Algorithm Homework and Test Problems

Steven S. Skiena
Department of Computer Science
State University of New York
Stony Brook, NY 11794-4400
skiena@cs.sunysb.edu

August 4, 2000

This is the same list of problems as distributed in Fall 1999, although of course the actual homework and exam problems are subject to change.

1 Mathematical Preliminaries and Analysis

In this section, we should have all the problems dealing with the big oh notation, recurrence relations, logarithms, etc.

1.1 Summations

1.2 Big Oh

1. Graph the following expressions. For each expression, state for which values of $n$ that expression is the most efficient.

For the following functions:

\[ 4n^2 \quad \log_3 n \quad 3^n \quad 20n \quad 2 \quad \log_2 n \quad n^{2/3} \]

(a) Graph all seven functions on a single plot.

(b) Partition the integers $n \geq 1$ into groups such that all the integers in a group have the same function which yields the smallest values of $f(n)$.

(c) Partition the integers $n \geq 1$ into groups such that all the integers in a group have the same function which yields the largest values of $f(n)$.

(d) Arrange the expressions by asymptotic growth rate from slowest to fastest.

Source: Shaffer, chapter 3 problem 3.1
2. Show that for any real constants $a$ and $b$, $b > 0$,
\[(n + a)^b = \Theta(n^b)\]

Source: CLR, problem 2.1-2 (*)

3. (a) Is $2^{n+1} = O(2^n)$?
(b) Is $2^{2n} = O(2^n)$?

Source: CLR, problem 2.1-4 (*)

4. For each of the following, compute how large a problem instance do you need before algorithm $A$ is faster than algorithm $B$. How much time do the algorithms take on that instance?

(a) Algorithm $A$ takes $n^2$ days to solve a problem of size $n$. Algorithm $B$ takes $n^3$ seconds on the same problem.
(b) Algorithm $A$ takes $n^2$ days to solve a problem of size $n$. Algorithm $B$ takes $2^n$ seconds on the same problem.

Source: Brassard and Bratley, problem 2.7 and 2.8

5. Let $P$ be a problem. The worst-case time complexity of $P$ is $O(n^2)$. The worst-case time complexity of $P$ is also $\Omega(n \log n)$. Let $A$ be an algorithm that solves $P$. Which subset of the following statements are consistent with this information about the complexity of $P$?

- $A$ has worst-case time complexity $O(n^2)$.
- $A$ has worst-case time complexity $O(n^{3/2})$.
- $A$ has worst-case time complexity $O(n)$.
- $A$ has worst-case time complexity $\Theta(n^2)$.
- $A$ has worst-case time complexity $\Theta(n^3)$.

Source: Skiena, problem 1-1

6. Suppose that an algorithm $A$ runs in worst-case time $f(n)$ and that algorithm $B$ runs in worst-case time $g(n)$. For each of the following questions, answer either yes, no, or can’t tell and explain why.

(a) Is $A$ faster than $B$ for all $n$ greater than some $n_0$ if $g(n) = \Omega(f(n) \log n)$?
(b) Is $A$ faster than $B$ for all $n$ greater than some $n_0$ if $g(n) = O(f(n) \log n)$?
(c) Is $A$ faster than $B$ for all $n$ greater than some $n_0$ if $g(n) = \Theta(f(n) \log n)$?
(d) Is $B$ faster than $A$ for all $n$ greater than some $n_0$ if $g(n) = \Omega(f(n) \log n)$?
(e) Is $B$ faster than $A$ for all $n$ greater than some $n_0$ if $g(n) = O(f(n) \log n)$?
(f) Is $B$ faster than $A$ for all $n$ greater than some $n_0$ if $g(n) = \Theta(f(n) \log n)$?

Source: Skiena, problem 1-2
7. For each of the following pairs of functions, either $f(n)$ is in $O(g(n))$, $f(n)$ is in $\Omega(g(n))$, or $f(n) = \Theta(g(n))$. Determine which relationship is correct and briefly explain why.

(a) $f(n) = \log n^2$; $g(n) = \log n + 5$
(b) $f(n) = \sqrt{n}$; $g(n) = \log n^2$
(c) $f(n) = \log^2 n$; $g(n) = \log n$
(d) $f(n) = n$; $g(n) = \log^2 n$
(e) $f(n) = n \log n + n$; $g(n) = \log n$
(f) $f(n) = 10$; $g(n) = \log 10$
(g) $f(n) = 2^n$; $g(n) = 10n^2$
(h) $f(n) = 2^n$; $g(n) = 3^n$

Source: Shaffer, chapter 3, problem 3.8 (*)

8. List the functions below from the lowest to the highest order. If any two or more are of the same order, indicate which.

\[
\begin{align*}
n &< n - n^3 + 7n^5 & < & & 2^n & < & & n \log n & < & & \ln n \\
 & n^2 + \log n & < & & \sqrt{n} & < & & e^n & < & & \log \log n \\
n^3 & < & & (\log n)^2 & < & & n! & < & & n^{1+\varepsilon} \text{ where } 0 < \varepsilon < 1
\end{align*}
\]

Source: Baase, problem 1.16 (*)

9. For each of the following pairs of functions $f(n)$ and $g(n)$, determine whether $f(n) = O(g(n))$, $g(n) = O(f(n))$, or both.

(a) $f(n) = (n^2 - n)/2$, $g(n) = 6n$
(b) $f(n) = n + 2\sqrt{n}$, $g(n) = n^2$
(c) $f(n) = n \log n$, $g(n) = n \sqrt{n}/2$
(d) $f(n) = n + \log n$, $g(n) = \sqrt{n}$
(e) $f(n) = 2(\log n)^2$, $g(n) = \log n + 1$
(f) $f(n) = 4n \log n + n$, $g(n) = (n^2 - n)/2$

Source: Ian Parberry, section 3.1 problem 100-107

10. Prove that $n^3 - 3n^2 - n + 1 = \Theta(n^3)$.

Source: Ian Parberry, section 3.3 problem 139

11. Prove that

(a) $2^n = O(n!)$
(b) \( n! = \Omega(2^n) \)

Source: Ian Parberry, section 3.3 problem 147 and 148.

12. For each of the following pairs of functions \( f(n) \) and \( g(n) \), give an appropriate positive constant \( c \) such that \( f(n) \leq c \cdot g(n) \) for all \( n > 1 \).

(a) \( f(n) = n^2 + n + 1, \ g(n) = 2n^3 \)

(b) \( f(n) = n\sqrt{n} + n^2, \ g(n) = n^2 \)

(c) \( f(n) = n^2 - n + 1, \ g(n) = n^2/2 \)

Source: Ian Parberry, section 3.3 problem 165,166,168

13. Prove that if \( f_1(n) = O(g_1(n)) \) and \( f_2(n) = O(g_2(n)) \), then \( f_1(n) + f_2(n) = O(g_1(n) + g_2(n)) \).

