Recurrence Relations

Last Updated: January 24th, 2026

Introduction

Determining the running time of a recursive algorithm often requires one to determine the big-O growth of a function $T(n)$ that is defined in terms of a *recurrence relation*. Recall that a **recurrence relation** for a sequence of numbers is simply an equation that provides the value of the n th number in terms of an algebraic expression involving n and one or more of the previous numbers of the sequence. The most common recurrence relation we will encounter in this course is the **uniform divide-and-conquer recurrence relation**, or **uniform recurrence** for short.

Uniform Divide-and-Conquer Recurrence Relation: one of the form

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

where $a > 0$ and $b > 1$ are integer constants. This equation is explained as follows.

$T(n)$: $T(n)$ denotes the number of steps required by some divide-and-conquer algorithm $\mathcal{A}$ on a problem instance having size n.

Divide $\mathcal{A}$ divides original problem instance into a subproblem instances.

Conquer Each subproblem instance has size n/b and hence is solved (conquered) in $T(n/b)$ steps by making a recursive call to $\mathcal{A}$.

Combine $f(n)$ represents the number of steps needed to both divide the original problem instance and combine the a solutions into a final solution for the original problem instance.

For example,

$$T(n) = 7T(n/2) + n^2,$$

is a uniform divide-and-conquer recurrence with $a = 7$, $b = 2$, and $f(n) = n^2$.

Mergesort

The **Mergesort** algorithm is a divide-and-conquer algorithm for sorting an array of size n of comparable elements. The algorithm begins by checking if input array a has $n \leq 2$ elements. If so, then a is sorted in place by making at most one swap. Otherwise, a is divided into two (almost) equal halves a_{left} and a_{right}. Both of these subarrays are sorted by making recursive calls to Mergesort. Once sorted, a `merge` operation merges the elements of a_{left} and a_{right} into an auxiliary array. This sorted auxiliary array is then copied over to the original array. Merging requires $\Theta(n)$ steps. Therfore, the uniform divide-and-conquer recurrence that represents the number of steps $T(n)$ required by the **Mergesort** algorithm is

