Growth Rate of Solution to Recurrences

Divide and Conquer Algorithms

One of the most basic and powerful algorithmic techniques is **divide and conquer**. Consider, for example, the binary search algorithm, which we will describe in the context of guessing a number between 1 and 100. Suppose

Growth Rate of Solution to Recurrences

```
MergeSort(A,low,high)

// This algorithm sorts the portion of list A from
// location low to location high.
if (low == high)
      return
else
      mid = ⌊(low + high)/2⌋
      MergeSort(A,low,mid)
      MergeSort(A,mid+1,high)
      Merge the sorted lists from the previous two steps
      return
```

Growth Rate of Solution to Recurrences

Thus, we obtain the following recurrence for the running time of merge sort:

$$T(n) = \begin{cases} 2T(n/2) + n & \text{if } n > 1, \\ 1 & \text{if } n = 1. \end{cases} \tag{4.19}$$

Recurrences such as this one can be understood via the idea of a recursion tree, which we introduce next. This concept allows us to analyze recurrences that arise in divide-and-conquer algorithms, as well as those that arise in other recursive situations, such as the Tower of Hanoi.

Growth Rate of Solution to Recurrences

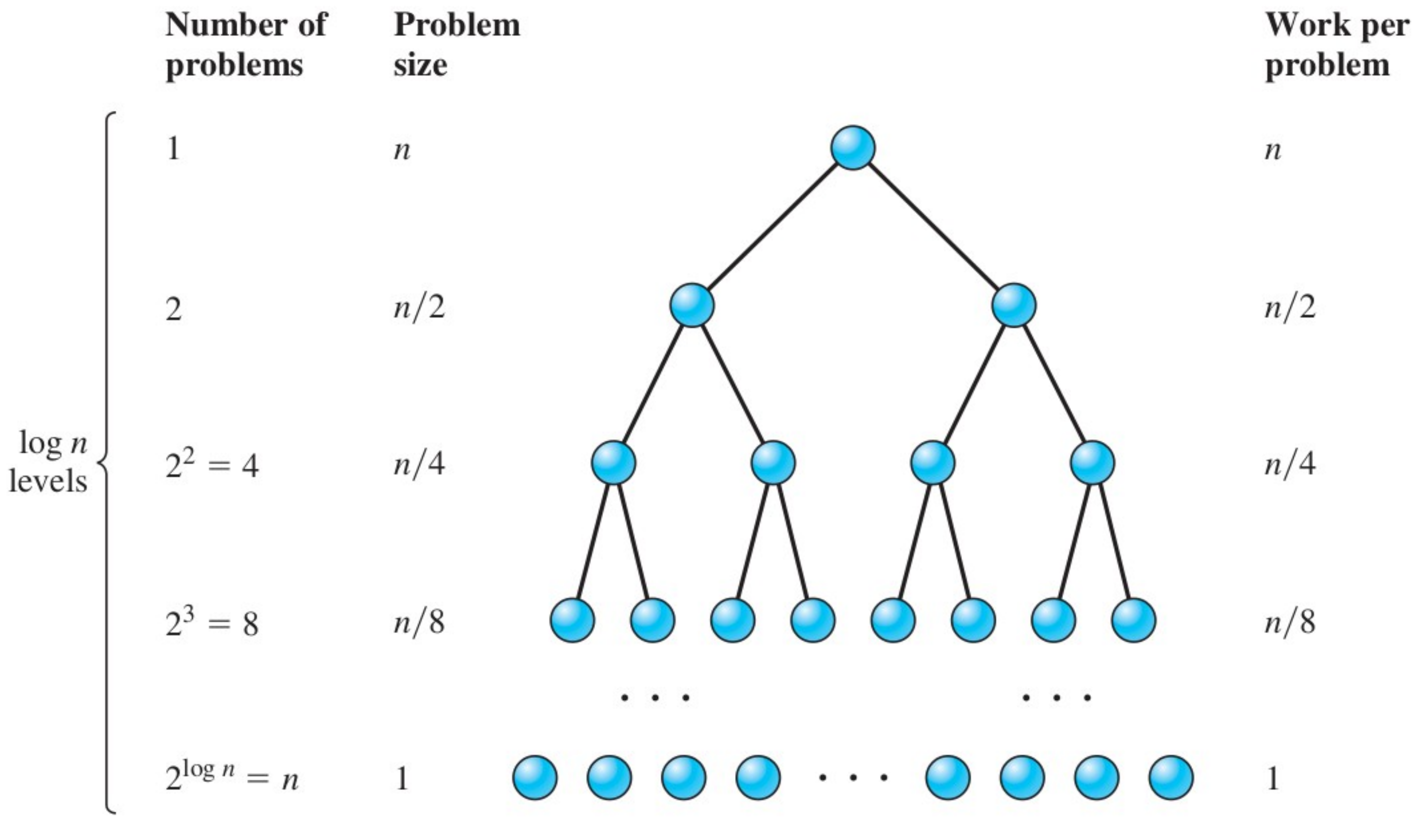


Figure 4.6: A finished recursion tree diagram

Growth Rate of Solution to Recurrences

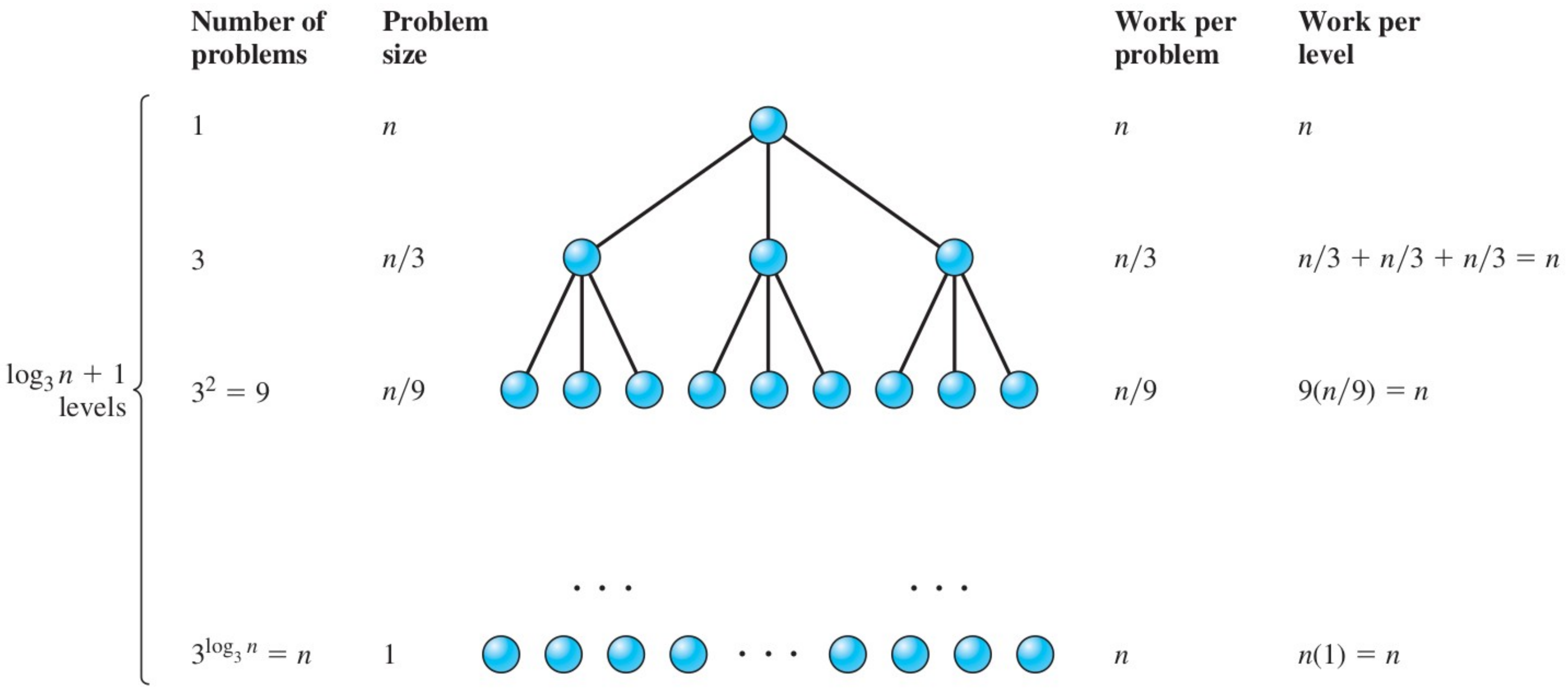


Figure 4.8: The recursion tree diagram for the recurrence in Exercise 4.3-2

Growth Rate of Solution to Recurrences

Exercise 4.3-2

Use a recursion tree to find a big Θ bound for the solution to the recurrence

$$T(n) = \begin{cases} 3T(n/3) + n & \text{if } n \geq 3, \\ 1 & \text{if } n < 3. \end{cases}$$

Assume that n is a power of 3.