Source: Ian Parberry, section 3.4 problem 171

14. Prove that if \( f_1(N) = \Omega(g_1(n)) \) and \( f_2(n) = \Omega(g_2(n)) \), then \( f_1(n) + f_2(n) = \Omega(g_1(n) + g_2(n)) \).

Source: Ian Parberry, section 3.4 problem 172

15. Prove that if \( f_1(n) = O(g_1(n)) \) and \( f_2(n) = O(g_2(n)) \), then \( f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n)) \)

Source: Ian Parberry, section 3.4 problem 177

16. Show that big-O is transitive. That is, if \( f(n) = O(g(n)) \) and \( g(n) = O(h(n)) \), then \( f(n) = O(h(n)) \).

Source: Ian Parberry, section 3.4 problem 183

17. Is \( f(cn) = \Theta(f(n)) \) for all functions \( f \) and positive constants \( c \)? (Hint: Consider particularly fast-growing functions.)

Source: Baase, problem 1.17 (*)

18. Give a proof or counterexample to the following claim: for all functions \( f(n) \) and \( g(n) \), either \( f(n) = O(g(n)) \) or \( g(n) = O(f(n)) \).

Source: Skiena, problem 1-7, Baase, problem 1.22 and Ian Parberry, section 3.4 problem 178 (*)

19. Does \( f(n) = O(g(n)) \) imply that \( 2^{f(n)} = O(2^{g(n)}) \)? Explain your reasoning!

Source: Skiena, problem 1-6 (*)
1.3 Recurrence Relations

1.4 Induction

1.5 Proof by Contradiction

1.6 Number Theory

1.7 Binomial Coefficients and Counting

1.8 Logarithms

1. (a) Prove that \( \log_a(xy) = \log_a x + \log_a y \)
(b) Prove that \( \log_a x^y = y \log_a x \)
(c) Prove that \( \log_a x = \frac{\log_b x}{\log_b a} \)
(d) Prove that \( x^{\log_b y} = y^{\log_b x} \)

Source: Brassard and Bratley, problem 1.15 (*)

2. Prove that that the binary representation of \( n \geq 1 \) has \( \lfloor \log_2 n \rfloor + 1 \) bits.
Source: Ian Parberry, section 2.13 problem 92

3. In one of my research papers I give a comparison-based sorting algorithm that runs in \( O(n \log(\sqrt{n})) \). Given the existence of an \( \Omega(n \log n) \) lower bound for sorting, how can this be possible?
Source: Skiena, problem 2-8 (*)

1.9 Estimation

1. Do all the books you own total at least one million pages? How many total pages are stored in your school library?
Source: Schaffer, chapter 2 problem 2.21

2. How many words are there in the textbook?
Source: Schaffer, chapter 2 problem 2.22

3. How many hours are one million seconds? How many days? Answer these questions by doing all arithmetic in your head.
Source: Schaffer, chapter 2 problem 2.23

4. Estimate how many cities and towns there are in the United States.
Source: Schaffer, chapter 2 problem 2.24
5. Estimate how many cubic miles of water flow out of the mouth of the Mississippi River each day? Do not look up any supplemental facts. Describe all assumptions you made in arriving at your answer.

Source: Shaffer, chapter 2 problem 2.18

6. A sorting algorithm takes 1 second to sort 1000 items on your local machine. How long will it take to sort 10,000 items...

(a) if you believe that the algorithm takes time proportional to $n^2$, and
(b) if you believe that the algorithm takes time roughly proportional to $n \log n$?

Source: Brassard and Bratley, problem 2.6

1.10 Probability

1. Prove that two tracks selected at random from a disk with $n$ tracks are on average separated by $n/3$ tracks.

Source: Shaffer, problem 9.3

1.11 Program Analysis

1. For each of these questions, briefly explain your answer.

(a) If I prove that an algorithm takes $O(n^2)$ worst-case time, is it possible that it takes $O(n)$ on some inputs?

(b) If I prove that an algorithm takes $O(n^2)$ worst-case time, is it possible that it takes $O(n)$ on all inputs?

(c) If I prove that an algorithm takes $\Theta(n^2)$ worst-case time, is it possible that it takes $O(n)$ on some inputs?

(d) If I prove that an algorithm takes $\Theta(n^2)$ worst-case time, is it possible that it takes $O(n)$ on all inputs?

(e) Is the function $f(n) = \Theta(n^2)$, where $f(n) = 100n^2$ for even $n$ and $f(n) = 20n^2 - n \log_2 n$ for odd $n$?

Source: Skiena, problem 1-3

2. How can we modify almost any algorithm to have a good best-case running time?

Source: CLR, problem 1.2-6 (*)

3. What value is returned by the following function? Express your answer as a function of $n$. Give the worst-case running time using the big-Oh notation.

\[
\text{function} \text{ mystery}(n)
\]

1. $r := 0$

2. for $i := 1$ to $n - 1$ do
3. \[ for \ j := i + 1 \ to \ n \ do \]
4. \[ for \ k := 1 \ to \ j \ do \]
5. \[ r := r + 1 \]
6. \[ \text{return}(r) \]

**Source:** Ian Parberry, section 6.1, problem 280.

4. What value is returned by the following function? Express your answer as a function of \(n\). Give the worst-case running time using big-Oh notation.

\[
\text{function} \ \text{pesky}(n)
\]
1. \[ r := 0 \]
2. \[ for \ i := 1 \ to \ n \ do \]
3. \[ for \ j := 1 \ to \ i \ do \]
4. \[ for \ k := j \ to \ i + j \ do \]
5. \[ r := r + 1 \]
6. \[ \text{return}(r) \]

**Source:** Ian Parberry, section 6.1 problem 281

5. What value is returned by the following function? Express your answer as a function of \(n\). Give the worst-case running time using big-Oh notation.

\[
\text{function} \ \text{prestiferous}(n)
\]
1. \[ r := 0 \]
2. \[ for \ i := 1 \ to \ n \ do \]
3. \[ for \ j := 1 \ to \ i \ do \]
4. \[ for \ k := j \ to \ i + j \ do \]
5. \[ for \ l := 1 \ to \ i + j - k \ do \]
6. \[ r := r + 1 \]
7. \[ \text{return}(r) \]

**Source:** Ian Parberry, section 6.1 problem 282 (*)

6. What value is returned by the following function? Express your answer as a function of \(n\). Give the worst-case running time using big-Oh notation.

\[
\text{function} \ \text{comundrum}(n)
\]
1. \[ r := 0 \]
2. \[ for \ i := 1 \ to \ n \ do \]
3. \[ for \ j := i + 1 \ to \ n \ do \]
4. \[ for \ k := i + j - 1 \ to \ n \ do \]

7
5. \[ r := r + 1 \]
6. \[ \text{return}(r) \]

*Source:* Ian Parberry, section 6.1 problem 283 \((**))

### 1.12 Proofs of Correctness

1. (a) Prove the correctness of the following recursive algorithm for incrementing natural numbers:

   \[
   \text{function \text{increment}}(y) \\
   \text{comment Return } y + 1 \\
   1. \quad \text{if } y = 0 \text{ then return}(1) \text{ else} \\
   2. \quad \text{if } (y \mod 2) = 1 \text{ then} \\
   3. \quad \text{return}(2 \cdot \text{increment}([y/2])) \\
   4. \quad \text{else return}(y + 1)
   \]

   (b) Analyze this algorithm. How many right-shifts does it use, i.e., how many times is line 2 executed in the worst case?