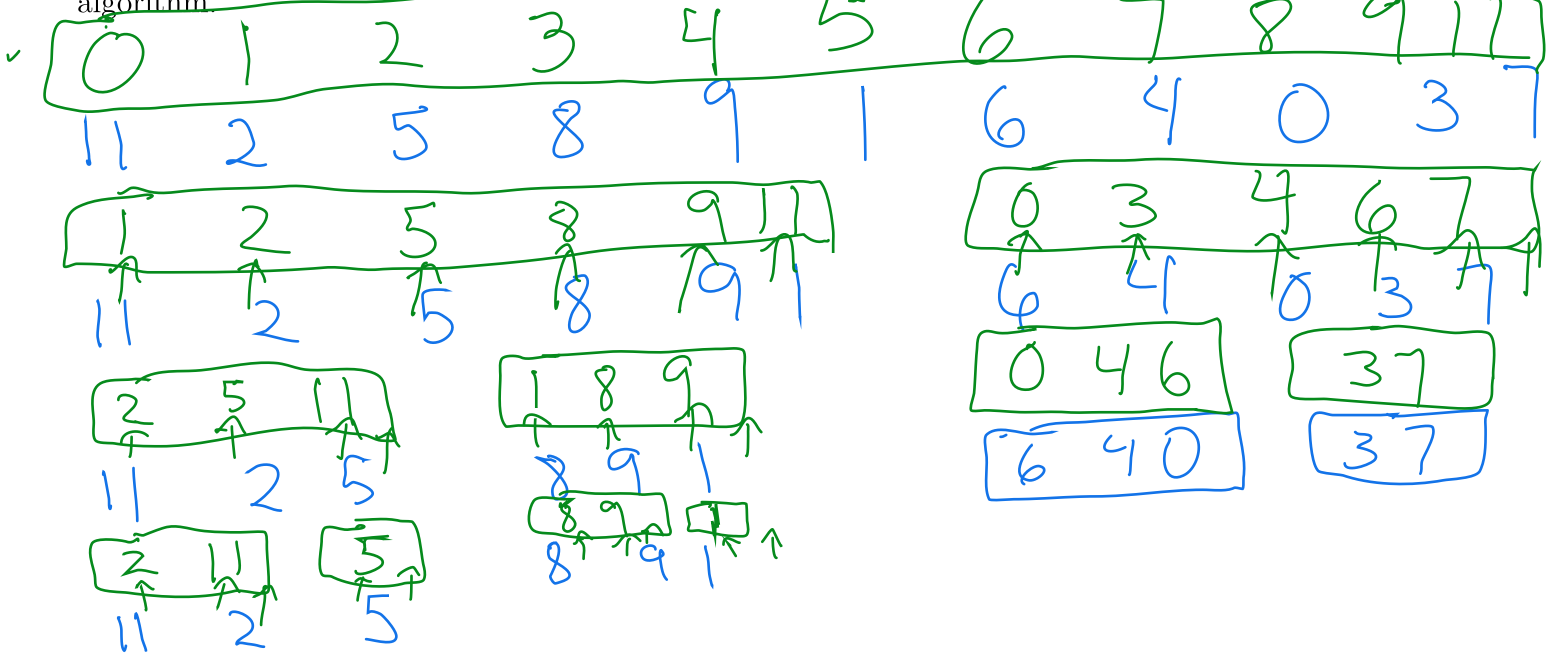
$$\boxed{T(n) = 2T(n/2) + n,}$$

and we'll see in this lecture that it yields a growth of $T(n) = \Theta(n \log n)$.

Example 1. Demonstrate the `Mergesort` divide-and-conquer algorithm on the array

$$a = 11, 2, 5, 8, 9, 1, 6, 4, 0, 3, 7$$

and provide a divide-and-conquer recurrence that describes the number of steps required by the algorithm.



It should be emphasized that not every divide-and-conquer algorithm produces a uniform divide-and-conquer recurrence. For example, the *Median-of-Five Find Statistic* algorithm described in the next lecture produces the recurrence

$$T(n) = T(n/5) + T(7n/10) + an,$$

where $a > 0$ is a constant. We refer to such recurrences as **non-uniform divide-and-conquer recurrences**. They are non-uniform in the sense that the created subproblem instances may not all have the same size.

The complexity analysis of a divide-and-conquer algorithm often reduces to determining the big-O growth of a solution $T(n)$ to a divide-and-conquer recurrence. In this lecture we examine two different ways of solving such recurrences, which are summarized as follows.

Master Theorem The **Master Theorem** provides a solution $T(n)$ to a uniform recurrence, provided a, b, and $f(n)$ satisfy certain conditions.

Substitution Method The **substitution method** uses mathematical induction to prove that some candidate function $T(n)$ is a solution to a given divide-and-conquer recurrence. The candidate solution is usually obtained by making an educated guess, or by analyzing the associated recursion tree.

The Master theorem and substitution method represent *proof methods*, meaning that the application of either method will yield a growth for $T(n)$ that is beyond doubt.

Recursion Trees and The Master Theorem

Master Theorem. Let $a \geq 1$ and $b > 1$ be constants, $f(n)$ a function, and $T(n)$ be defined on the nonnegative integers by

$$T(n) = aT(n/b) + f(n). \tag{1}$$

Then the growth of $T(n)$ can be asymptotically determined under the following assumptions.

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \cdot \log n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$, then $T(n) = \Theta(f(n))$.
4. If $f(n) = \Theta(n^{\log_b a} \log_b^r n)$, then $T(n) = \Theta(n^{\log_b a} \log_b^{r+1} n)$

We prove a relaxed version of the Master theorem by assuming that n is always a power of b, so that the resulting recursion tree associated with the recurrence in Equation 1 is a perfect a-ary tree. The more general case when n is not a power of b follows from this theorem with additional work involving the analysis of the insignificant effects that floors and ceilings have on the growth of $T(n)$.

Lemma 1. Let $a \geq 1$ and $b > 1$ be constants, and let $f(n)$ be a nonnegative function defined on exact powers of b. Define $T(n)$ on exact powers of b by the recurrence relation

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ aT(n/b) + f(n) & \text{if } n = b^i \end{cases}$$

Recursive work

for some positive integer i. Then

$$T(n) = \boxed{\Theta(n^{\log_b a})} + \boxed{\sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)}.$$

