} \right) n^{\log_a a},
\]

where the last reduction uses Equation (2). We now have \( T(n) = O(n^{\log_a a}) \) along the constrained sequence \( n = b^k \). Combining this upper bound with the lower bound from above, we have \( T(n) = \Theta(n^{\log_a a}) \) along the constrained sequence \( n = b^k \).

Moreover, on this constrained sequence, any floor or ceiling functions in the recurrence have no effect. That is, the analysis to this point is valid whether the recurrence is either of the forms

\[
T(n) = aT \left( \left\lfloor \frac{n}{b} \right\rfloor \right) + f(n)
\]

\[
= aT \left( \left\lceil \frac{n}{b} \right\rceil \right) + f(n),
\]

or any combination of floor and ceiling operations on the \( a \) recursive arguments.
We now exploit the fact that $T(n)$ is nondecreasing to show that $T(n) = O(n^{\log_b a})$ without the restriction to the constrained sequence $n = b^k$. Again, the particular form of the recurrence (Equations (3)) will not enter into the analysis.

Introducing new constants, the constrained result is $\exists k_2, K_4, K_5$ such that $k \geq k_2$ implies

$$K_4 (b^k)^{\log_b a} \leq T(b^k) \leq K_5 (b^k)^{\log_b a}$$
$$K_4 (b^{\log_b a})^k \leq T(b^k) \leq K_5 (b^{\log_b a})^k$$
$$K_4 a^k \leq T(b^k) \leq K_5 a^k$$

Now, for an unconstrained $n \geq b^{k_2}$, we have, for some $j \geq k_2$, $b^j \leq n < b^{j+1}$. The nondecreasing nature of $T$ forces

$$T(n) \leq T(b^{j+1}) \leq K_5 a^{j+1} = K_5 a \cdot a^j = K_5 a \cdot (b^{\log_b a})^j = K_5 a \cdot (b^j)^{\log_b a}$$
$$= K_5 a \left[ x^{\log_b a} \right]_{x=b^j} \leq K_5 a \left[ x^{\log_b a} \right]_{x=n} = K_5 a \cdot n^{\log_b a}.$$  

That is, $T(n) = O(n^{\log_b a})$ without restriction.

Similarly,

$$T(n) \geq T(b^j) \geq K_4 a^j = \frac{K_4}{a} \cdot a^{j+1} = \frac{K_4}{a} \cdot (b^{\log_b a})^{j+1} = \frac{K_4}{a} \cdot (b^{j+1})^{\log_b a}$$
$$= \frac{K_4}{a} \left[ x^{\log_b a} \right]_{x=b^{j+1}} \geq \frac{K_4}{a} \left[ x^{\log_b a} \right]_{x=n} = \frac{K_4}{a} \cdot n^{\log_b a},$$

which implies $T(n) = \Omega(n^{\log_b a})$ without restriction. Combining the two results, we have the desired relationship: $T(n) = \Theta(n^{\log_b a})$.  
