Appendix

Solutions to Selected Exercises

1.2

j	1	2	3	4	5	6	7	8	9	10	11
$S[j]$	A	B	R	A	C	A	D	A	B	R	A
$\Delta_2[j]$	17	16	15	14	13	12	11	13	12	4	1

j	1	2	3	4	5	6	7	8	9	10	11	12
$S[j]$	A	A	B	A	A	A	B	A	A	B	A	A
$\Delta_2[j]$	18	17	16	15	14	13	8	14	13	8	2	2

1.3 The last k entries of the Δ_2 array of S all contain the value k.

1.4 Text: AAAAAAAAABBC, Pattern: BCBCBC

1.6 Define an array B and store in $B[i]$ the sum of the first i elements of A. Formally: $B[0] = 0$, and

$$B[i] = B[i-1] + A[i] \qquad \text{for } 1 \le i \le n.$$

Each element of the array B can thus be evaluated in $O(1)$, and we have

$$sum(i, j) = B[j] - B[i-1].$$

1.7

	V_1	V_2	V_3	V_4
A_1	yes	no	yes	yes
A_2	no	yes	no	no
A_3	no	no	no	no
A_4	no	no	no	no

2.1 The idea is to scan the list with two pointers, one advancing at twice or three times the pace of the other. Formally:

```
p ← head      q ← head
while next(q) ≠ NIL
    p ← next(p)
    q ← next(next(next(p)))
q ← p
while next(q) ≠ NIL
    print value(p)
    p ← next(p)
    q ← next(next(p))
```

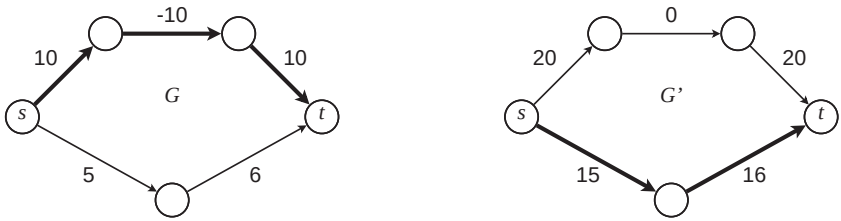
Using the same technique, there is a cycle in the list if and only if p and q will point to the same element in some iteration.

2.3 (a) Use binary search.
(b) Use, alternatingly, one step of binary search, and one step of a linear search from the beginning of the array.
(c) Search from the beginning in exponentially increasing steps, that is, check at indices $1, 2, 4, \ldots, 2^i$ until an index i is found for which $A[2^i]$ is positive but $A[\min(n, 2^{i+1})]$ is negative. Then use binary search on the range $[2^i, \min(n, 2^{i+1})]$.

2.4 The parking lot is in fact a stack. If the input order is 1, 3, 2, this cannot be rectified by the use of a stack. Indeed, the cars 3 and 2 have to enter the stack, since car 1 has to be the first to get on the highway. But then car 3 will precede car 2.

2.8 (b) $f(n) = 2^{\sqrt{\log n}}$
(c) $T(n) = \log^* n$ (see Section 8.2.3).

3.1 The following figure gives a counter example. There are two paths from s to t in the graph G: the upper path has weight 10, and the lower path weight 11. Adding 10 to the weight of each edge yields the graph G' on the right side. Now it is the lower path that has the smaller weight. The shortest paths are emphasized. The reason for the failure of this procedure is that not all the paths consist of the same number of edges, so adding a constant weight to each edge does not have the same effect on all the paths. For instance, the total weight of the upper path was increased by 30, and that of the lower path only by 20.



Counterexample for Exercise 3.1.

3.2 (a) Set the weight of all edges touching vertex T to infinity.

(b) Let ε be the smallest difference between different weights (if all weights are integers, then $\varepsilon = 1$), and decrease the weights of all the edges corresponding to airline X by $\frac{\varepsilon}{n}$, where n is the number of vertices. There are at most $n-1$ edges in a path, so the choice of the shortest paths is not affected by this amendment, but it gives priority to the edges with reduced weights.

(c) Give the same weight to all edges.

(d) There is an edge (fr_a, to_a) with weight 0 for every flight a, where fr_a and to_a are vertices corresponding to airports. For each airport, there is an edge (to_a, fr_b) from every incoming flight a to every outgoing flight b. If a and b are operated by the same airline, the weight of the edge is 0, otherwise 1.

(e) Use the same graph as in point (d), but change the weight of the edge (fr_a, to_a) to the length of the flight a, and the weight of (to_a, fr_a) to the waiting time at the airport for the connection between a and b.