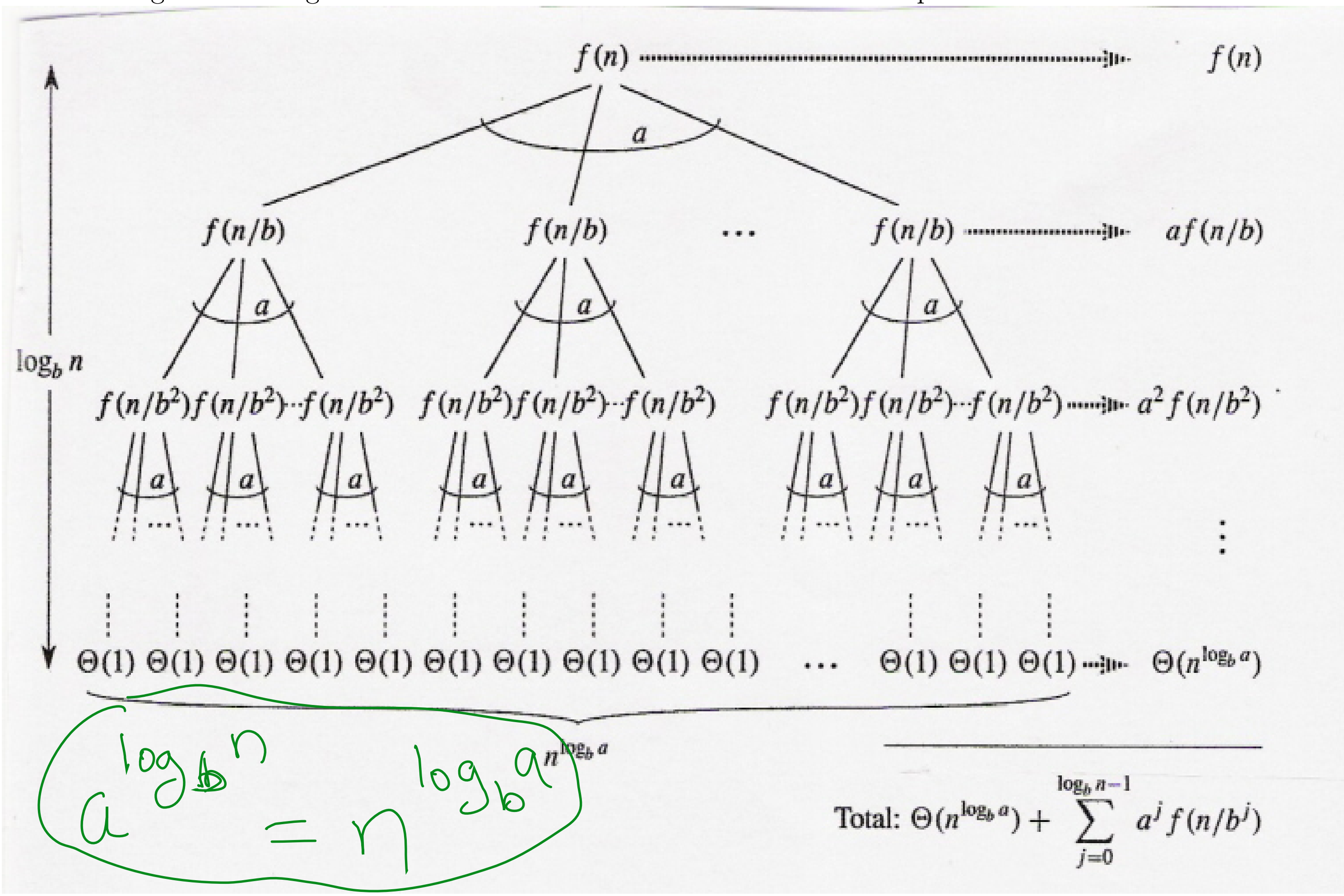
Base Case work

Proof of Lemma 1. Notice that, since n is a power of b, there are $\log_b n + 1$ levels of recursion: $0, 1, \ldots, \log_b n$. Moreover, level j, $0 \leq j \leq \log_b n - 1$ contributes a total of $a^j f(n/b^j)$ to the total value of $T(n)$, while level $\log_b n$ contributes $\Theta(1) \cdot a^{\log_b n} = \Theta(n^{log_b a})$ (see the general recursion-tree in Figure 1 below). Hence,

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j).$$

QED

Figure 1: The general recursion tree for uniform divide-and-conquer recurrences



$a^{\log_b n} = n^{\log_b a}$

Lemma 2. Let

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j),$$

where a, b, and $f(n)$ are defined as in Lemma 1. Then

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $g(n) = \mathrm{O}(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$ then $g(n) = \Theta(n^{\log_b a} \cdot \log n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant positive $c < 1$, then $g(n) = \Theta(f(n))$.
4. If $f(n) = \Theta(n^{\log_b a} \log_b^r n)$, then $g(n) = \Theta(n^{\log_b a} \log_b^{r+1} n)$

The proof of Lemmma 2 makes use of the geometric series formula

$$\sum_{j=0}^{k-1} r^j = \frac{r^k - 1}{r - 1},$$

as well as the formula

$$b^{j \log_b a} = b^{\log_b a^j} = a^j.$$

Proof of Lemma 2.