   *Source:* Ian Parberry, section 5.2 problem 267 and section 6.2 problem 301

2. (a) Prove the correctness of the following recursive algorithm to multiply two natural numbers is correct, for all integer constants \( c \geq 2 \).

   \[
   \text{function \text{multiply}}(y, z) \\
   \text{comment Return the product } yz. \\
   1. \quad \text{if } z = 0 \text{ then return}(0) \text{ else} \\
   2. \quad \text{return}(\text{multiply}(cy, \lfloor z/c \rfloor) + y \cdot (z \mod c))
   \]

   (b) Analyze this algorithm. How many additions does it use, i.e., how many times is line 2 executed in the worst case?

   *Source:* Ian Parberry, section 5.2 problem 267 and section 6.2 problem 305

### 1.13 Programming Exercises

### 2 Data Structures

#### 2.1 Simple Data Structures

1. For each of the four types of linked lists in the following table, what is the asymptotic worst-case running time for each dynamic-set operation listed?
| Search($L, k$) | singly sorted | doubly sorted | singly unsorted | doubly unsorted |
| Insert($L, x$) | | | | |
| Delete($L, x$) | | | | |
| Successor($L, x$) | | | | |
| Predecessor($L, x$) | | | | |
| Minimum($L$) | | | | |
| Maximum($L$) | | | | |

Source: CLR, problem 11-1 (*)

2. A common problem for compilers and text editors is determining whether the parentheses (or other brackets) in a string are balanced and properly nested. For example, the string $((())())$ contains properly nested pairs of parentheses, which the strings $)(()$ and $(())$ do not.

   (a) Give an algorithm that returns true if a string contains properly nested and balanced parentheses, and false otherwise. **Hint:** At some point while scanning an illegal string from left to right will you have encountered more right parentheses than left parentheses.

   (b) Give an algorithm that returns the position of the first offending parenthesis if the string is not properly nested and balanced. That is, if an excess right parenthesis is found, return its position; if there are too many left parentheses, return the position of the first left parenthesis. Return -1 if the string is properly balanced and nested.

Source: Shaffer, chapter 1 problem 1.13

3. Write a program to reverse the direction of a given singly-linked list. In other words, after the reversal all pointers should now point backwards.

Source: Manber problem 4.2

4. Suppose that we are given a stack of $n$ elements which we would like to sort, by returning a stack containing the records in sorted order (with the smallest value on top). We are allowed to use only the following operations to manipulate the data:

   - **Pop-Push**($s_1, s_2$) – pop the top item from stack $s_1$ and push it onto stack $s_2$.
   - **Compare**($s_1, s_2$) – test whether the top element of stack $s_1$ is less than the top element of stack $s_2$.

   (a) Give a $\Theta(n^2)$ sorting algorithm using just these operations and three stacks.

   (b) Give an initial stack which cannot be sorted just using these operations and two stacks.
(c) Give a $\Theta(n^2)$ sorting algorithm using just these operations and two stacks, where you are allowed to temporarily park elements in a constant number of additional registers. (hint: use insertion sort)

Source: Shaffer, chapter 8 problem 8.2 (*)

5. Programs sometimes crash for mysterious reasons. A compiler crashed trying to translate a short program containing the following (correct and valid) case statement, with a message complaining it had run out of memory space:

```plaintext
case i of
1: Statement(1);
2: Statement(2);
4: Statement(3);
256: Statement(4);
65535: Statement(5);
```

Speculate as to what data structure the compiler used that caused the problem.

Source: Manber, problem 4.30 (**) 

6. Design a data structure that allows one to search, insert, and delete an integer $X$ in $O(1)$ time (ie constant time, independent of the total number of integers stored). Assume that $1 \leq X \leq n$ and that there are $m + n$ units of space available for the symbol table, where $m$ is the maximum number of integers that can be in the table at any one time. (Hint: use two arrays $A[1..n]$ and $B[1..m]$.) You are not allowed to initialize either $A$ or $B$, as that would take $O(m)$ or $O(n)$ operations. This means the arrays are full of random garbage to begin with, so you must be very careful.

Source: Skiena problem 2-15 (**) 

### 2.2 Heaps 

1. Devise an algorithm for finding the $k$ smallest elements of an unsorted set of $n$ integers in $O(n + k \log n)$.

Source: Ian Parberry, section 13.1 problem 608

2. Given an array-based heap on $n$ elements and a real number $x$, efficiently determine whether the $k$th smallest in the heap is greater than or equal to $x$. Your algorithm should be $O(k)$ in the worst-case, independent of the size of the heap. Hint: you not have to find the $k$th smallest element; you need only determine its relationship to $x$.

Source: Manber, problem 6.34 (*)

3. Give an $O(n \log k)$-time algorithm which merges $k$ sorted lists with a total of $n$ elements into one sorted list. (hint: use a heap to speed up the elementary $O(kn)$-time algorithm).

Source: CLR, problem 7.5-6 (*)
4. (a) Give an efficient algorithm to find the second-largest key among \( n \) keys. You can do better than \( 2n - 3 \) comparisons.

(b) Give an efficient algorithm to find the third-largest key among \( n \) keys. How many key comparisons does your algorithm do in the worst case? Must your algorithm determine which key is largest and second-largest in the process?

Source: Baase, problem 1.11 and 3.7 (*)

2.3 Union-Find

1. There is well-known data structure for union and find operations that supports these operations in worst-case time \( O(\log n) \) and amortized time \( O(\log^* n) \).

Design a data structure that can perform a sequence \( m \) union and find operations on a universal set of \( n \) elements, consisting of a sequence of all unions followed by a sequence of all finds, in time \( O(m + n) \).

