This file contains the exercises, hints, and solutions for Chapter 7 of the book "Introduction to the Design and Analysis of Algorithms," 2nd edition, by A. Levitin. The problems that might be challenging for at least some students are marked by ▷; those that might be difficult for a majority of students are marked by ►.

Exercises 7.1

1. Is it possible to exchange numeric values of two variables, say, u and v, without using any extra storage?

2. Will the comparison counting algorithm work correctly for arrays with equal values?

3. Assuming that the set of possible list values is $\{a,\ b,\ c,\ d\}$, sort the following list in alphabetical order by the distribution counting algorithm:

$$b,\ \ c,\ \ d,\ \ c,\ \ b,\ \ a,\ \ a,\ \ b.$$

4. Is the distribution counting algorithm stable?

5. Design a one-line algorithm for sorting any array of size n whose values are n distinct integers from 1 to n.

6. ▷ The ***ancestry problem*** asks to determine whether a vertex u is an ancestor of vertex v in a given binary (or, more generally, rooted ordered) tree of n vertices. Design a $O(n)$ input enhancement algorithm that provides sufficient information to solve this problem for any pair of the tree's vertices in constant time.

7. ► The following technique, known as ***virtual initialization***, provides a time-efficient way to initialize just some elements of a given array $A[0..n-1]$ so that for each of its elements, we can say in constant time whether it has been initialized and, if it has been, with which value. This is done by utilizing a variable *counter* for the number of initialized elements in A and two auxiliary arrays of the same size, say $B[0..n-1]$ and $C[0..n-1]$, defined as follows. $B[0], ..., B[counter-1]$ contain the indices of the elements of A that were initialized: $B[0]$ contains the index of the element initialized first, $B[1]$ contains the index of the element initialized second, and so on. Furthermore, if $A[i]$ was the kth element ($0 \le k \le counter-1$) to be initialized, $C[i]$ contains k.

   a. Sketch the state of arrays $A[0..7]$, $B[0..7]$, and $C[0..7]$ after the three assignments

$$A[3] \leftarrow x;\quad A[7] \leftarrow z;\quad A[1] \leftarrow y.$$

   b. In general, how can we check with this scheme whether $A[i]$ has been initialized and, if it has been, with which value?

8. a. Write a program for multiplying two sparse matrices, a p-by-q matrix A and a q-by-r matrix B.

   b. Write a program for multiplying two sparse polynomials $p(x)$ and $q(x)$ of degrees m and n, respectively.

9. Write a program that plays the game of tic-tac-toe with the human user by storing all possible positions on the game's 3-by-3 board along with the best move for each of them.

Hints to Exercises 7.1

1. Yes, it is possible. How?
2. Check the algorithm's pseudocode to see what it does upon encountering equal values.
3. Trace the algorithm on the input given (see Figure 7.2 for an example).
4. Check whether the algorithm can reverse a relative ordering of equal elements.
5. Where will $A[i]$ be in the sorted array?
6. Take advantage of the standard traversals of such trees.
7. a. Follow the definitions of the arrays B and C in the description of the method.

   b. Find, say, $B[C[3]]$ for the example in part (a).
8. a. Use linked lists to hold nonzero elements of the matrices.

   b. Represent each of the given polynomials by a linked list with nodes containing exponent i and coefficient a_i for each nonzero term $a_i x^i$.
9. You may want to take advantage of the board's symmetry to decrease the number of the positions that need to be stored.

Solutions to Exercises 7.1

1. The following operations will exchange values of variables u and v:
   $u \leftarrow u + v$ //u holds $u + v$, v holds v
   $v \leftarrow u - v$ //u holds $u + v$, v holds u
   $u \leftarrow u - v$ //u holds v, v holds u

   Note: The same trick is applicable, in fact, to any binary data by employing the "exclusive or" (XOR) operation:
   $u \leftarrow u\text{XOR}v$
   $v \leftarrow u\text{XOR}v$
   $u \leftarrow u\text{XOR}v$

2. Yes, it will work correctly for arrays with equal elements.

3. Input: A: $b,\ c,\ d,\ c,\ b,\ a,\ a,\ b$

	a	b	c	d
Frequencies	2	3	2	1

	a	b	c	d
Distribution values	2	5	7	8

	$D[a..d]$			
$A[7] = b$	2	**5**	7	8
$A[6] = a$	**2**	4	7	8
$A[5] = a$	**1**	4	7	8
$A[4] = b$	0	**4**	7	8
$A[3] = c$	0	3	**7**	8
$A[2] = d$	0	3	6	**8**
$A[1] = c$	0	3	**6**	7
$A[0] = b$	0	**3**	5	7

$S[0..7]$							
				b			
	a						
a							
			b				
						c	
							d
					c		
		b					

4. Yes, it is stable because the algorithm scans its input right-to-left and puts equal elements into their section of the sorted array right-to-left as well.