Case 1: $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$. Then there exists a constant $C > 0$ such that, for sufficiently large n,

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j) \leq \sum_{j=0}^{\log_b n - 1} a^j C \left(\frac{n}{b^j}\right)^{\log_b a - \epsilon} = C \sum_{j=0}^{\log_b n - 1} a^j \frac{n^{\log_b a} n^{-\epsilon}}{b^{\log_b a^j} b^{-j\epsilon}} =$$

$$\frac{C \cdot n^{\log_b a}}{n^\epsilon} \sum_{j=0}^{\log_b n - 1} a^j \frac{(b^\epsilon)^j}{a^j} = \frac{C \cdot n^{\log_b a}}{n^\epsilon} \sum_{j=0}^{\log_b n - 1} (b^\epsilon)^j = \frac{C \cdot n^{\log_b a}}{(b^\epsilon - 1)} \frac{n^\epsilon - 1}{n^\epsilon} =$$

$$\frac{C \cdot n^{\log_b a}}{(b^\epsilon - 1)}(1 - \frac{1}{n^\epsilon}) = \mathrm{O}(n^{\log_b a}).$$

Case 2: $f(n) = \Theta(n^{\log_b a})$. Then there are constant $C_1 > 0$ for which

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j) \geq \sum_{j=0}^{\log_b n - 1} a^j C_1 \left(\frac{n}{b^j}\right)^{\log_b a} = C_1 \sum_{j=0}^{\log_b n - 1} a^j \frac{n^{\log_b a}}{b^{\log_b a^j}} =$$

$$C_1 \cdot n^{\log_b a} \sum_{j=0}^{\log_b n - 1} a^j \frac{1}{a^j} = C_1 \cdot n^{\log_b a} \sum_{j=0}^{\log_b n - 1} 1 = C_1 \cdot n^{\log_b a} \log_b n = \Omega(n^{\log_b a} \log_b n).$$

Similarly, we may show that $g(n) = \mathrm{O}(n^{\log_b a} \log_b n)$. Therefore, $g(n) = \Theta(n^{\log_b a} \log_b n)$.

Case 3: $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and $af(n/b) \leq cf(n)$ for some positive constant $c < 1$. Since $f(n)$ is equal to the first term ($j = 0$) of the sum that addds to $g(n)$, and all terms are nonnegative, we have that $g(n) = \Omega(f(n))$. Also, we may use induction to prove that, for all $j \geq 0$,

$$a^j f(\frac{n}{b^j}) \leq c^j f(n).$$

This gives

$$g(n) \leq f(n) \sum_{j=0}^{\log_b n - 1} c^j \leq \left(\frac{1 - c^{\log_b n}}{1 - c}\right) f(n)$$

which implies $g(n) = \mathrm{O}(f(n))$. Therefore, $g(n) = \Theta(f(n))$.

Case 4: $f(n) = \Theta(n^{\log_b a} \log_b^r n)$. Then there is a constant C_1 for which

$$g(n) = \sum_{j=0}^{\log_b n-1} a^j f(n/b^j) \geq \sum_{j=0}^{\log_b n-1} a^j C_1 \left(\frac{n}{b^j}\right)^{\log_b a} \log_b^r(\frac{n}{b^j}) = C_1 \sum_{j=0}^{\log_b n-1} a^j \frac{n^{\log_b a}}{b^{\log_b a^j}} (\log_b n - j)^r =$$

$$C_1 \cdot n^{\log_b a} \sum_{j=0}^{\log_b n-1} a^j \frac{1}{a^j} (\log_b n - j)^r = C_1 \cdot n^{\log_b a} \sum_{j=0}^{\log_b n-1} (\log_b n - j)^r = C_1 \cdot n^{\log_b a} (\log_b^r n + (\log_b n - 1)^r + \cdots + 1) \geq$$

$$C_1 \cdot n^{\log_b a} \cdot C \cdot \log_b^{r+1} n = \Omega(n^{\log_b a} \cdot \log_b^{r+1} n).$$

Similarly, we may show that

$$g(n) = \mathrm{O}(n^{\log_b a} \cdot \log_b^{r+1} n).$$

Completing the Proof of the Master Theorem

Lemma 1 tells us that $T(n) = \Theta(n^{\log_b a}) + g(n)$. Thus, by Lemma 2, we have the following for each of the four cases of the Master Theorem.

Case 1. $T(n) = \Theta(n^{\log_b a}) + \mathrm{O}(n^{\log_b a}) = \Theta(n^{\log_b a})$.

Case 2. $T(n) = \Theta(n^{\log_b a}) + \Theta(n^{\log_b a} \log n) = \Theta(n^{\log_b a} \log n)$.