Source: Ian Parberry, section 11.4 problem 562 (*)

2.4 Trees and Other Dictionary Structures

1. Design a dictionary data structure in which insertions, deletions, and membership queries can be all processed in \( O(1) \) time in the worst case. You may assume the set elements are integers drawn from a finite set \( 1, 2, \ldots, n \), and initialization can take \( O(n) \) time.

Source: Ian Parberry, section 11.5 problem 565

2. Describe how to modify any balanced tree data structure such that search, insert, delete, minimum, and maximum still take \( O(\log n) \) time each, but successor and predecessor now take \( O(1) \) time each. Which operations have to be modified to support this?

Source: Problem Skiena 2-7 (*)

3. Suppose you have access to a balanced dictionary data structure, which supports each of the operations search, insert, delete, minimum, maximum, successor, and predecessor in \( O(\log n) \) time. Explain how to modify the insert and delete operations so they still take \( O(\log n) \) but now minimum and maximum take \( O(1) \) time. (Hint: think in terms of using the abstract dictionary operations, instead of mucking about with pointers and the like.)

Source: Skiena, problem 2-9 (*)

4. You are given the task of reading in \( n \) numbers and then printing them out in sorted order. Suppose you have access to a balanced dictionary data structure, which supports each of the operations search, insert, delete, minimum, maximum, successor, and predecessor in \( O(\log n) \) time.
• Explain how you can use this dictionary to sort in $O(n \log n)$ time using only the following abstract operations: minimum, successor, insert, search.

• Explain how you can use this dictionary to sort in $O(n \log n)$ time using only the following abstract operations: minimum, insert, delete, search.

• Explain how you can use this dictionary to sort in $O(n \log n)$ time using only the following abstract operations: insert and in-order traversal.

Source: Skiena, problem 2-12 (*)

5. Given the pre-order and in-order traversals of a binary tree, is it possible to reconstruct the tree? If so, sketch an algorithm to do it. If not, give a counterexample. Repeat the problem if you are given the pre-order and post-order traversals.

Source: Skiena, problem 4-6

6. A concatenate operation takes two sets $S_1$ and $S_2$, where every key in $S_1$ is smaller than any key in $S_2$, and merges them together. Give an algorithm to concatenate two binary search trees into one binary search tree. The worst-case running time should be $O(h)$, where $h$ is the maximal height of the two trees.

Source: Manber, problem 4.19

7. Let $T_1$ and $T_2$ be two binary search trees, each having $n$ nodes and the exact same set of keys. Prove that by applying $2n$ rotations to $T_1$, its shape can be converted to $T_2$. (hint: convert to a right going chain).

Source: CLR, problem 14.2-5 and Manber, problem 4.24 (*)

8. Design a data structure to support the following operations:

• $insert(x, T)$ – Insert item $x$ into the set $T$.

• $delete(k, T)$ – Delete the $k$th smallest element from $T$.

• $member(x, T)$ – Return true iff $x \in T$.

All operations must take $O(\log n)$ time on an $n$-element set.

Source: Ian Parberry, section 11.5 problem 567 (*)

9. In the bin-packing problem, we are given $n$ metal objects, each weighing between zero and one kilogram. We also have a collection of large, but fragile bins. Our goal is to find the smallest number of bins that will hold the $n$ objects, with no bin holding more than one kilogram.

• The any-fit heuristic for bin packing is as follows. Take each of the objects in the order in which they are given. For each object, place it into any partially filled bin which has room. If no such bin exists, start a new bin. Design an algorithm that implements the first-fit heuristic (taking as input the $n$ weights $w_1, w_2, \ldots, w_n$ and outputting the number of bins needed when using first-fit) in $O(n \log n)$ time.
• Repeat the above using the best-fit heuristic, where we put the next object in the partially filled bin with the smallest amount of extra room after the object is inserted.

• Repeat the above using the worst-fit heuristic, where we put the next object in the partially filled bin with the largest amount of extra room after the object is inserted.

Source: Ian Parberry, section 11.5 problem 570 (*)

10. Let $A[1..n]$ be an array of real numbers. Design an algorithm to perform any sequence of the following operations:

• $Add(i, y)$ – add the value $y$ to the $i$th number.
• $Partial\-sum(i)$ – return the sum of the first $i$ numbers, i.e. $\sum_{i=1}^{n} A[i]$.

There are no insertions or deletions; the only change is to the values of the numbers. Each operation should take $O(\log n)$ steps. You may use one additional array of size $n$ as a work space.

Source: Manber, problem 4.27 (*)

11. Extend the data structure of the previous problem to support insertions and deletions. Each element now has both a key and a value. An element is accessed by its key. The addition operation applied to the values, but the element are accessed by its key. The $Partial\-sum$ operation is different.

• $Add(k, y)$ – add the value $y$ to the item with key $k$.
• $Insert(k, y)$ – insert a new item with key $k$ and value $y$.
• $Delete(k)$ – delete the item with key $k$.
• $Partial\-sum(k)$ – return the sum of all the elements currently in the set whose key is less than $y$, i.e. $\sum_{x_i < y} x_i$.

The worst case running time should still be $O(n \log n)$ for any sequence of $O(n)$ operations.

Source: Manber problem 4.28 (**)

12. Certain highways can be modeled as an ordered list of toll booths $t_1, t_2, \ldots, t_n$ where one pays a toll of $a_i$ whenever passing toll booth $t_i$, for all $1 \leq i \leq n$. Design a data structure that performs all of the following operations in $O(\log n)$ time.

• $insert(k, a)$ – Insert a new toll booth after $t_k$ with toll $a$.
• $delete(k)$ – Delete toll booth $t_k$.
• $toll(i, j)$ – Return the total toll for booths $i$ through $j$, inclusive.
• $update(i, j, a)$ – Add $a$ to the toll for booths $i$ through $j$, inclusive.
13. Suppose that we are given a sequence of \( n \) values \( x_1, x_2, \ldots, x_n \) and seek to quickly answer repeated queries of the form: given \( i \) and \( j \), find the smallest value in \( x_i, \ldots, x_j \).

   (a) Design a data structure that uses \( O(n^2) \) space and answers queries in \( O(1) \) time.
   
   (b) Design a data structure that uses \( O(n) \) space and answers queries in \( O(\log n) \) time. For partial credit, your data structure can use \( O(n \log n) \) space and have \( O(\log n) \) query time.

Source: Ian Parberry, section 11.5 problem 573 (**)

2.5 Programming Exercises

3 Sorting and Searching

3.1 Applications of Sorting

1. Outline a reasonable method of solving each of the following problems. Give the order of the worst-case complexity of your methods.

   (a) You are given a pile of thousands of telephone bills and thousands of checks sent in to pay the bills. Find out who did not pay.