5. **for** $i \leftarrow 0$ **to** $n - 1$ **do** $S[A[i] - 1] \leftarrow A[i]$

6. Vertex u is an ancestor of vertex v in a rooted ordered tree T if and only if the following two inequalities hold

$$preorder(u) \le preorder(v) \quad \text{and} \quad postorder(u) \ge postorder(v),$$

   where *preorder* and *postorder* are the numbers assigned to the vertices by the preorder and postorder traversals of T, respectively. Indeed, preorder traversal visits recursively the root and then the subtrees numbered from left to right. Therefore,

$$preorder(u) \le preorder(v)$$

if and only if either u is an ancestor of v (i.e., u is on the simple path from the root's tree to v) or u is to the left of v (i.e., u and v are not on the same simple path from the root to a leaf and $T(u)$ is to the left of $T(v)$ where $T(u)$ and $T(v)$ are the subtrees of the nearest common ancestor of u and v, respectively). Similarly, postorder traversal visits recursively the subtrees numbered from left to right and then the root. Therefore,

$$postorder(u) \geq postorder(v)$$

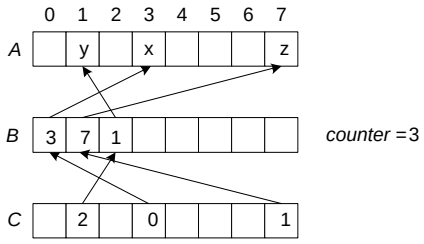
if and only if either u is an ancestor of v or v is to the left of u. Hence,

$$preorder(u) \leq preorder(v) \quad \text{and} \quad postorder(u) \geq postorder(v)$$

is necessary and sufficient for u to be an ancestor of v.

The time efficiencies of both traversals are in $O(n)$ (Section 4.4); once the preorder and postorder numbers are precomputed, checking the two inequalities takes constant time for any given pair of the vertices.

7. a. The following diagram depicts the results of these assignments (the values of the unspecified elements in the arrays are undefined):



b. $A[i]$ is initialized if and only if $0 \leq C[i] \leq counter - 1$ and $B[C[i]] = i$. (It is useful to note that the elements of array C define the inverse to the mapping defined by the elements of array B.) Hence, if these two conditions hold, $A[i]$ contains the value it has been initialized with; otherwise, it has not been initialized.

8. n/a

9. n/a

Exercises 7.2

1. Apply Horspool's algorithm to search for the pattern `BAOBAB` in the text

   `BESS_KNEW_ABOUT_BAOBABS`

2. Consider the problem of searching for genes in DNA sequences using Horspool's algorithm. A DNA sequence consists of a text on the alphabet {`A`, `C`, `G`, `T`} and the gene or gene segment is the pattern.

   a. Construct the shift table for the following gene segment of your chromosome 10:

   `TCCTATTCTT`

   b. Apply Horspool's algorithm to locate the above pattern in the following DNA sequence:

   `TTATAGATCTCGTATTCTTTTATAGATCTCCTATTCTT`

3. How many character comparisons will be made by Horspool's algorithm in searching for each of the following patterns in the binary text of 1000 zeros?

   a. 00001

   b. 10000

   c. 01010

4. For searching in a text of length n for a pattern of length m ($n \geq m$) with Horspool's algorithm, give an example of

   a. worst-case input.

   b. best-case input.

5. Is it possible for Horspool's algorithm to make more character comparisons than the brute-force algorithm would make in searching for the same pattern in the same text?

6. If Horspool's algorithm discovers a matching substring, how large a shift should it make to search for a next possible match?

7. How many character comparisons will the Boyer-Moore algorithm make in searching for each of the following patterns in the binary text of 1000 zeros?

a. 00001

b. 10000

c. 01010

8. a. Would the Boyer-Moore algorithm work correctly with just the bad-symbol table to guide pattern shifts?

   b. Would the Boyer-Moore algorithm work correctly with just the good-suffix table to guide pattern shifts?

9. a. If the last characters of a pattern and its counterpart in the text do match, does Horspool's algorithm have to check other characters right to left, or can it check them left to right too?

   b. Answer the same question for the Boyer-Moore algorithm.