3.3 Let $\mathcal{P}$ be the set of all the paths from s to t. We wish to maximize the probability of getting from s to t without corruption, that is, find

$$\max_{Q \in \mathcal{P}} \prod_{(a,b) \in Q} p(a, b).$$

Applying log and multiplying by -1, this maximization is equivalent to finding

$$\min_{Q \in \mathcal{P}} \sum_{(a,b) \in Q} -\log p(a, b).$$

Therefore, change the weight of the edge (a, b) to $-\log p(a, b)$; now the problem can be solved by applying algorithm A.

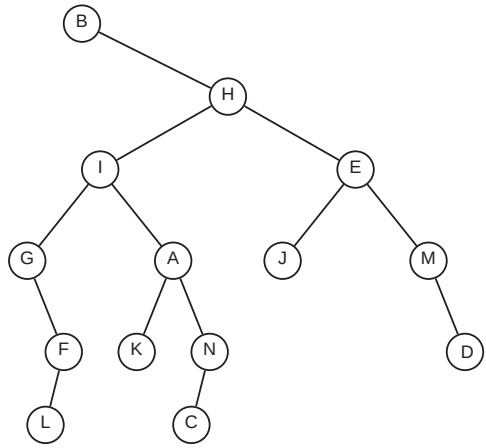
3.4 Define a graph $G = (V, E)$. The vertices correspond to countries, and there is an edge $(x, y) \in E$ if and only if x and y have a common

border. The problem is to find a color assignment to the vertices, that is, a function $f : V \to \{1, 2, \ldots, k\}$, such that

$$(x, y) \in E \;\rightarrow\; f(x) \neq f(y).$$

It should be noted that while every map can thus be transformed into a graph, it is not true that every graph corresponds to a map. The subset of graphs corresponding to maps is called *planar graphs*. Such graphs can be drawn in a way that avoids the crossing of edges. For example, the full graph with four vertices is planar, but the full graph with five vertices is not. The four-color problem relates to planar graphs. For nonplanar graphs, the question whether the vertices can be colored by four colors is apparently also very difficult.

4.2 The following figure displays the only binary tree with the given traversals.



Tree with in-order traversal B G L F I K A C N H J E M D and post-order traversal L F G K C N A I J D M E H B.

It is not always possible to determine the tree on the basis of its pre-order and post-order traversals alone. The two trees in the following figure have both A B as pre-order and B A as post-order traversal.

4.5 Use in-order traversals of the trees to get the ordered lists of their elements. Then merge the lists and store the merged list, in order, in an

array A at positions $A[1]$ to $A[m-1]$, where $m = 2^n = k + \ell + 1$. The full binary search tree storing the merged list has value $A[\frac{m}{2}]$ as its root, the two children of the root hold the elements with indices $\frac{m}{4}$ and $\frac{3m}{4}$, and in general, the 2^i elements on level i, for $i = 0, 1, \ldots, n-1$, are those with indices

$$2^{n-1-i} + j\,2^{n-i} \qquad \text{for } 0 \le j < 2^i.$$

5.2 Note that we need a lower bound for $N(k)$:

$$\begin{aligned} N(k) &= N(k-1) + N(k-2) + 1 \\ &> 2N(k-2) > 4N(k-4) > \cdots > 2^i N(k-2i) \\ &= 2^{k/2} = \sqrt{2}^{\,k} = 1.414^k. \end{aligned}$$

5.6(a) $\begin{pmatrix} F_i \\ F_{i-1} \end{pmatrix} = A^{i-1} \begin{pmatrix} F_1 \\ F_0 \end{pmatrix}$.

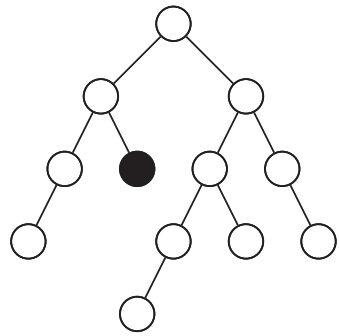
(b) $A^{16} = \left(A^8\right)^2$.

(c) $A^8 = \left((A^2)^2\right)^2$.

(d) $A^5 = A^4 \times A$. $A^{12} = A^8 \times A^4$. For general k, write k in its standard binary representation as a sum of powers of 2.