   (b) You are given a list containing the title, author, call number and publisher of all the books in a school library and another list of 30 publishers. Find out how many of the books in the library were published by each of those 30 companies.

   (c) You are given all the book checkout cards used in the campus library during the past year, each of which contains the name of the person who took out the book. Determine how many distinct people checked out at least one book.

Source: Baase, problem 2.30

2. Newt Gingrich is given the job of partitioning \( 2n \) players into two teams of \( n \) players each. Each player has a numerical rating that measures how good he/she is at the game. Newt seeks to divide the players as \textit{unfairly} as possible, so as to create the biggest possible talent imbalance between team \( A \) and team \( B \). Show how Newt can do the job in \( O(n \log n) \) time.

Source: Skiena, problem 2-1

3. Given two sets \( S_1 \) and \( S_2 \) (each of size \( n \)), and a number \( x \), describe an \( O(n \log n) \) algorithm for finding whether there exists a pair of elements, one from \( S_1 \) and one from \( S_2 \), that add up to \( x \). (For partial credit, give a \( \Theta(n^2) \) algorithm for this problem.)

Source: Skiena, problem 2-5
4. For each of the following problems, give an algorithm that finds the desired numbers within the given amount of time. To keep your answers brief, feel free to use algorithms from the book as subroutines. For the example, \( S = \{6, 13, 19, 3, 8\} \), 19 – 3 maximizes the difference, while 8 – 6 minimizes the difference.

(a) Let \( S \) be an unsorted array of \( n \) integers. Give an algorithm that finds the pair \( x, y \in S \) that maximizes \( |x - y| \). Your algorithm must run in \( O(n) \) worst-case time.

(b) Let \( S \) be a sorted array of \( n \) integers. Give an algorithm that finds the pair \( x, y \in S \) that maximizes \( |x - y| \). Your algorithm must run in \( O(1) \) worst-case time.

(c) Let \( S \) be an unsorted array of \( n \) integers. Give an algorithm that finds the pair \( x, y \in S \) that minimizes \( |x - y| \), for \( x \neq y \). Your algorithm must run in \( O(n \log n) \) worst-case time.

(d) Let \( S \) be a sorted array of \( n \) integers. Give an algorithm that finds the pair \( x, y \in S \) that minimizes \( |x - y| \), for \( x \neq y \). Your algorithm must run in \( O(n) \) worst-case time.

Source: Skiena, problem 2-6

5. Give an efficient algorithm to determine whether two sets (of size \( m \) and \( n \)) are disjoint. Analyze the complexity of your algorithm in terms of \( m \) and \( n \). Be sure to consider the case where \( m \) is substantially smaller than \( n \).

Source: Manber, problem 6.24

6. The nuts and bolts problem is defined as follows. You are given a collection of \( n \) bolts of different widths, and \( n \) corresponding nuts. You can test whether a given nut and bolt together, from which you learn whether the nut is too large, too small, or an exact match for the bolt. The differences in size between pairs of nuts or bolts can be too small to see by eye, so you cannot rely on comparing the sizes of two nuts or two bolts directly. You are to match each bolt to each nut.

(a) Give an \( O(n^2) \) algorithm to solve the nuts and bolts problem.

(b) Suppose that instead of matching all of the nuts and bolts, you wish to find the smallest bolt and its corresponding nut. Show that this can be done in only \( 2n - 2 \) comparisons.

(c) Show that any algorithm for the nuts and bolts problem must take \( \Omega(n \log n) \) comparisons in the worst case. (Hint: sorting \( n \) items takes \( \Omega(n \log n) \) comparisons in the worst case.)

Source: Ian Parberry, section 13.1 problems 610 and 615. (*)

### 3.2 Quicksort

1. The running time for Quicksort depends upon both the data being sorted and the partition rule used to select the pivot. Although Quicksort is \( O(n \log n) \) on average, certain partition rules cause Quicksort to take \( \Theta(n^2) \) time if the array is already sorted.
(a) Suppose we always pick the pivot element to be the key from the last position of the subarray. On a sorted array, does Quicksort now take $\Theta(n)$, $\Theta(n \log n)$, or $\Theta(n^2)$?

(b) Suppose we always pick the pivot element to be the key from the middle position of the subarray. On a sorted array, does Quicksort now take $\Theta(n)$, $\Theta(n \log n)$, or $\Theta(n^2)$?

(c) Suppose we always pick the pivot element to be the key of the median element of the first three keys of the subarray. (The median of three keys is the middle value, so the median of 5, 3, 8 is five.) On a sorted array, does Quicksort now take $\Theta(n)$, $\Theta(n \log n)$, or $\Theta(n^2)$?

(d) Suppose we always pick the pivot element to be the key of the median element of the first, last, and middle elements of the subarray. On a sorted array, does Quicksort now take $\Theta(n)$, $\Theta(n \log n)$, or $\Theta(n^2)$?

Source: Skiena, problem 2-13

2. Suppose an array $A$ consists of $n$ elements, each of which is red, white, or blue. We seek to sort the elements so that all the reds come before all the whites, which come before all the blues. The only operation permitted on the keys are

- $\text{Examine}(A, i)$ – report the color of the $i$th element of $A$.
- $\text{Swap}(A, i, j)$ – swap the $i$th element of $A$ with the $j$th element.

Find a correct and efficient algorithm for red-white-blue sorting. There is a linear-time solution.

Source: Baase, problem 2.38 (*)

### 3.3 Mergesort

### 3.4 Other Sorting Algorithm

1. An inversion of a permutation is a pair of elements which are out of order.

   (a) Show that a permutation of $n$ items has at most $n(n - 1)/2$ inversions. Which permutation(s) have exactly $n(n - 1)/2$ inversions?

   (b) Let $P$ be a permutation and $P^r$ be the reversal of this permutation. Show that $P$ and $P^r$ have a total of exactly $n(n - 1)/2$ inversions.

   (c) Use the previous result to argue that the expected number of inversions in a random permutation is $n(n - 1)/4$.

Source: Baase, problem 2.9

2. Show that $n$ positive integers in the range $1$ to $k$ can be sorted in $O(n \log k)$ time.

Source: Ian Parberry, section 13.1 problem 607

3. We seek to sort a sequence $S$ of $n$ integers with many duplications, such that the number of distinct integers in $S$ is $O(\log n)$. 

16
(a) Give an $O(n \log \log n)$ worst-case time algorithm to sort such subsequences.
(b) Why doesn’t this violate the $\Omega(n \log n)$ lower bound for sorting?

Source: Manber, problem 6.32 (*)

### 3.5 Searching

1. Consider the numerical 20 questions game. In this game, player 1 thinks of a number in the range 1 to $n$. Player 2 has to figure out this number by asking the fewest number of true/false questions. Assume that nobody cheats.