10. Implement Horspool's algorithm, the Boyer-Moore algorithm, and the brute-force algorithm of Section 3.2 in the language of your choice and run an experiment to compare their efficiencies for matching

    a. random binary patterns in random binary texts.

    b. random natural language patterns in natural language texts.

Hints to Exercises 7.2

1. Trace the algorithm in the same way it is done in the section for another instance of the string-matching problem.
2. A special alphabet notwithstanding, this application is not different than applications to natural language strings.
3. For each pattern, fill in its shift table and then determine the number of character comparisons (both successful and unsuccessful) on each trial and the total number of trials.
4. Find an example of a binary string of length m and a binary string of length n ($n \geq m$) so that Horspool's algorithm makes

   a. the largest possible number of character comparisons before making the smallest possible shift.

   b. the smallest possible number of character comparisons.
5. It is logical to try a worst-case input for Horspool's algorithm.
6. Can the algorithm shift the pattern by more than one position without the possibility of missing another matching substring?
7. For each pattern, fill in the two shift tables and then determine the number of character comparisons (both successful and unsuccessful) on each trial and the total number of trials.
8. Check the description of the Boyer-Moore algorithm.
9. Check the descriptions of the algorithms.
10. n/a

Solutions to Exercises 7.2

1. The shift table for the pattern BAOBAB in a text comprised of English letters, the period, and a space will be

c	A	B	C	D	.	.	.	O	.	.	.	Z	.	_
$t(c)$	1	2	6	6	6			3	6			6	6	6

The actual search will proceed as shown below:

```
B E S S _ K N E W _ A B O U T _ B A O B A B S
B A O B A B                   B A O B A B
            B A O B A B             B A O B A B
                B A O B A B
```

2. a. For the pattern TCCTATTCTT and the alphabet {A, C, G, T}, the shift table looks as follows:

c	A	C	G	T
$t(c)$	5	2	10	1

b. Below the text and the pattern, we list the characters of the text that are aligned with the last T of the pattern, along with the corresponding number of character comparisons (both successful and unsuccessful) and the shift size:

the text: TTATAGATCTCGTATTCTTTTATAGATCTCCTATTCTT
the pattern: TCCTATTCTT

T: 2 comparisons, shift 1
C: 1 comparison, shift 2
T: 2 comparisons, shift 1
A: 1 comparison, shift 5
T: 8 comparisons, shift 1
T: 3 comparisons, shift 1
T: 3 comparisons, shift 1
A: 1 comparison, shift 5
T: 2 comparisons, shift 1
C: 1 comparison, shift 2
C: 1 comparison, shift 2
T: 2 comparisons, shift 1
A: 1 comparison, shift 5
T: 10 comparisons to stop the successful search

3. a. For the pattern 00001, the shift table is

c	0	1
$t(c)$	1	5

The algorithm will make one unsuccessful comparison and then shift the pattern one position to the right on each of its trials:

```
0 0 0 0 0 0                                   0 0 0 0 0
0 0 0 0 1
  0 0 0 0 1
              etc.
                                              0 0 0 0 1
```

The total number of character comparisons will be $C = 1 \cdot 996 = 996$.

b. For the pattern `10000`, the shift table is

c	`0`	`1`
$t(c)$	1	4

The algorithm will make four successful and one unsuccessful comparison and then shift the pattern one position to the right on each of its trials:

```
0 0 0 0 0 0                                   0 0 0 0 0
1 0 0 0 0
  1 0 0 0 0 0
              etc.
                                              1 0 0 0 0
```

The total number of character comparisons will be $C = 5 \cdot 996 = 4980$.

c. For the pattern `01010`, the shift table is

c	`0`	`1`
$t(c)$	2	1

The algorithm will make one successful and one unsuccessful comparison and then shift the pattern two positions to the right on each of its trials:

```
0 0 0 0 0 0                                     0 0 0 0 0 0
0 1 0 1 0
    0 1 0 1 0
                    etc.
                                                0 1 0 1 0
```

The left end of the pattern in the trials will be aligned against the text's characters in positions 0, 2, 4, ..., 994, which is 498 trials. (We can also get this number by looking at the positions of the right end of the pattern. This leads to finding the largest integer k such that $4 + 2(k - 1) \leq 999$, which is $k = 498$.) Thus, the total number of character comparisons will be $C = 2 \cdot 498 = 996$.