Case 3. $T(n) = \Theta(n^{\log_b a}) + \Theta(f(n)) = \Theta(f(n))$, since in this case we assume $f(n) = \Omega(n^{\log_b a+\epsilon})$.

Case 4. $T(n) = \Theta(n^{\log_b a}) + \Theta(n^{\log_b a} \log^{r+1} n) = \Theta(n^{\log_b a} \log^{r+1} n)$.

Example 2. Determine the order of growth of $T(n)$ for the following recurrences.

1. $T(n) = 16T(n/4) + n$
2. $T(n) = T(n/5) + 20$
3. $T(n) = 3T(n/4) + n \log n$
4. $T(n) = 2T(n/2) + n \log n$

Substitution Method

The substitution method is an inductive method for proving the big-O growth of a function $T(n)$ that satisfies some divide-and-conquer recurrence. It requires that we already have a candidate function $g(n)$ for representing the growth of $T(n)$. For example, suppose we desire to show that $T(n) = \mathrm{O}(g(n))$. Then we peroform the following two steps.

$T\left(\frac{n}{5}\right) < T(n)$ $\quad \frac{n}{5} < n$

K

Inductive Assumption Assume that $T(k) \leq Cg(k)$, for all $k < n$ and for some constant $C > 0$.

Prove Inductive Step Show that $T(n) \leq Cg(n)$. Do this by replacing any term $aT(n/b)$ of the recurrence with $aCg(n/b)$, and show that the resulting non-recurrence expression is bounded above by $Cg(n)$.

Notice that there is no basis step to the above proof method. This is because we only care about the growth of $T(n)$ for sufficiently large n.

Example 3. Show that if $T(n) = 2T(n/2) + 3n$, then $T(n) = O(n \log n)$.

Inductive assumption: $T(k) \leq ck\log k$ for all $k < n$, and some const. $c > 0$.

show $\boxed{T(n) \leq cn\log n.}$

$$T(n) = 2T(\underbrace{n/2}_{=k}) + 3n \leq 2c\left(\frac{n}{2}\right)\log\left(\frac{n}{2}\right) + 3n$$

$$cn(\log n - \log 2) + 3n =$$

$$cn(\log n - 1) + 3n =$$

$$cn\cancel{\log n} - cn + 3n \leq cn\cancel{\log n}$$

$$\Leftrightarrow -cn + 3n \leq 0 \Leftrightarrow c\cancel{n} \geq 3\cancel{n}$$

$$\Leftrightarrow c \geq 3$$

$$\log_b a = ? \qquad b^? = a$$

$$c \geq n$$

$$c \leq 1 \qquad c \geq -10$$

$T(n) = 2T(n/2) + 3n$

Example 4. Similarly, show that $T(n) = \Omega(n \log n)$, where $T(n)$ satisfies the recurrence from Example 3.

New inductive assumption:

$T(k) \geq c k \log k$ for all $k < n$ and some const. $c > 0$.

Example 5. Show that if $T(n) = 2T(n/2) + n\log n$, then $T(n) = \mathrm{O}(n\log^2 n)$.

Inductive assumption:

$T(k) \le ck\log^2 k$ for all $k < n$ and some const. $c > 0$.

Show $T(n) \le cn\log^2 n$

$$T(n) = 2T(\underbrace{n/2}_{=k}) + n\log n \le 2c\left(\frac{n}{2}\right)\log^2\left(\frac{n}{2}\right) + n\log n$$

$$= cn(\log n - 1)^2 + n\log n =$$

$$\cancel{cn\log^2 n} - 2cn\log n + cn + n\log n \le \cancel{cn\log^2 n}$$

$$\iff -2cn\log n + cn + n\log n \le 0 \iff$$

$$2cn\log n - cn \ge n\log n \iff$$

$$c(2n\log n - n) \ge n\log n \iff$$

$$c \ge \frac{n\log n}{2n\log n - n} = \frac{1}{2 - \frac{n}{n\log n}} = \frac{1}{2 - \frac{1}{\log n}} \to \frac{1}{2} \text{ as } n \to \infty$$

is true so long as $c > \frac{1}{2}$

$\frac{1}{2}$ $\cancel{x}$ x x x C

Sometimes it is necessary to add one or more lesser-degree terms to the inductive assumption when the assumption fails to provide a term that counterbalances the $f(n)$ term of the recurrence. This happens in cases where $f(n)$ is little-o of the original inductive assumption. For example, instead of $T(k) \leq Ck^2$, one could instead use $T(k) \leq Ck^2 + Dk$ in case $f(n)$ is a linear term,
$T(k) \leq Ck^2 + D$ in case $f(n)$ is a constant term, or
$T(k) \leq Ck^2 + D \log k$ in case $f(n)$ is $\Theta(\log n)$, etc..