   (a) What is an optimal strategy if $n$ is known?
   (b) What is a good strategy if $n$ is not known?

Source: Manber, problem 6.1 and 6.2

2. Let $M$ be an $n \times m$ integer matrix in which the entries of each row are sorted in increasing order (from left to right) and the entries in each column are in increasing order (from top to bottom). Give an efficient algorithm to find the position of an integer $x$ in $M$, or to determine that $x$ is not there. How many comparisons of $x$ with matrix entries does your algorithm use in worst case?

Source: Baase, problem 2.41 (*)

3. Prove that a successful sequential search for a random element in an $n$ element list performs, on average, $(n+1)/2$ comparisons.

Source: Brassard and Bratley, problem 7.9

4. The mode of a set of numbers is the number that occurs most frequently in the set. The set $(4, 6, 2, 4, 3, 1)$ has a mode of 4.

   (a) Give an efficient and correct algorithm to compute the mode of a set of n numbers.
   (b) Suppose we know that there is an (unknown) element that occurs $n/2 + 1$ times in the set. Give a worst-case linear-time algorithm to find the mode. For partial credit, your algorithm may run in expected linear time.

Source: Skiena problem 2-4 (*)

### 3.6 Programming Exercises

### 4 Divide and Conquer

#### 4.1 Binary Search

1. Suppose you are given an array $A$ of $n$ sorted numbers that has been circularly shifted $k$ positions to the right. For example, $\{35, 42, 5, 15, 27, 29\}$ is a sorted array that has
been circularly shifted \( k = 2 \) positions, while \( \{27, 29, 35, 42, 5, 15\} \) has been shifted \( k = 4 \) positions.

- Suppose you know what \( k \) is. Give an \( O(1) \) algorithm to find the largest number in \( A \).
- Suppose you do not know what \( k \) is. Give an \( O(\lg n) \) algorithm to find the largest number in \( A \). For partial credit, you may give an \( O(n) \) algorithm.

Source: Skiena, problem 3-11

2. Suppose that you are given a sorted sequence of distinct integers \( \{a_1, a_2, \ldots, a_n\} \). Give an \( O(\lg n) \) algorithm to determine whether there exists an index \( i \) such that \( a_i = i \). For example, in \( \{-10, -3, 3, 5, 7\} \), \( a_3 = 3 \). In \( \{2, 3, 4, 5, 6, 7\} \), there is no such \( i \).

Source: Skiena, problem 3-12 (*)

3. Given an array storing numbers ordered by value, modify the binary search routine to return the position of the first number with value \( x \) in the situation where \( x \) can appear multiple times in the array. Be sure that your algorithm is \( \Theta(\log n) \), that is, do not resort to sequential search once an occurrence of \( x \) is found.

Source: Shaffer, chapter 3 problem 3.13 (*)

4. Suppose that \( S \) and \( T \) are sorted arrays, each containing \( n \) distinct elements.

   (a) Give an \( O((\lg n)^2) \) algorithm to find the \( n \)th smallest of the \( 2n \) element.

   (b) Give a lower bound or faster algorithm for the problem.

Source: Baase, problem 3.15 (*)

4.2 Recursive Algorithms

1. You have fifty coins that are all supposed to be gold coins of the same weight, but you know that one coin is fake and weighs less than the others. You have a balance scale which can take two subsets of coins and tell you if the two sides weigh the same, and which side is lighter if not. Outline an algorithm for finding the fake coin. How many weighings will you do?

Source: Baase, problem 1.33

2. A max-min algorithm finds both the largest and smallest elements in an array of \( n \) values. Design and analyze a divide-conquer max-min algorithm that uses \( \lceil 3n/2 \rceil - 2 \) comparisons for any integer \( n \). (Hint: first consider the case where \( n \) is a power of 2.)

Source: Ian Parberry, section 7.1 problem 317 and Baase, problem 1.12 (*)

3. Give an efficient divide-and-conquer algorithm to find the \( k \)th largest element in the merge of two sorted sequences \( S_1 \) and \( S_2 \). The best algorithm runs in time \( O(\log(\max(m,n))) \), where \( |S_1| = n \) and \( |S_2| = m \).

Source: Ian Parberry, section 7.8 problem 377 (*)
4. Given an array of $n$ real numbers, consider the problem of finding the maximum sum in any contiguous subvector of the input. For example, in the array

$$\{31, -41, 59, 26, -53, 58, 97, -93, -23, 84\}$$

the maximum is achieved by summing the third through seventh elements, where $59 + 26 + (-53) + 58 + 97 = 187$. When all numbers are positive, the entire array is the answer, while when all numbers are negative, the empty array maximizes the total at 0.

- Give a simple, clear, and correct $\Theta(n^2)$-time algorithm to find the maximum contiguous subvector.
- Now give a $\Theta(n)$-time dynamic programming algorithm for this problem. To get partial credit, you may instead give a correct $O(n \log n)$ divide-and-conquer algorithm.

*Source:* Skiena, problem 3-5, Baase, problem 2.42 and Ian Parberry, section 7.8 problem 371 (*)

5. Let $X = \{x_1, x_3, ..., x_n\}$ be a sequence of arbitrary real numbers. Give a linear time algorithm to find the subsequence of consecutive elements $x_i, x_{i+1}, ..., x_j$ whose product is maximum over all consecutive subsequences. The product of the empty subsequence is defined as 1. Observe that $X$ can contain reals less than 1, including negative numbers.

*Source:* Manber, problem 5.12 (*)

### 4.3 Programming Exercises

### 5 Dynamic Programming

#### 5.1 Greedy Algorithms

1. The natural greedy algorithm for making change of $n$ units using the smallest number of coins is as follows. Give the customer one unit of the highest denomination coin of at most $n$ units, say $d$ units. Now repeat to make change of the remaining $n - d$ units. For each of the following nations coinage, establish whether or not this greedy algorithm always minimizes the number of coins returned in change. If so, prove it, if not give a counter example.

(a) United States coinage, which consists of half dollars (50 cents), quarters (25 cents), dimes (10 cents), nickels (5 cents), and pennies.

(b) English coinage before the decimalization, which consisted of half-crowns (30 pence), florins (24 pence), shillings (12 pence), sixpence (6 pence), threepence (3 pence), pennies (1 pence), half pennies (1/2 pence), and farthings (1/4 pence).
(c) Portuguese coinage, which includes coins for \(1, 2, 3, 5, 10, 20, 25\) and \(50\) escudos. You need only consider change for an integer number of escudos.

(d) Martian coinage, where the available denominations are \(1, p, p^2, \ldots, p^n\), where \(p > 1\) and \(n \geq 0\) are integers.