4. a. The worst case: e.g., searching for the pattern $\underbrace{\texttt{10...0}}_{m-1}$ in the text of n 0's. $C_w = m(n-m+1)$.

   b. The best case: e.g., searching for the pattern $\underbrace{\texttt{0...0}}_{m}$ in the text of n 0's. $C_b = m$.

5. Yes: e.g., for the pattern $\underbrace{\texttt{10...0}}_{m-1}$ and the text $\underbrace{\texttt{0...0}}_{n}$, $C_{bf} = n-m+1$ while $C_{Horspool} = m(n-m+1)$.

6. We can shift the pattern exactly in the same manner as we would in the case of a mismatch, i.e., by the entry $t(c)$ in the shift table for the text's character c aligned against the last character of the pattern.

7. a. For the pattern 00001, the shift tables will be filled as follows:

   the bad-symbol table

c	0	1
$t_1(c)$	1	5

   .

   the good-suffix table

k	the pattern	d_2
1	0000**1**	5
2	000**01**	5
3	00**001**	5
4	0**0001**	5

   On each of its trials, the algorithm will make one unsuccessful comparison and then shift the pattern by $d_1 = \max\{t_1(0) - 0,\ 1\} = 1$ position to the right without consulting the good-suffix table:

```
0 0 0 0 0 0                                   0 0 0 0 0
0 0 0 0 1
  0 0 0 0 1
            etc.
                                              0 0 0 0 1
```

   The total number of character comparisons will be $C = 1 \cdot 996 = 996$.

   b. For the pattern 10000, the shift tables will be filled as follows:

   the bad-symbol table

$c.$	0	1
$t_1(c)$	1	4

   .

   the good-suffix table

k	the pattern	d_2
1	1000**0**	3
2	100**00**	2
3	10**000**	1
4	**10000**	5

   .

   On each of its trials, the algorithm will make four successful and one

unsuccessful comparison and then shift the pattern by the maximum of $d_1 = \max\{t_1(0) - 4,\ 1\} = 1$ and $d_2 = t_2(4) = 5$, i.e., by 5 characters to the right:

```
0 0 0 0 0 0 0 0 0 0                         0 0 0 0 0
1 0 0 0 0
          1 0 0 0 0
                          etc.
                                            1 0 0 0 0
```

The total number of character comparisons will be $C = 5 \cdot 200 = 1000$.

c. For the pattern `01010`, the shift tables will be filled as follows:

the bad-symbol table

c	0	1
$t_1(c)$	2	1

.

the good-suffix table

k	the pattern	d_2
1	0101**0**	4
2	010**10**	4
3	01**010**	2
4	0**1010**	2

.

On each trial, the algorithm will make one successful and one unsuccessful comparison. The shift's size will be computed as the maximum of $d_1 = \max\{t_1(0) - 1,\ 1\} = 1$ and $d_2 = t_2(1) = 4$, which is 4. If we count character positions starting with 0, the right end of the pattern in the trials will be aligned against the text's characters in positions 4, 8, 12, ..., with the last term in this arithmetic progression less than or equal to 999. This leads to finding the largest integer k such that $4 + 4(k - 1) \le 999$, which is $k = 249$.

```
0 0 0 0 0 0                                 0 0 0 0 0 0
0 1 0 1 0
        0 1 0 1 0
                    etc.
                                        0 1 0 1 0
```

Thus, the total number of character comparisons will be $C = 2 \cdot 249 = 498$.

8. a. Yes, the Boyer-Moore algorithm can get by with just the bad-symbol shift table.

   b. No: The bad-symbol table is necessary because it's the only one used by the algorithm if the first pair of characters does not match.

9. a. Horspool's algorithm can also compare the remaining $m - 1$ characters of the pattern from left to right because it shifts the pattern based only on the text's character aligned with the last character of the pattern.

b. The Boyer-Moore algorithm must compare the remaining $m - 1$ characters of the pattern from right to left because of the good-suffix shift table.

10. n/a

Exercises 7.3

1. For the input 30, 20, 56, 75, 31, 19 and hash function $h(K) = K \bmod 11$

   a. construct the open hash table.

   b. find the largest number of key comparisons in a successful search in this table.

   c. find the average number of key comparisons in a successful search in this table.

2. For the input 30, 20, 56, 75, 31, 19 and hash function $h(K) = K \bmod 11$

   a. construct the closed hash table.

   b. find the largest number of key comparisons in a successful search in this table.

   c. find the average number of key comparisons in a successful search in this table.

3. Why is it not a good idea for a hash function to depend on just one letter (say, the first one) of a natural language word?