(e) If k is a power of 2, one can evaluate A^k in $\log k$ steps. This is called *repeated squaring*. For general k, one first prepares $B[i] = A^{2^i}$ for $0 \le i < \log k$, using $B[i] = (B[i-1])^2$, and then multiplies the elements $B[i]$ with indices corresponding to the positions of the 1-bits in the binary representation of k, overall $O(\log k)$.

5.7 One such tree is given in the following figure. It is a minimal AVL tree of depth 4. Deleting the black node requires 2 rotations. In general, deleting one of the leaves of a minimal AVL tree of depth $k+2$ may lead to k rotations.



Minimal AVL tree requiring two rotations for deleting the black node.

6.1 (a) No. The insertion may imply a node split, but the deletion does not necessarily then imply a fusion: it might be possible to do a balancing step with the other sibling node (not the one created by the split, since it is also only about half filled).

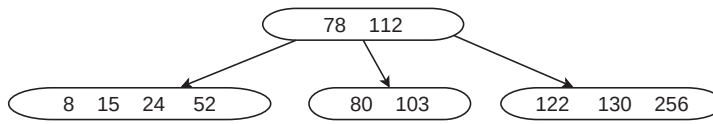
(b) No again. An element x which is not in one of the leaf nodes is replaced by its successor. Re-inserting x will place it in a leaf, and it might remain there if there are no splits.

6.2 First extract the maximal element of A or the minimal element of B and call it s; it will serve as a separator. Denote the root nodes by R_A and R_B, respectively.

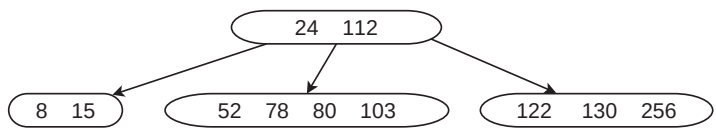
If A and B have the same height h, create a new root R containing only s, and let R_A and R_B be the children of R. Perform a balancing step with R, R_A and R_B (which, depending on the number of elements in the nodes, may lead to merging the 3 nodes into one new root).

If the height h' of B is less than h, let D be the rightmost node of A on level $h - h'$, and let E be the parent node of D. Insert s as the rightmost element in E and add R_B as the rightmost child of E. Now perform a balancing step with E, D and R_B; this may involve adding elements to R_B, or splitting E, which in turn may trigger more updates.

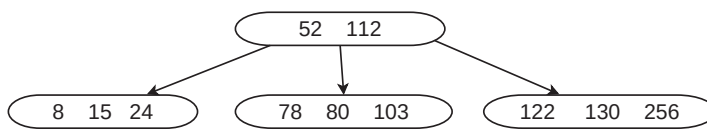
6.3 (a)



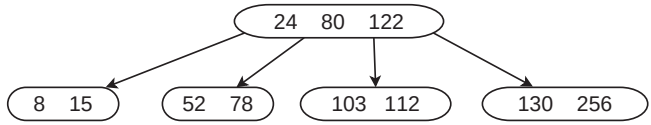
(b)



(c)



(d)



(e) If the order of the tree is 4, the deletion of 40 does not trigger any change in the structure, since the number of elements in a node should then be between 1 and 3.

7.1 (a) The depths of both A and B are $k - 1$.

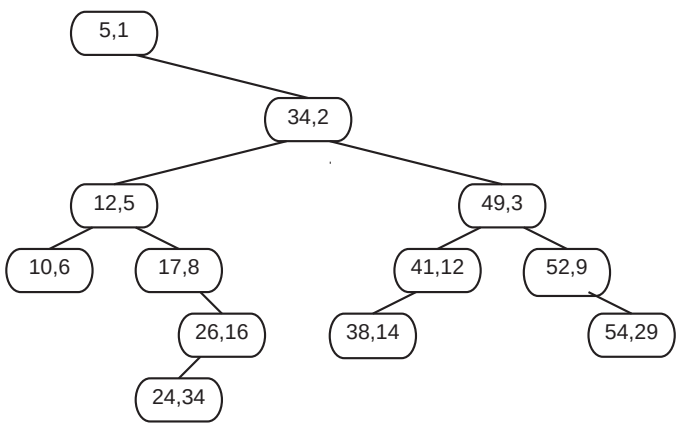
(b) The depth of C will be k.

(c) Extract the last element of B (rightmost on lowest level) and call it x. Create a new binary tree with x as root and A and B as subtrees. Apply heapify on x. Complexity $O(k) = O(\log m)$.