Growth Rate of Solution to Recurrences

Exercise 4.3-3

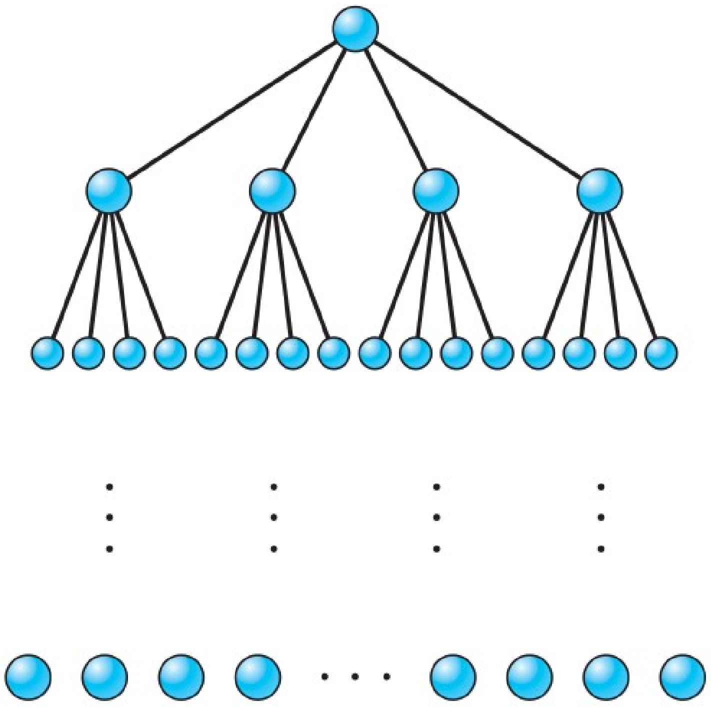
Use a recursion tree to solve the recurrence

$$T(n) = \begin{cases} 4T(n/2) + n & \text{if } n \geq 2, \\ 1 & \text{if } n = 1. \end{cases}$$

Assume that n is a power of 2. Convert your solution to a big Θ statement about the behavior of the solution.

Growth Rate of Solution to Recurrences

	Number of problems	Problem size	Work per problem	Work per level
$\log n + 1$ levels	1	n	n	n
	4	$n/2$	$n/2$	$n/2 + n/2 + n/2 + n/2 = 2n$
	$4^2 = 16$	$n/4$	$n/4$	$16(n/4) = 4n$
	$\vdots$			
	$4^{\log_2 n} = n^2$	1	1	$n^2 \cdot 1 = n^2$



Master Theorem

Theorem 4.9 **(Master Theorem, Preliminary Version)** Let a be an integer greater than or equal to 1, and let b be a real number greater than 1. Let c be a positive real number, and d, a nonnegative real number. Given a recurrence of the form

$$T(n) = \begin{cases} aT(n/b) + n^c & \text{if } n > 1, \\ d & \text{if } n = 1, \end{cases}$$

in which n is restricted to be a power of b, we get the following:

1. If $\log_b a < c$, then $T(n) = \Theta(n^c)$.
2. If $\log_b a = c$, then $T(n) = \Theta(n^c \log n)$.
3. If $\log_b a > c$, then $T(n) = \Theta(n^{\log_b a})$.

Master Theorem

Theorem 4.10

(Master Theorem) Let a and b be positive real numbers, with $a \geq 1$ and $b > 1$. Let $T(n)$ be defined for integers n that are powers of b by

$$T(n) = \begin{cases} aT(n/b) + f(n) & \text{if } n > 1, \\ d & \text{if } n = 1. \end{cases}$$

Then we have the following:

1. If $f(n) = \Theta(n^c)$, where $\log_b a < c$, then $T(n) = \Theta(n^c) = \Theta\big(f(n)\big)$.
2. If $f(n) = \Theta(n^c)$, where $\log_b a = c$, then $T(n) = \Theta(n^c \log n) = \Theta\big(f(n) \log n\big)$.
3. If $f(n) = \Theta(n^c)$, where $\log_b a > c$, then $T(n) = \Theta(n^{\log_b a})$.