In all three of the above scenarios, it's important to note that D can be either positive or negative. Thus, after algebraic manipulation, a final inequality such as $D \leq -9$ is valid. This is true since, e.g., $T(n) = Cn^2 - 9 \log n = \Theta(n^2)$.

$$\frac{Cn^2 - 9\log n}{n^2} = C - \frac{9 \log n}{n^2} \to C$$

Example 6. Prove that if $T(n) = 2T(n/2) + 7$ then $T(n) = \mathrm{O}(n)$.

Inductive Assumption:

$T(k) \le ck$, for some const. $c > 0$ and all $k < n$

Show $T(n) \le cn$

$$T(n) = 2T(\underbrace{n/2}_{=k}) + 7 \le 2c\frac{n}{2} + 7 =$$

$cn + 7 \le cn \iff 7 \le 0$, which is false

2nd attempt: $T(k) \le ck + D$ for some constants $c > 0$ and integer D.

Show $\underline{T(n) \le cn + D}$.

$$T(n) = 2T(\underbrace{n/2}_{=k}) + 7 \le 2\left(c\left(\frac{n}{2}\right) + D\right) + 7 =$$

$cn + 2D + 7 \le cn + D \iff$

$D \le -7$

Core Exercises for Recurrences

1. Use the Master Theorem to give tight asymptotic bounds for the following recurrences.

   a. $T(n) = 2T(n/4) + 1$

   b. $T(n) = 2T(n/4) + \sqrt{n}$

   c. $T(n) = 2T(n/4) + n$

   d. $T(n) = 2T(n/4) + n^2$

2. Use the Master Theorem to show that the solution to the binary-search recurrence $T(n) = T(n/2) + a$, where $a > 0$ is a constant, is $T(n) = \Theta(\log n)$.

3. Use the Master Theorem to determine the big-O growth of $T(n)$ if it satisfies the recurrence $T(n) = 3T(n/2) + n$.

4. Use the Master Theorem to determine the big-O growth of $T(n)$ if it satisfies the recurrence $T(n) = 4T(n/2) + n^2 \log n$.

5. Use the substitution method to prove that $T(n) = 2T(n/2) + an$, $a > 0$ a constant, has solution $T(n) = \Theta(n \log n)$.

6. Use the substitution method to prove that $T(n) = 4T(n/2) + n$ has solution $T(n) = \mathrm{O}(n^2)$.

7. Use the substitution method to prove that $T(n) = T(an) + T(bn) + n$ has solution $T(n) = \mathrm{O}(n)$, where a and b are positive constants, with $a + b < 1$.

8. Use the substitution method to prove that $T(n) = 8T(n/2) - \log n$ has solution $T(n) = \Omega(n^3)$.

9. Use the substitution method to prove that $T(n) = 2T(n/4) + \sqrt{n} \log n$ has solution $T(n) = \Omega(\sqrt{n} \log^2 n)$.

Solutions to Core Exercises for Recurrences

1. Use the Master Theorem to give tight asymptotic bounds for the following recurrences.

   a. Case 1: $T(n) = \Theta(\sqrt{n})$

   b. Case 2: $T(n) = \Theta(\sqrt{n} \log n)$

   c. Case 3: $T(n) = \Theta(n)$

   d. Case 3: $T(n) = \Theta(n^2)$

2. Use Case 2 of the Master Theorem: $T(n) = \Theta(\log n)$.

3. Use case 1. $T(n) = \Theta(n^{\log 3})$.

4. Use Case 4 of the Master Theorem: $T(n) = \Theta(n^2 \log^2 n)$.

5. Assume $T(k) \leq ck \log k$, for all $k < n$. Then

$$T(n) = 2T(n/2) + an \leq 2c(n/2)\log(n/2) + an = cn \log n - cn + an \leq cn \log n$$

   iff $c \geq a$. Therefore by induction, $T(n) = \mathrm{O}(n \log n)$. $T(n) = \Omega(n \log n)$ is proved similarly.

6. The hypothesis $T(k) \leq Ck^2$, for all $k < n$ is insufficient, since it leads to the inequality $n \leq 0$. Using $T(k) \leq Ck^2 + Dk$ yields

$$T(n) = 4T(n/2) + n \leq 4(C(n/2)^2 + Dn/2) + n = Cn^2 + 2Dn + n \leq Cn^2 + Dn \Leftrightarrow$$

$$Dn + n \leq 0 \Leftrightarrow D \leq -1.$$

   Therefore, the inequality is established so long as we choose $D \leq -1$, and $C > 0$.