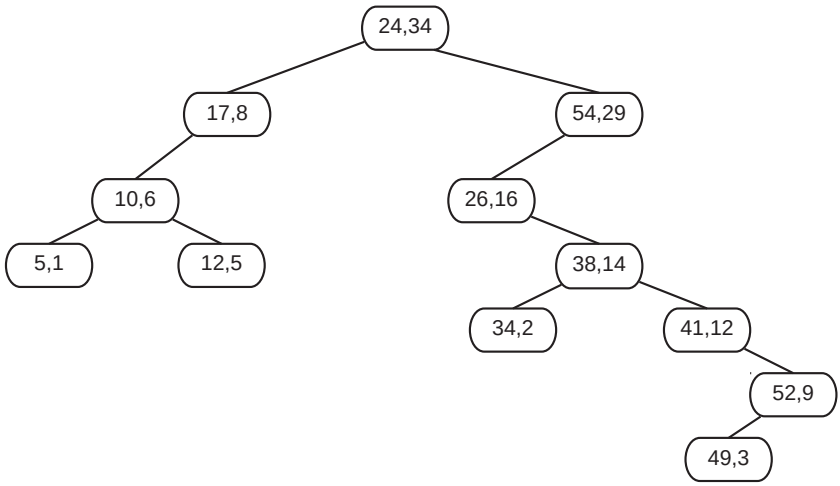
7.2 (a) Correctness does not change, the $n/2$ first iterations are just redundant, so it is less efficient.

(b) Does not work. Counterexample: a vector with elements 5, 4, 3, 7.

7.4 (a) The search-min-heap is:



The search-max-heap is:



(b) and (c) For the search-min-heap: find the element (s_0, h_0) with minimal h-value and make it the root of the tree. Partition the rest of the set into L, those having smaller s values than s_0, and R, those with larger s values than s_0. These will be the left and right subtrees, which are constructed recursively.

8.2 Interestingly, the bound will still be logarithmic.

8.3 (a) Sorting the edges, $E \log E$, applying BFS or DFS on a graph with at most V edges in each of the E iteration, for a total of

$$E \log E + EV = O(EV).$$

(b)

$$E \log E + E \log V = O(E \log E).$$

(c) The connected components in the Union-Find forest are trees and are the same components (contain the same nodes) as those in Kruskal's algorithm, but the trees may differ.

9.1 $h(X) = X \bmod 9$. Let $X = \sum_{i=0}^{r} d_i 10^i$, so that $d_r d_{r-1} \cdots d_2 d_1 d_0$ is the decimal representation of X. But $h(10^i) = 1$ for all i, and we get that $h(X) = h(\sum_{i=0}^{r} d_i 10^i) = h(\sum_{i=0}^{r} d_i)$ is just the sum of the digits.

9.4 No. The probability of choosing x in $[0, \frac{1}{2})$ is $\frac{1}{2}$, but then $x^2 \in [0, \frac{1}{4})$, so the lower quarter of the table has a higher chance of being chosen. The

distribution is therefore not uniform, so the function is not suitable as a hash function.

10.1 Denote the elements by A, B, C, D, E. Compare A with B and C with D. Without loss of generality, $A > B$ and $C > D$. Compare A and C and assume $A > C$. Insert E into the sorted list $A > C > D$, using binary search. That is, compare E to C, and if it is smaller, compare with A, otherwise, compare with D. This takes 2 comparisons. This yields one of 4 possible orderings: $EACD$, $AECD$, $ACED$, or $ACDE$. We know already that B is smaller than A, so the sort is completed by inserting B in the correct place to the right of A for any of the preceding four orderings. That is, B has to be inserted into CD, ECD, CED, or CDE. This can again be done in two comparisons by binary search.

10.2(a) Each weighing has three possible outcomes, the number of possible answers is 24 (12 balls times 2 possible choices for each), so the lower bound of comparisons is $\ell = \lceil \log_3 24 \rceil = 3$.

(b) (1) Compare 4 against 4 balls. (1a) If they are equal, we know that the different ball is in the remaining 4. (2) Compare 3 of the remaining with 3 good balls. (2a) If they are not equal we know already if the different ball is lighter or heavier, so (3) we choose 2 of them and compare them 1 to 1. (3a) If they are equal, the different ball is the remaining one, otherwise (3b) we see which one is the different one. (2b) If the second comparison gave equality, the different ball is the only one left out. So we use (3) one test to see if it is lighter or heavier.