4. Find the probability of all n keys being hashed to the same cell of a hash table of size m if the hash function distributes keys evenly among all the cells of the table.

5. ►The ***birthday paradox*** asks how many people should be in a room so that the chances are better than even that two of them will have the same birthday (month and day). Find the quite unexpected answer to this problem. What implication for hashing does this result have?

6. Answer the following questions for the separate-chaining version of hashing.

   a. Where would you insert keys if you knew that all the keys in the dictionary are distinct? Which dictionary operations, if any, would benefit from this modification?

   b. We could keep keys of the same linked list sorted. Which of the dictionary operations would benefit from this modification? How could we take advantage of this if all the keys stored in the entire table need to be sorted?

7. Explain how hashing can be applied to check whether all elements of a list are distinct. What is the time efficiency of this application?

8. Fill in the following table with the average-case efficiency classes for the five implementations of the ADT dictionary:

	unordered array	ordered array	binary search tree	separate chaining	linear probing
search					
insertion					
deletion					

9. We have discussed hashing in the context of techniques based on space-time tradeoffs. But it also takes advantage of another general strategy. Which one?

10. Write a computer program that uses hashing for the following problem. Given a natural language text, generate a list of distinct words with the number of occurrences of each word in the text. Insert appropriate counters in the program to compare the empirical efficiency of hashing with the corresponding theoretical results.

Hints to Exercises 7.3

1. Apply the open hashing (separate chaining) scheme to the input given as it is done in the chapter's text for another input (see Figure 7.5). Then compute the largest number and average number of comparisons for successful searches in the constructed table.

2. Apply the closed hashing (open addressing) scheme to the input given as it is done in the chapter's text for another input (see Figure 7.6). Then compute the largest number and average number of comparisons for successful searches in the constructed table.

3. How many different addresses can such a hash function produce? Would it distribute keys evenly?

4. The question is quite similar to computing the probability of having the same result in n throws of a fair die.

5. Find the probability that n people have different birthdays. As to the hashing connection, what hashing phenomenon deals with coincidences?

6. a. There is no need to insert a new key at the end of the linked list it is hashed to.

   b. Which operations are faster in a sorted linked list and why? For sorting, do we have to copy all elements in the nonempty lists in an array and then apply a general purpose sorting algorithm or is there a way to take advantage of the sorted order in each of the nonempty linked lists?

7. After you answer these questions, compare the efficiency of this algorithm with that of the brute-force algorithm (Section 2.3) and of the presorting-based algorithm (Section 6.1).

8. Consider this question as a mini-review: the answers are in Section 7.3 for the last two columns and in the appropriate sections of the book for the others. (Of course, you should use the best algorithms available.)

9. If you need to refresh you memory, check the book's table of contents.

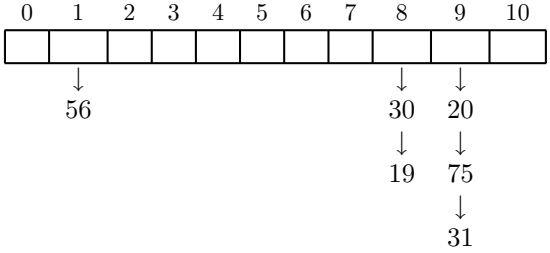
10. n/a

Solutions to Exercises 7.3

1. a.
   The list of keys: 30, 20, 56, 75, 31, 19
   The hash function: $h(K) = K \bmod 11$
   The hash addresses:

K	30	20	56	75	31	19
$h(K)$	8	9	1	9	9	8

   The open hash table:



   b. The largest number of key comparisons in a successful search in this table is 3 (in searching for $K = 31$).

   c. The average number of key comparisons in a successful search in this table, assuming that a search for each of the six keys is equally likely, is

$$\frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 2 + \frac{1}{6} \cdot 3 + \frac{1}{6} \cdot 2 = \frac{10}{6} \approx 1.7.$$

2. a.
   The list of keys: 30, 20, 56, 75, 31, 19
   The hash function: $h(K) = K \bmod 11$
   The hash addresses:

K	30	20	56	75	31	19
$h(K)$	8	9	1	9	9	8

0	1	2	3	4	5	6	7	8	9	10
								30		
								30	20	
	56							30	20	
	56							30	20	75
31	56							30	20	75
31	56	19						30	20	75

   b. The largest number of key comparisons in a successful search is 6 (when searching for $K = 19$).

   c. The average number of key comparisons in a successful search in this table, assuming that a search for each of the six keys is equally likely, is

$$\frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 2 + \frac{1}{6} \cdot 3 + \frac{1}{6} \cdot 6 = \frac{14}{6} \approx 2.3.$$

3. The number of different values of such a function would be obviously limited by the size of the alphabet. Besides, it is usually not the case that the probability of a word to start with a particular letter is the same for all the letters.