\textit{Source:} Brassard and Bratley, problem 6.2, 6.3 and 6.4. (*)

2. In the one-processor scheduling problem, we are given a set \(J\) of \(n\) jobs. Each job \(i\) has a processing time \(t_i\) and a deadline \(d_i\). A feasible schedule is a permutation of the jobs such that when the jobs are performed in that order, then every job is finished before its deadline. The greedy algorithm for the one-processor scheduling process the jobs earliest deadline first.

Show that if a feasible schedule exists, then the schedules produced by this greedy algorithm is feasible.

\textit{Source:} Ian Parberry, section 9.5 problem 473. (*)

3. Consider the problem of storing \(n\) books on shelves in a library. The order of the books is fixed by the cataloging system and so cannot be rearranged. Therefore, we can speak of a book \(b_i\), where \(1 \leq i \leq n\), that has a thickness \(t_i\) and height \(h_i\). The length of each bookshelf at this library is \(L\).

Suppose all the books have the same height \(h\) (i.e., \(h = h_i = h_j\) for all \(i, j\)) and the shelves are all separated by a distance of greater than \(h\), so any book fits on any shelf. The greedy algorithm would fill the first shelf with as many books as we can until we get the smallest \(i\) such that \(b_i\) does not fit, and then repeat with subsequent shelves.

Show that the greedy algorithm always finds the optimal shelf placement, and analyze its time complexity.

\textit{Source:} Skiena, problem 3-1

4. This is a generalization of the previous problem. Now consider the case where the height of the books is not constant, but we have the freedom to adjust the height of each shelf to that of the tallest book on the shelf. Thus the cost of a particular layout is the sum of the heights of the largest book on each shelf.

- Give an example to show that the greedy algorithm of stuffing each shelf as full as possible does not always give the minimum overall height.
- Give an algorithm for this problem, and analyze its time complexity. Hint: use dynamic programming.

\textit{Source:} Skiena, problem 3-2. (*)
5.2 Fibonacci Numbers

5.3 Approximate and Exact String Matching

1. The longest common substring (not subsequence) of two strings $X$ and $Y$ is the longest string which appears as a run of consecutive letters in both strings. For example, the longest common substring of photograph and tomography is ograph.

   (a) Let $n = |X|$ and $m = |Y|$. Give a $\Theta(nm)$ dynamic programming algorithm for longest common substring based on the longest common subsequence / edit distance algorithm.

   (b) Give a simpler $\Theta(nm)$ which does not rely on dynamic programming.

   (c) (hard) Give an $O(n + m)$ algorithm for longest common substring using suffix trees.

   Source: Baase, problem 6.16 (*)

2. Let $A = a_1a_2 \cdots a_n$ and $B = b_1b_2 \cdots b_n$ be two character strings of the same length. Give an $O(n)$ algorithm to determine whether $B$ is a cyclic shift of $A$. $B$ is a cyclic shift of $A$ if there exists an index $k$, $1 \leq k \leq n$ such that $a_i = b_{(k+i)\mod n}$ for all $i$, $1 \leq i \leq n$. (Hint: recall that testing whether string $S$ is a substring of $T$ can be done in $O(|S| + |T|)$ time.)

   Source: Manber, problem 6.44 and Baase, problem 5.20 (*)

3. Give an $O(n^2)$ algorithm to find the longest monotonically increasing sequence in a sequence of $n$ numbers.

   Source: CLR, problem 16.3-5 (*)

4. Suppose you are given three strings of characters: $X$, $Y$, and $Z$, where $|X| = n$, $|Y| = m$, and $|Z| = n + m$. $Z$ is said to be a shuffle of $X$ and $Y$ iff $Z$ can be formed by interleaving the characters from $X$ and $Y$ in a way that maintains the left-to-right ordering of the characters from each string.

   (a) Show that echocohilaptes is a shuffle of chocolate and chips, but chocochilatspe is not.

   (b) Give an efficient dynamic-programming algorithm that determines whether $Z$ is a shuffle of $X$ and $Y$. Hint: The values the dynamic programming matrix you construct should be Boolean, not numeric.

   Source: Baase, problem 6.17 (*)
5.4 Design Problems

1. Consider a city whose streets are defined by an $X \times Y$ grid. We are interested in walking from the upper left-hand corner of the grid to the lower right-hand corner.

Unfortunately, the city has bad neighborhoods, which are defined as intersections we do not want to walk in. We are given an $X \times Y$ matrix $BAD$, where $BAD[i,j] = \text{"yes"}$ if and only if the intersection between streets $i$ and $j$ is somewhere we want to avoid.

(a) Give an example of the contents of $BAD$ such that there is no path across the grid avoiding bad neighborhoods.

(b) Give an $O(XY)$ algorithm to find a path across the grid that avoids bad neighborhoods.

(c) Give an $O(XY)$ algorithm to find the shortest path across the grid that avoids bad neighborhoods. You may assume that all blocks are of equal length. For partial credit, give an $O(X^2Y^2)$ algorithm.

Source: Skiena, problem 3-3 (*)

2. Consider the same situation as the previous problem. We have a city whose streets are defined by an $X \times Y$ grid. We are interested in walking from the upper left-hand corner of the grid to the lower right-hand corner. We are given an $X \times Y$ matrix $BAD$, where $BAD[i,j] = \text{"yes"}$ if and only if the intersection between streets $i$ and $j$ is somewhere we want to avoid.

If there were no bad neighborhoods to contend with, the shortest path across the grid would have length $(X - 1) + (Y - 1)$ blocks, and indeed there would be many such paths across the grid. Each path would consist of only rightward and downward moves.

Give an algorithm that takes the array $BAD$ and returns the number of safe paths of length $X + Y - 2$. For full credit, your algorithm must run in $O(XY)$.

Source: Skiena, problem 3-4 (*)

3. The knapsack problem is as follows: given a set of integers $S = \{s_1, s_2, \ldots, s_n\}$, and a given target number $T$, find a subset of $S$ which adds up exactly to $T$. For example, within $S = \{1, 2, 5, 9, 10\}$ there is a subset which adds up to $T = 22$ but not $T = 23$.

Find counterexamples to each of the following algorithms for the knapsack problem. That is, give an $S$ and $T$ such that the subset is selected using the algorithm does not leave the knapsack completely full, even though a such a solution exists.

(a) Put the elements of $S$ in the knapsack in left to right order if they fit, i.e. the first-fit algorithm.

(b) Put the elements of $S$ in the knapsack from smallest to largest, i.e. the best-fit algorithm.

(c) Put the elements of $S$ in the knapsack from largest to smallest, i.e. the worst-fit algorithm.
Now give a correct dynamic programming algorithm which runs in $O(nT)$ time.