Master Theorem

Theorem 4.11

Let a and b be positive real numbers, with $a \geq 1$ and $b \geq 2$. Let $T(n)$ satisfy the recurrence

$$T(n) = \begin{cases} aT(\lceil n/b \rceil) + f(n) & \text{if } n > 1, \\ d & \text{if } n = 1. \end{cases}$$

Then we have the following:

1. If $f(n) = \Theta(n^c)$, where $\log_b a < c$, then $T(n) = \Theta(n^c) = \Theta\big(f(n)\big)$.
2. If $f(n) = \Theta(n^c)$, where $\log_b a = c$, then $T(n) = \Theta(n^c \log n) = \Theta\big(f(n) \log n\big)$.
3. If $f(n) = \Theta(n^c)$, where $\log_b a > c$, then $T(n) = \Theta(n^{\log_b a})$.

Recurrences and Selection

Exercise 4.6-2 Give the fastest algorithm you can to find the median ($i = \lceil n/2 \rceil$).

The Idea of Selection

One common problem that arises in algorithms is that of **selection**. In this situation, we are given n distinct data items from some set that has an underlying order. That is, given any two items a and b from that set, we can determine whether $a < b$. (Integers satisfy this property, but colors do not.) Given these n items and some value i with $1 \leq i \leq n$, we are asked to find the ith-smallest item in the set. For example, in the set

$$S = \{3, 2, 8, 6, 4, 11, 7\}, \tag{4.28}$$

Recurrences and Selection

A Recursive Selection Algorithm

Suppose that we magically knew how to find the median in $O(n)$ time. That is, we have a routine MagicMedian that returns the median when given a set A as input. We could then use this routine in a divide-and-conquer algorithm for Select, as follows.

Recurrences and Selection

```
Select1(A,i,n)

// Selects the ith-smallest element in set A,
// where n = |A|
if (n == 1)
     return the one item in A
else
     p = MagicMiddle(A)
     Let H be the set of elements greater than p
     Let L be the set of elements less than or equal to p
     if (i ≤ |L|)
          return Select1(L,i,|L|)
     else
          return Select1(H,i − |L|,|H|)
```

Recurrences and Selection

```
MagicMiddle(A)

(1)  Let n = |A|
(2)  if (n < 60)
(3)        use sorting to return the median of A
(4)  else
(5)        Break A into k = ⌈n/5⌉ groups G_1,...,G_k
           with ⌊n/5⌋ of size 5 and perhaps one of smaller size
(6)        for i = 1 to k
(7)              find m_i, the median of G_i (by sorting)
(8)        Let M = {m_1,..., m_k}
(9)        return Select1(M,⌈k/2⌉,k)
```

Recurrences and Selection

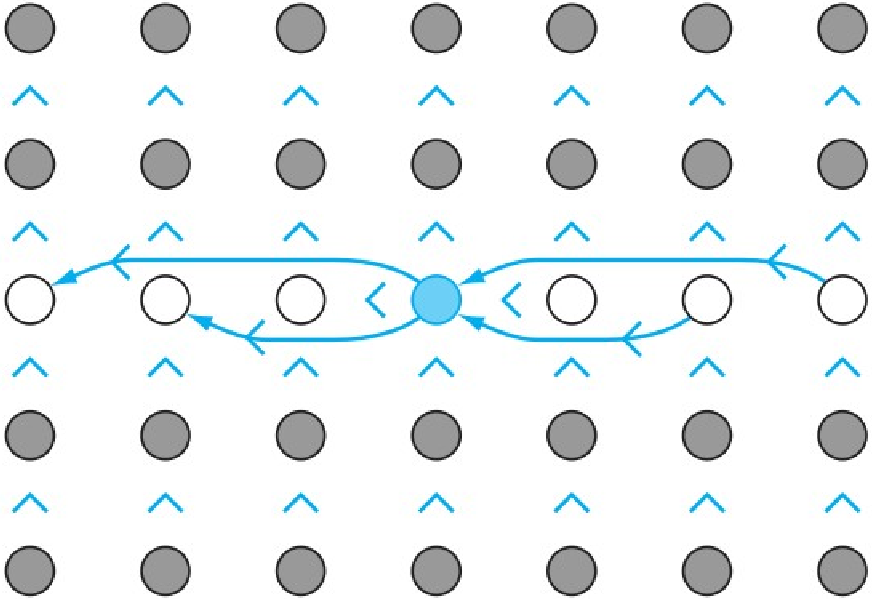


Figure 4.12: Dividing a set into $n/5$ parts of size 5, finding the median of each part, and finding the median of the medians