4. The probability of all n keys to be hashed to a particular address is equal to $\left(\frac{1}{m}\right)^n$. Since there are m different addresses, the answer is $\left(\frac{1}{m}\right)^n m = \frac{1}{m^{n-1}}$.

5. The probability of n people having different birthdays is $\frac{364}{365}\frac{363}{365}...\frac{365-(n-1)}{365}$. The smallest value of n for which this expression becomes less than 0.5 is 23. Sedgewick and Flajolet [SF96] give the following analytical solution to the problem:

$$(1-\frac{1}{M})(1-\frac{2}{M})...(1-\frac{n-1}{M}) \approx \frac{1}{2} \quad \text{where } M = 365.$$

Taking the natural logarithms of both sides yields

$$\ln(1-\frac{1}{M})(1-\frac{2}{M})...(1-\frac{n-1}{M}) \approx -\ln 2 \quad \text{or} \quad \sum_{k=1}^{n-1} \ln(1-\frac{k}{M}) \approx -\ln 2.$$

Using $\ln(1-x) \approx -x$, we obtain

$$\sum_{k=1}^{n-1} \frac{k}{M} \approx \ln 2 \quad \text{or} \quad \frac{(n-1)n}{2M} \approx \ln 2. \quad \text{Hence,} \quad n \approx \sqrt{2M \ln 2} \approx 22.5.$$

The implication for hashing is that we should expect collisions even if the size of a hash table is much larger (by more than a factor of 10) than the number of keys.

6. a. If all the keys are known to be distinct, a new key can always be inserted at the beginning of its linked list; this will make the insertion operation $\Theta(1)$. This will not change the efficiencies of search and deletion, however.

b. Searching in a sorted list can be stopped as soon as a key larger than the search key is encountered. Both deletion (that must follow a search) and insertion will benefit for the same reason. To sort a dictionary stored in linked lists of a hash table, we can merge the k nonempty lists to get the entire dictionary sorted. (This operation is called the *k-way merge*.) To do this efficiently, it's convenient to arrange the current first elements of the lists in a min-heap.

7. Insert successive elements of the list in a hash table until a matching element is encountered or the list is exhausted. The worst-case efficiency will be in $\Theta(n^2)$: all n distinct keys are hashed to the same address so that the number of key comparisons will be $\sum_{i=1}^{n}(i-1) \in \Theta(n^2)$. The average-case efficiency, with keys distributed about evenly so that searching for each of them takes $\Theta(1)$ time, will be in $\Theta(n)$.

8.

	unordered array	ordered array	binary search tree	separate chaining	linear probing
search	$\Theta(n)$	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$	$\Theta(1)$
insertion	$\Theta(1)$	$\Theta(n)$	$\Theta(\log n)$	$\Theta(1)$	$\Theta(1)$
deletion	$\Theta(1)$	$\Theta(n)$	$\Theta(\log n)$	$\Theta(1)$	$\Theta(1)$

9. Representation change—one of the three varieties of transform-and-conquer.

10. n/a

Exercises 7.4

1. Give examples of using an index in real-life applications that do not involve computers.

2. a. Prove the equality

$$1 + \sum_{i=1}^{h-1} 2\lceil m/2 \rceil^{i-1}(\lceil m/2 \rceil - 1) + 2\lceil m/2 \rceil^{h-1} = 4\lceil m/2 \rceil^{h-1} - 1$$

   that was used in the derivation of upper bound (7.7) for the height of a B-tree.

   b. Complete the derivation of inequality (7.7).

3. Find the minimum order of the B-tree that guarantees that the number of disk accesses in searching in a file of 100 million records does not exceed 3. Assume that the root's page is stored in main memory.

4. Draw the B-tree obtained after inserting 30 and then 31 in the B-tree in Figure 7.8. Assume that a leaf cannot contain more than three items.

