1.4.2 Modify ThreeSum to work properly even when the int values are so large that adding two of them might cause overﬂow

1.4.4 Develop a table like the one on page 181 for TwoSum.

1.4.5 Give tilde approximations for the following quantities:

1.4.6 Give the order of growth （增长数量级）(as a function（函数，功能） of N ) of the running times of each of the following code fragments:

1.4.15 Faster 3-sum. As a warmup, develop an implementation（实现） TwoSumFaster that uses a linear algorithm（线性） to count the pairs that sum to zero after the array is sorted (instead of the binary-search-based linearithmic（线性对数） algorithm). Then apply a similar idea to develop a quadratic （平方）algorithm for the 3-sum problem.

1.5.1 Show the contents of the id[] array and the number of times the array is accessed(访问) for each input pair(一对) when you use quick-ﬁnd for the sequence（序列） 9-0 3-4 5-8 7-2 2-1 5-7 0-3 4-2.

1.5.2 Do Exercise 1.5.1, but use quick-union (page 224). In addition, draw the forest（森林） of trees represented by the id[] array after each input pair is processed.（处理）

1.5.3 Do Exercise 1.5.1, but use weighted quick-union (page 228).

1.5.8 Give a counterexample（反例） that shows why this intuitive（直接） implementation of union() for quick-ﬁnd is not correct:

public void union(int p, int q) {

if (connected(p, q)) return;

for (int i = 0; i < id.length; i++)

if (id[i] == id[p]) id[i] = id[q];

count--;

}

2.2.2 Give traces, in the style of the trace given with Algorithm 2.4, showing how the keys E A S Y Q U E S T I O N are sorted with top-down mergesort.

2.2.3 Answer Exercise 2.2.2 for bottom-up mergesort.

2.2.4 Does the abstract in-place merge produce proper output if and only if（当且仅当） the two input subarrays are in sorted order? Prove your answer, or provide a counterexample.

2.2.5 Give the sequence of subarray sizes in the merges performed by both the topdown and the bottom-up mergesort algorithms, for N = 39.

2.2.8 Suppose that Algorithm 2.4 is modiﬁed to skip the call on merge() whenever a[mid] <= a[mid+1]. Prove that the number of compares used to mergesort a sorted array is linear.

2.3.1 Show, in the style of the trace given with partition(), how that method patitions the array E A S Y Q U E S T I O N.

2.3.2 Show, in the style of the quicksort trace given in this section, how quicksort sorts the array E A S Y Q U E S T I O N (for the purposes of this exercise, ignore the initial shufﬂe).

2.3.3 What is the maximum number of times during the execution of Quick.sort() that the largest item can be exchanged, for an array of length N ?

2.3.4 Suppose that the initial random shufﬂe is omitted. Give six arrays of ten elements for which Quick.sort() uses the worst-case number of compares.

2.3.5 Give a code fragment that sorts an array that is known to consist of items having just two distinct keys.

2.4.5 Give the heap that results when the keys E A S Y Q U E S T I O N are inserted in that order into an initially empty max-oriented heap.

2.4.7 The largest item in a heap must appear in position 1, and the second largest must be in position 2 or position 3. Give the list of positions in a heap of size 31 where the kth largest (i) can appear, and (ii) cannot appear, for k=2, 3, 4 (assuming the values to be distinct).

2.4.9 Draw all of the different heaps that can be made from the ﬁve keys A B C D E, then draw all of the different heaps that can be made from the ﬁve keys A A A B B.

2.4.18 In MaxPQ, suppose that a client calls insert() with an item that is larger than all items in the queue, and then immediately calls delMax(). Assume that there are no duplicate keys. Is the resulting heap identical to the heap as it was before these operations? Answer the same question for two insert() operations (the ﬁrst with a key larger than all keys in the queue and the second for a key larger than that one) followed by two delMax() operations.

3.1.10 Give a trace of the process of inserting the keys E A S Y Q U E S T I O N into an initially empty table using SequentialSearchST. How many compares are involved?

3.1.11 Give a trace of the process of inserting the keys E A S Y Q U E S T I O N into an initially empty table using BinarySearchST. How many compares are involved?

3.2.1 Draw the BST that results when you insert the keys E A S Y Q U E S T I O N, in that order (associating the value i with the ith key, as per the convention in the text) into an initially empty tree. How many compares are needed to build the tree?

3.2.4 Suppose that a certain BST has keys that are integers between 1 and 10, and we search for 5. Which sequence below cannot be the sequence of keys examined（检查）?

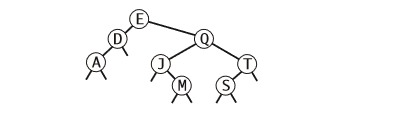
a. 10, 9, 8, 7, 6, 5

b. 4, 10, 8, 7, 53

c. 1, 10, 2, 9, 3, 8, 4, 7, 6, 5

d. 2, 7, 3, 8, 4, 5 e. 1, 2, 10, 4, 8, 5

3.2.15 Give the sequences of nodes examined when the methods in BST are used to compute each of the following quantities for the tree drawn at right.



a. floor("Q")

b. select(5)

c. ceiling("Q")

d. rank("J")

3.2.18 Draw the sequence of BSTs that results when you delete the keys from the tree of Exercise 3.2.1, one by one, in alphabetical order.

4.3.3 Show that if a graph’s edges all have distinct weights, the MST is unique.

4.3.6 Give the MST of the weighted graph obtained by deleting vertex(顶点) 7 from tinyEWG.txt (see page 604).

4.3.13 Give a counterexample that shows why the following strategy does not necessarily（不一定）ﬁnd the MST: ‘Start with any（任意）vertex as a single-vertex MST, then add V-1 edges to it, always taking next a min-weight edge incident to（依附于） the vertex most recently added to the MST.’

4.3.18 Give traces that show the process of computing the MST of the graph deﬁned in Exercise 4.3.6 with the lazy version of Prim’s algorithm, the eager version of Prim’s algorithm, and Kruskal’s algorithm.

4.4.1 True or false: Adding a constant（常数） to every edge weight does not change the solution to the single-source（单点） shortest-paths problem.

4.4.4 Draw the SPT for source 0 of the edge-weighted digraph（加权有向） obtained by deleting vertex 7 from tinyEWD.txt (see page 644), and give the parent-link representation of the SPT. Answer the question for the same graph with all edge reversed.

4.4.6 Give a trace that shows the process of computing the SPT of the digraph deﬁned in Exercise 4.4.5 with the eager version of Dijkstra’s algorithm.

4.4.9 The table below, from an old published road map, purports to give the length of the shortest routes connecting the cities. It contains an error. Correct the table. Also, add a table that shows how to achieve the shortest routes