7. Assume $T(k) \leq Ck$, for all $k < n$ and some constant $C > 0$. Then

$$T(n) = T(an) + T(bn) + n \leq Can + Cbn + n = Cn(a + b) + n \leq Cn \Leftrightarrow$$

$$Cn(1 - a - b) \geq n \Leftrightarrow C \geq 1/(1 - a - b).$$

   Since $a+b < 1$, we have $1-a-b > 0$, and thus the inequality holds, so long as $C \geq 1/(1-a-b)$.

8. The hypothesis $T(k) \geq Ck^3$, for all $k < n$ is insufficient, since it leads to the inequality $\log n \leq 0$ which is false for $n \geq 2$. However, using $T(k) \geq Ck^3 + D \log k$ yields

$$T(n) = 8T(n/2) - \log n \geq 8(C(n/2)^3 + D\log(n/2)) - \log n =$$

$$Cn^3 + 8D \log n - 8D - \log n \geq Cn^3 + D \log n \Leftrightarrow 7D \log n - 8D \geq \log n \Leftrightarrow$$

$$D(7 - 8/\log n) \geq 1 \Leftrightarrow D \geq \frac{1}{(7 - 8/\log n)}.$$

   Notice that, for $n \geq 1$, the right side always exceeds $1/7$ (why?) gets arbitrarily close to $1/7$ as n increases. Therefore, the inequality holds so long we choose $D > 1/7$. This works because recall that we only need the inequality to hold for *sufficiently large values of* n. Furthermore, since the expression gets closer and closer to $1/7$ and $D > 1/7$, there will be some K value for which $n \geq K$ implies that the expression will become less than D, which is desired. Remember that big-O and big-Ω inequalities only need to be true for sufficiently large n. In theory, the value of K does not matter, so long as we know it exists.

9. Assume $T(k) \geq C\sqrt{k}\log^2 k$ for all $k < n$. Then

$$T(n) = 2T(n/4) + \sqrt{n}\log n \geq 2(C\sqrt{n/4}\log^2(n/4)) + \sqrt{n}\log n =$$

$$C\sqrt{n}(\log n - 2)^2 + \sqrt{n}\log n = C\sqrt{n}\log^2 n - 4C\sqrt{n}\log n + 4C\sqrt{n} + \sqrt{n}\log n \geq C\sqrt{n}\log^2 n \Leftrightarrow$$

$$4C\sqrt{n}\log n - 4C\sqrt{n} \leq \sqrt{n}\log n \Leftrightarrow C(4 - 4/\log n) \leq 1 \Leftrightarrow C \leq \frac{1}{4(1 - \log n)}.$$

Notice that the right side is always greater than 1/4 when $n \geq 3$. Therefore, the inequality is proved so long as $C \leq 1/4$ and $n \geq 3$.

Additional Exercises

A. Draw the recursion tree that results when applying Mergesort to the array $5, -2, 0, 7, 3, 11, 2, 9, 5, 6$, Label each node with the sub-problem to be solved at that point of the recursion. Assume arrays of size 1 and 2 are base cases. Assume that odd-sized arrays are split so that the left subproblem has one more integer than the right. Next to each node, write the solution to its associated subproblem.

B. Write an algorithm for the function `Merge` that takes as input two sorted integer arrays a and b of sizes m and n respectively, and returns a sorted integer array of size $m+n$ whose elements are the union of elements from a and b.

C. Solve the recurrence $T(n) = 2T(\sqrt{n}) + \log n$. Hint: Let $S(k) = T(2^k)$ and write a divide-and-conquer recurrence for $S(k)$.

D. Argue that the solution to the recurrence $T(n) = T(n/3) + T(2n/3) + an$, where $a > 0$ is a constant, is $T(n) = \Theta(n \log n)$, by using an appropriate recursion tree. The verify it using the substitution method.

E. Use a recursion tree to show that the solution of $T(n) = T(n-1) + n$ is $T(n) = \mathrm{O}(n^2)$.

F. Use a recursion tree to solve the recurrence $T(n) = T(n-2) + n^2$; providing a tight upper and lower bound.

G. Use a recursion tree to estimate the big-O growth of $T(n)$ which satisifies the recurrence $T(n) = 2T(n-1) + 1$.

H. Suppose $T(n)$ and $g(n)$ are positive functions (meaning they output positive values for each input $n \geq 0$) that satisfy $T(n) \leq Cg(n)$ for all $n \geq k$, and some constant $C > 0$. Prove that There is a constant $C' > 0$ for which $T(n) \leq C'g(n)$, for *all* $n \geq 0$.