5. Outline an algorithm for finding the largest key in a B-tree.

6. a. A ***top-down 2-3-4 tree*** is a B-tree of order 4 with the following modification of the *insert* operation. Whenever a search for a leaf for a new key encounters a full node (i.e., a node with three keys), the node is split into two nodes by sending its middle key to the node's parent (or, if the full node happens to be the root, the new root for the middle key is created). Construct a top-down 2-3-4 tree by inserting the following list of keys in the initially empty tree:

$$10,\ 6,\ 15,\ 31,\ 20,\ 27,\ 50,\ 44,\ 18.$$

   b. What is the principal advantage of this insertion procedure compared with the one described for 2-3 trees in Section 6.3? What is its disadvantage?

7. a. ▷ Write a program implementing a key insertion algorithm in a B-tree.

   b. ▶ Write a program for visualization of a key insertion algorithm in a B-tree.

Hints to Exercises 7.4

1. Thinking about searching for information should lead to a variety of examples.

2. a. Use the standard rules of sum manipulation and, in particular, the geometric series formula.

   b. You will need to take the logarithms base $\lceil m/2 \rceil$ in your derivation.

3. Find this value from the inequality in the text that provides the upper-bound of the B-tree's height.

4. Follow the insertion algorithm outlined in this section.

5. The algorithm is suggested by the definition of the B-tree.

6. a. Just follow the description of the algorithm given in the statement of the problem. Note that a new key is always inserted in a leaf and that full nodes are always split on the way down, even though the leaf for the new key may have a room for it.

   b. Can a split of a full node cause a cascade of splits through the chain of its ancestors? Can we get a taller search tree than necessary?

7. n/a

Solutions to Exercises 7.4

1. Here are a few common examples of using an index: labeling drawers of a file cabinet with, say, a range of letters; an index of a book's terms indicating the page or pages on which the term is defined or mentioned; marking a range of pages in an address book, a dictionary; or an encyclopedia; marking a page of a telephone book or a dictionary with the first and last entry on the page; indexing areas of a geographic map by dividing the map into square regions.

2. a.

$$\begin{aligned} & 1 + \sum_{i=1}^{h-1} 2\lceil m/2 \rceil^{i-1}(\lceil m/2 \rceil - 1) + 2\lceil m/2 \rceil^{h-1} \\ = \;& 1 + 2(\lceil m/2 \rceil - 1) \sum_{i=1}^{h-1} \lceil m/2 \rceil^{i-1} + 2\lceil m/2 \rceil^{h-1} \\ = \;& 1 + 2(\lceil m/2 \rceil - 1) \sum_{j=0}^{h-2} \lceil m/2 \rceil^{j} + 2\lceil m/2 \rceil^{h-1} \\ = \;& 1 + 2(\lceil m/2 \rceil - 1) \frac{\lceil m/2 \rceil^{h-1} - 1}{(\lceil m/2 \rceil - 1)} + 2\lceil m/2 \rceil^{h-1} \\ = \;& 1 + 2\lceil m/2 \rceil^{h-1} - 2 + 2\lceil m/2 \rceil^{h-1} \\ = \;& 4\lceil m/2 \rceil^{h-1} - 1. \end{aligned}$$

   b. The inequality

$$n \geq 4\lceil m/2 \rceil^{h-1} - 1$$

   is equivalent to

$$\frac{n+1}{4} \geq \lceil m/2 \rceil^{h-1}.$$

   Taking the logarithms base $\lceil m/2 \rceil$ of both hand sides yields

$$\log_{\lceil m/2 \rceil} \frac{n+1}{4} \geq \log_{\lceil m/2 \rceil} \lceil m/2 \rceil^{h-1}$$

   or

$$\log_{\lceil m/2 \rceil} \frac{n+1}{4} \geq h - 1.$$

   Hence,

$$h \leq \log_{\lceil m/2 \rceil} \frac{n+1}{4} + 1$$

   or, since h is an integer,

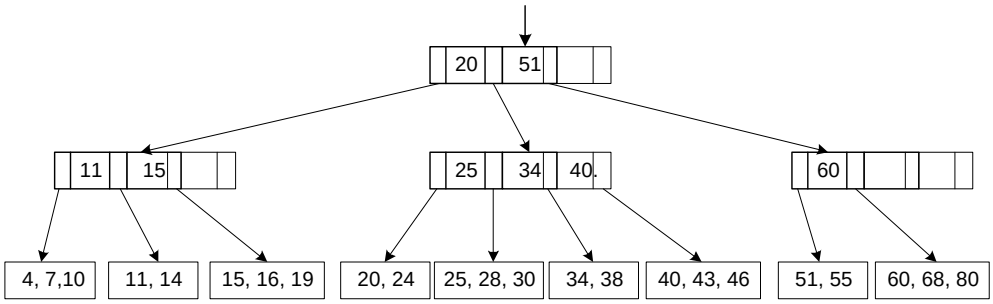
$$h \leq \lfloor \log_{\lceil m/2 \rceil} \frac{n+1}{4} \rfloor + 1.$$