Recurrences and Selection

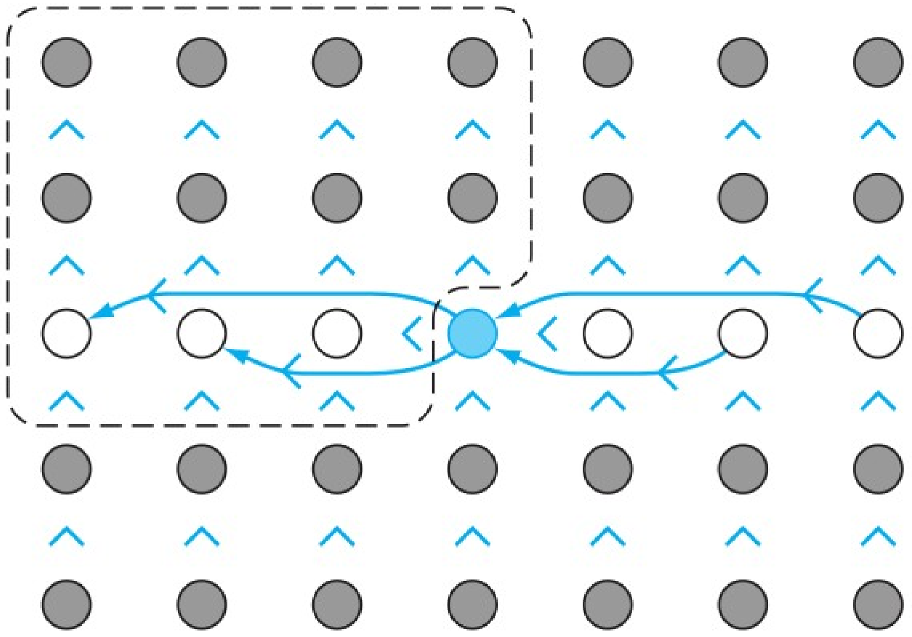


Figure 4.13: The enclosed elements are less than the median of the medians

Recurrences and Selection

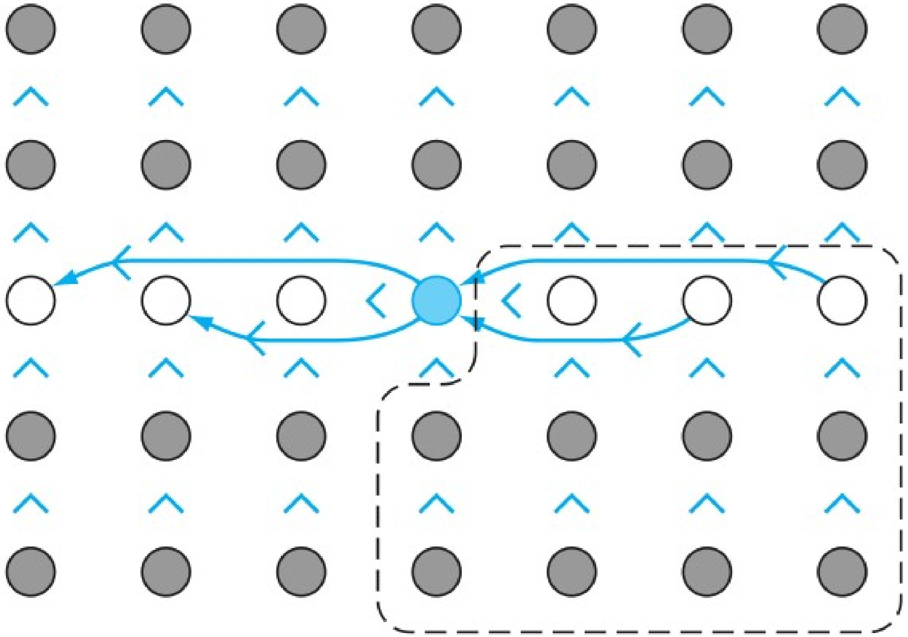


Figure 4.14: The enclosed elements are greater than the median of the medians

Recurrences and Selection

Lemma 4.14 The value returned by MagicMiddle(A) is in the middle half of A.