I. Suppose $f(n) = n^s \log^r n$, where $s > \log_b a$ and $r \geq 0$. Show that there exists a positive $c < 1$ for which

$$af(n/b) < cf(n),$$

where we may assume that n is an arbitrary power of b.

Solutions to Additional Exercises

A. We use lr-strings for the addresses of each node. For example, λ denotes the root, while lrr denotes the right child of the right child of the left child of the root. Then

$$\lambda : 5, -2, 0, 7, 3, 11, 2, 9, 5, 6$$

$$l : 5, -2, 0, 7, 3 \qquad r : 11, 2, 9, 5, 6$$

$$ll : 5, -2, 0 \qquad lr : 7, 3 \qquad rl : 11, 2, 9 \qquad rr : 5, 6$$

$$lll : 5, -2 \qquad llr : 0 \qquad rll : 11, 2 \qquad rlr : 9$$

B.
```
//Merges a and b into merged. n= |a|, m = |b|
void merge(int[] a, int n, int[] b, int m, int[] merged)
{
    int i=0;
    int j=0;
    int value_a = a[0];
    int value_b = b[0];
    int count = 0;

    while(true)
    {
        if(value_a <= value_b)
        {
            merged[count++] = value_a;
            i++;

            if(i < n)
                value_a = a[i];
            else
                break;
        }
        else
        {
            merged[count++] = value_b;
            j++;

            if(j < m)
                value_b = b[j];
            else
                break;
        }
    }

//copy remaining values
    if(j < m)
        for(; j < m; j++)
```

```
                merged[count++] = b[j];
        else
            for(; i < n; i++)
                 merged[count++] = a[i];
    }
```

C. Letting $S(k) = T(2^k)$, we then have the divide-and-conquer recurrence

$$S(k) = T(2^k) = 2T((2^k)^{1/2}) + \log 2^k = 2T(2^{k/2}) + k = 2S(k/2) + k,$$

which, by the Master Theorem, yields $S(k) = \Theta(k \log k)$. Finally, letting $n = 2^k$, we have $T(n) = \Theta(\log n \log(\log n))$.

D. The recursion tree consists of a single branch of length n, where the work done at depth i is $n - i$. Hence, $T(n) = \mathrm{O}(\sum_{i=0}^{n}(n-i)) = \mathrm{O}(n^2)$.

E. The recursion tree consists of a single branch of length $n/2$, where the work done at depth i is $(n-2i)^2$. Hence,

$$T(n) = \Theta(\sum_{i=0}^{n/2}(n-2i)^2) = \Theta(n^3).$$

F. The recursion tree is a perfect binary tree having depth n. Thus, it has $2^{n+1} - 1$ nodes. Moreover, since the work at each node is always 1, we see that $T(n) = \Theta(2^n)$.

G. The recursion tree remains perfect all the way down to depth $\lfloor \log_3 n \rfloor$. Moreover, for each depth $i = 0, 1, \ldots, \lfloor \log_3 n \rfloor$, the work done at that depth always adds to n. Hence, the total work (i.e. $T(n)$) must be $\Omega(n \log n)$. Also, the longest branch has a depth not exceeding $\lfloor \log_{\frac{3}{2}} n \rfloor$. Moreover, the amount of work at each depth does not exceed n. Hence $T(n) = \mathrm{O}(n \log n)$. Therefore, $T(n) = \Theta(n \log n)$.

H. Let C_i, $i = 0, 1, \ldots, k-1$, be a constant for which $T(i) \le C_i g(i)$. Since $f(i)$ and $g(i)$ are both positive numbers, we know that such a C_i exists. Then choose $C' = \max(C_0, \ldots, C_{k-1}, C)$. Then $T(n) \le C' g(n)$, for all $n \ge 0$.

I. We have

$$a\left(\frac{n}{b}\right)^s \log\left(\frac{n}{b}\right)^r = a\left(\frac{n}{b}\right)^s (\log n - \log b)^r < cn^s \log^r n \Leftrightarrow$$

$$\frac{a}{b^s}\left(\frac{\log n - \log b}{\log n}\right)^r = \frac{a}{b^s}\left(1 - \frac{\log b}{\log n}\right)^r \le \frac{a}{b^s} < c,$$

and the last inequality is true since $s > \log_b a$ implies that $b^s > a$ and so $\frac{a}{b^s} < 1$. Therefore, there does exist a constant $c < 1$ for which $\frac{a}{b^s} < c$ is true.