3. If the tree's root is stored in main memory, the number of disk accesses will be equal to the number of the levels minus 1, which is exactly the height of the tree. So, we need to find the smallest value of the order m so that the height of the B-tree with $n = 10^8$ keys does not exceed 3. Using the upper bound of the B-tree's height, we obtain the following inequality

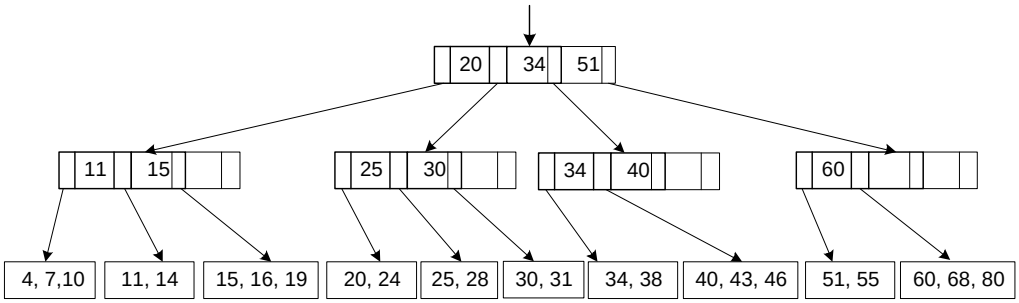
$$\lfloor \log_{\lceil m/2 \rceil} \frac{n+1}{4} \rfloor + 1 \le 3 \quad \text{or} \quad \lfloor \log_{\lceil m/2 \rceil} \frac{n+1}{4} \rfloor \le 2.$$

By "trial and error," we can find that the smallest value of m that satisfies this inequality is 585.

4. Since there is enough room for 30 in the leaf for it, the resulting B-tree will look as follows



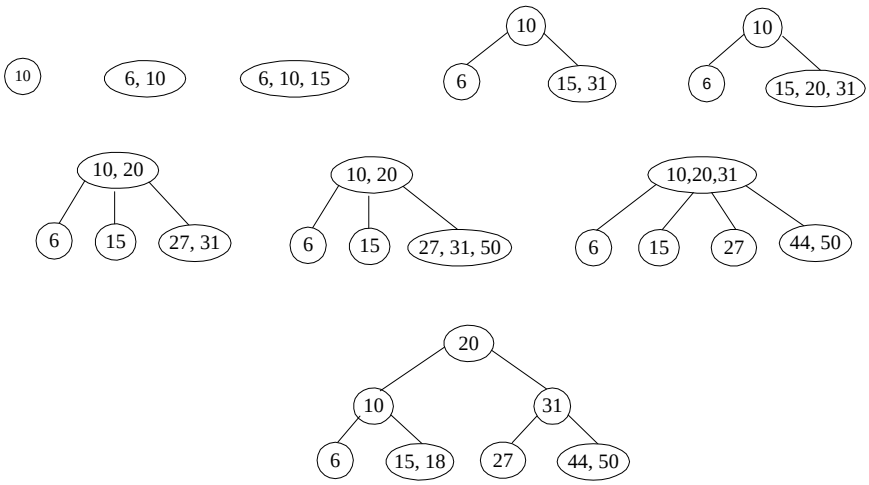
Inserting 31 will require the leaf's split and then its parent's split:



5. Starting at the root, follow the chain of the rightmost pointers to the (rightmost) leaf. The largest key is the last key in that leaf.

6. a. Constructing a top-down 2-3-4 tree by inserting the following list of keys in the initially empty tree:

10, 6, 15, 31, 20, 27, 50, 44, 18.



b. The principal advantage of splitting full nodes (4-nodes with 3 keys) on a way down during insertion of a new key lies in the fact that if the appropriate leaf turns out to be full, its split will never cause a chain reaction of splits because the leaf's parent will always have a room for an extra key. (If the parent is full before the insertion, it is split before the leaf is reached.) This is not the case for the insertion algorithm employed for 2-3 trees (see Section 6.3).

The disadvantage of splitting full nodes on the way down lies in the fact that it can lead to a taller tree than necessary. For the list of part (a), for example, the tree before the last one had a room for key 18 in the leaf containing key 15 and therefore didn't require a split executed by the top-down insertion.

7. n/a