Proof We let m^* denote the output of MagicMiddle(A), so that m^* is the $\lceil k/2 \rceil$th element of the m_i's in sorted order. Thus, $\lceil k/2 \rceil - 1$ medians m_i are less than m^*, as are the elements in G_i less than m_i. Choose j so that $m^* \in G_j$. Then the elements of G_j less than m_j are less than m^*. However, for all but perhaps one G_i (including G_j) with $m_i \leq m^*$, there are two elements less than m_i, so that the set S' of elements less than m^* has size at least $3(\lceil k/2 \rceil - 1)$. Because k is at least $n/5$ and $\lceil n/10 \rceil \geq n/10$, we have that

$$|S'| \geq 3\left(\left\lceil \frac{n}{10} \right\rceil - 1\right) \geq 3\left(\frac{n}{10} - 1\right).$$

Recurrences and Selection

Thus, if we choose n so that

$$3\left(\frac{n}{10}-1\right)=0.3n-3\geq\frac{n}{4}, \tag{4.29}$$

we will have $S' \geq n/4$. But Equation 4.29 gives us $0.3n - 3 \geq 0.25n$, or $n \geq 60$. Now because there are $k - \lceil k/2 \rceil$ medians m_i greater than m^*, we have, as with S', that if B' is the set of elements of A larger than m^*, then B' has at least $3(k - \lceil k/2 \rceil)$ elements. Because $\lceil k/2 \rceil < k/2 + 1$, we have

$$|B| \geq 3\left(k-\frac{k}{2}-1\right)=3\left(\frac{k}{2}-1\right)=3\left(\frac{1}{2}\left\lceil\frac{n}{5}\right\rceil-1\right)\geq 3\frac{n}{10}-3=0.3n-3.$$

Thus, if we choose n so that Equation 4.29 holds—that is, so that $n \geq 60$—then we have both $|S'| > n/4$ and $|B'| > n/4$. Therefore, m^* is in the middle half of A.

Recurrences and Selection

Important Concepts, Formulas, and Theorems

1. *Divide-and-conquer algorithm.* A *divide-and-conquer algorithm* is one that solves a problem by dividing the problem into "subproblems" that are smaller than, but otherwise of the same type as, the original one; recursively solving these subproblems; and then assembling the solution of these subproblems into a solution of the original one. Although not all problems can be solved by such a strategy, a great many problems of interest in computer science can be.

Recurrences and Selection

2. *Merge sort*. In *merge sort*, we sort a list of items that have some underlying order by dividing the list in half, sorting the first half (by recursively using merge sort), sorting the second half (by recursively using merge sort), and then merging the two sorted lists. For a list of length 1, merge sort returns the same list.

Recurrences and Selection

3. *Recursion tree diagram.* We draw a *recursion tree diagram* for a recurrence by levels, with each level representing a level of recursion. A level of a recursion tree diagram has five parts: two on the left, one in the middle, and two on the right. On the left, we keep track of the problem size and the number of problems; in the middle, we draw the tree; and on the right, we keep track of the work done per problem and the total amount of work done on

Recurrences and Selection

4. *The base level of a recursion tree.* The amount of work done on the lowest level in a recursion tree is the number of nodes times the value given by the initial condition; it is not determined by attempting to make a computation of "additional work" done at the lowest level.

5. *Bases for logarithms.* We use $\log n$ as an alternate notation for $\log_2 n$. A fundamental fact about logarithms is that $\log_b n = \Theta(\log_2 n)$ for any real number $b > 1$.

Recurrences and Selection

6. *An important fact about logarithms.* For any $b > 0$, we have $a^{\log_b n} = n^{\log_b a}$.

7. *Three behaviors of solutions.* The solution to a recurrence of the form $T(n) = aT(n/2) + n$ behaves in one of the following ways:

   a. If $a < 2$, then $T(n) = \Theta(n)$.

   b. If $a = 2$, then $T(n) = \Theta(n \log n)$.

   c. If $a > 2$, then $T(n) = \Theta(n^{\log_2 a})$.