(1b) If in the first comparison we got inequality, we know that the remaining 4 balls are good. Denote the 12 balls as 0000, LLLL, HHHH, according to whether we know if they are good, or possibly light or possibly heavy. (2) The second weighing will be LLLH against 000L, leaving HHH0 aside. (2a) If there is equality, the different ball is among HHH0, so it must be one of the HHH and it must be heavy. (3) Test H against H to find the different one, as earlier. (2b) If LLLH is lighter than 000L, the different ball is one of the 3 L balls on the left, so we need only one more test to find which. (2c) If LLLH is heavier than 000L, either the H on the left is heavier, or the L on the right is lighter. Compare one of them against a 0 ball.

(c) $\ell = \lceil \log_3 26 \rceil = 3$.

(d) If we start by weighing at most 4 against 4 and have equality, we are left with at least 5 balls, and thus 10 possibilities, which cannot be resolved in 2 weighings that have only 9 possible outcomes. If we start by weighing at least 5 against 5 and get inequality, we also have at least 10 possibilities.

10.3 Suppose the probability for getting an input that is already sorted is $1 - \frac{1}{n \log n}$. Then it is worth checking whether the input is sorted by a linear scan, and otherwise apply some other sort (e.g., mergesort). The expected complexity is then

$$\left(1 - \frac{1}{n \log n}\right) O(n) + \frac{1}{n \log n} O(n \log n) = O(n).$$

10.5 If a search tree can be built in less than order of $n \log n$, and in-order traversal of the tree, which can be performed in $O(n)$, would give the elements in sorted order in time less than $n \log n + n$, contradicting the lower bound $\Omega(n \log n)$ for comparison based sorting.

10.6 If $K = 2^m$, find the median and partition into two sets by comparing to the median, in time cn, for some constant $c > 1$. Then do the same for each half, in time $c\frac{n}{2} + c\frac{n}{2} = cn$ again. Then repeat for each quarter, etc., in $\log K$ iterations of $O(n)$ each.

If K is not a power of 2, let $L = 2^{\lceil \log K \rceil}$ be the next highest power of 2. Add $\left(\frac{L-K}{K}\right)n$ dummy elements of value ∞. Partitioning the $n + \left(\frac{L-K}{K}\right)n = \left(\frac{L}{K}\right)n$ elements into L subsets of equal size $\frac{n}{K}$ effectively partitions the n real elements into K subsets of equal size.

10.7 The recurrence for $T(n)$ would then be

$$T(n) \leq T\left(\frac{n}{2}\right) + T\left(\frac{3}{4}n\right) + cn$$

instead of eq. (10.17). This is the case $\alpha_1 + \alpha_2 = \frac{1}{2} + \frac{3}{4} > 1$, so $T(n)$ would not be linear in n.

11.1 (a) For the Fibonacci code, the number n_j of code words of length j is $F(j-1)$, for $j \geq 2$.

(b) Denote the sum $\sum_{j=2}^{\infty} n_j 2^{-j} = \sum_{j=2}^{\infty} F(j-1)2^{-j}$ by A. We the can derive the following equation:

$$\begin{aligned} A &= F(1)2^{-2} + \sum_{j=3}^{\infty} F(j-1)2^{-j} \\ &= \frac{1}{4} + \sum_{j=3}^{\infty} \big(F(j-2) + F(j-3)\big)2^{-j} \\ &= \frac{1}{4} + \sum_{j=2}^{\infty} \frac{1}{2} F(j-1)2^{-j} + \sum_{j=1}^{\infty} \frac{1}{4} F(j-1)2^{-j} \\ &= \frac{1}{4} + \frac{1}{2}A + \frac{1}{4}A, \end{aligned}$$

which implies $A = 1$.

11.3(a) 10, 111, 011, 001, 000, 1101, 1100, 0101, 0100.

(b) The example shows that at least one such code exists. Let us show that for every given affix code, there is another affix code with a larger number of code words. Let $A = \{a_1, \ldots, a_n\}$ be an affix code. Consider the set $B = \{b_1, \ldots, b_{2n}\}$ defined by $b_{2i} = a_i 0, b_{2i-1} = a_i 1$ for $1 \leq i \leq n$. B is an affix code with a code word set twice as large as A.

11.5 Add the XOR of all the bits as additional parity bit X. Denote the vector of Hamming parity bits by B. If $X = 0$ and $B = 0$, then there is no error. If $X = 0$ and $B \neq 0$, there is a double error, which cannot be corrected. If $X = 1$, then there is a single error at address B.

11.7 The number of comparisons to merge the n sets is $\sum_{i=1}^{n} n_i \ell_i$, where n_i is the number of times the sequence A_i participates in a merge. This is exactly the same minimization problem as Huffman's. The optimal solution is therefore to repeatedly merge the two shortest sequences.