**Source:** Ian Parberry, section 8.2 problem 395-397

4. Multiplying an $X \times Y$ matrix by a $Y \times Z$ matrix (using the standard, simple algorithm) takes $X \times Y \times Z$ multiplications. Since matrix multiplication is *associative*, we can parenthesize an ordered chain of matrix multiplications however we want, each of which will yield the same answer. The cost of doing the multiplication, however, can differ substantial.

(a) Compute the costs of the three possible parenthesizations of the matrix product $A \times B \times C \times D$, where $A$ is $20 \times 2$, $B$ is $2 \times 15$, $C$ is $15 \times 40$, and $D$ is $40 \times 4$.

(b) Construct an example with only three or four matrices in which the worst factorization does at least 100 times as many multiplications as the best factorization.

(c) Give a counterexample to the claim that simply multiplying from left to right yields the best parenthesization.

(d) Give a counterexample to the claim that simply multiplying from right to left yields the best parenthesization.

(e) Suppose $d_i$ is the largest dimension of a matrix in the ordering. Start by multiplying the two matrices which share this dimension, i.e. matrix $M_i$ by matrix $M_{i+1}$. Repeat this until the product has been evaluated. Give a counterexample to the claim that this yields the best parenthesization.

(f) Suppose $d_i$ is the smallest dimension of a matrix in the ordering. Start by multiplying the two matrices which share this dimension, i.e. matrix $M_i$ by matrix $M_{i+1}$. Repeat this until the product has been evaluated. Give a counterexample to the claim that this yields the best parenthesization.

(g) Give an $O(n^3)$ algorithm to find the optimal ordering, using dynamic programming.

**Source:** Ian Parberry, section 8.1 problem 384-389 and Baase, problem 6.2 and 6.4

5. We wish to compute the laziest way to dial given $n$-digit number on a standard push-button telephone using two fingers. We assume that the two fingers start out on the * and # keys, and that the effort required to move a finger from one button to another is proportional to the Euclidean distance between them. Design and analyze an algorithm that computes in time $O(n)$ the method of dialing that involves moving your fingers the smallest amount of total distance.

**Source:** Ian Parberry, section 13.6 problem 656  (*)

6. A certain string processing language allows the programmer to break a string into two pieces. Since this involves copying the old string, it costs $n$ units of time to break a string of $n$ characters into two pieces. Suppose a programmer wants to break a string into many pieces. The order in which the breaks are made can affect the total amount of time used. For example suppose we wish to break a 20 character string after
characters 3, 8, and 10. If the breaks are made in left-right order, then the first break costs 20 units of time, the second break costs 17 units of time and the third break costs 12 units of time, a total of 49 steps. If the breaks are made in right-left order, the first break costs 20 units of time, the second break costs 10 units of time, and the third break costs 8 units of time, a total of only 38 steps.

Give a dynamic programming algorithm that, given the list of character positions after which to break, determines the cheapest break cost in $O(n^2)$ time.

*Source:* Ian Parberry, section 8.5 problem 410  (*)

7. Consider the problem of examining a string $x = x_1x_2 \ldots x_n$ of characters from an alphabet on $k$ symbols, and a multiplication table over this alphabet, and deciding whether or not it is possible to parenthesize $x$ in such a way that the value of the resulting expression is $a$, where $a$ belongs to the alphabet. The multiplication table is neither commutative or associative, so the order of multiplication matters.

\[
\begin{array}{ccc}
| & a & b & c \\
a | & a & c & c \\
b | & a & a & b \\
c | & c & c & c \\
\end{array}
\]

For example, consider the following multiplication table and the string $b b b a$. Parenthesizing it $(b(b)(b))a$ gives $a$, but $((b(b)b)a)$ gives $c$.

Give an algorithm, with time polynomial in $n$ and $k$, to decide whether such a parenthesization exists for a given string, multiplication table, and goal element.

*Source:* Skiena, problem 3-8  (**)

5.5 Programming Exercises

6 Graph Algorithms

6.1 Simulating Graph Algorithms

1. For the following graph $G_1$: 

![Graph Diagram]
(a) Report the order of the vertices encountered on a breadth-first search of $G_1$ starting from vertex $A$. Break all ties by picking the vertices in alphabetical order (i.e $A$ before $Z$).

(b) Report the order of the vertices encountered on a depth-first search of $G_1$ starting from vertex $A$. Break all ties by picking the vertices in alphabetical order (i.e $A$ before $Z$).

(c) Find the minimum spanning tree of $G_1$.

(d) Find the shortest path spanning tree of $G_1$ rooted in $A$.

Source: Steven Skiena – August 28, 1999.

2. For the following graph $G_2$:

(a) Report the order of the vertices encountered on a breadth-first search of $G_2$ starting from vertex $A$. Break all ties by picking the vertices in alphabetical order (i.e $A$ before $Z$).

(b) Report the order of the vertices encountered on a depth-first search of $G_2$ starting from vertex $A$. Break all ties by picking the vertices in alphabetical order (i.e $A$ before $Z$).

(c) Find the minimum spanning tree of $G_2$.

(d) Find the shortest path spanning tree of $G_2$ rooted in $A$.

Source: Steven Skiena – August 28, 1999.

3. Do a topological sort of following graph $G$: 
6.2 Graph Theory

1. A connected graph is vertex biconnected if there is no vertex whose removal disconnects the graph. A connected graph is edge biconnected if there is no edge whose removal disconnects the graph.

Give a proof or counterexample for each for the following statements:

(a) A vertex biconnected graph is edge biconnected.
(b) An edge biconnected graph is vertex biconnected.

Source: Baase, problem 4.40

2. A tournament is a directed graph formed by taking the complete undirected graph and assigning arbitrary directions on the edges, i.e. a graph $G = (V, E)$ such that for all $u, v \in V$, exactly one of $(u, v)$ or $(v, u)$ is in $E$. Show that every tournament has a Hamiltonian path, that is, a path that visits every vertex exactly once. Give an algorithm to find this path.

Source: Ian Parberry, section 2.10 problem 68 and Brassard and Bratley, problem 9.39 (**)

3. Given two spanning trees $T$ and $R$ of graph $G = (V, E)$, show how to find the shortest sequence of trees $T_0, T_1, \ldots, T_k$ such that $T_0 = T$, $T = R$, and each tree $T_i$ differs from the previous one $T_{i-1}$ by an addition of one edge and a deletion of one edge.

Source: Manber, problem 7.103 (*)

4. A matching of an undirected graph $G = (V, E)$ is a set of edges no two of which have a vertex in common. A perfect matching is a matching in which all vertices are matched.

(a) Construct a graph $G$ with $2n$ vertices and $n^2$ edges such that $G$ has an an exponential number of perfect matchings.

(b) Construct a graph $G$ with $2n$ vertices and $n^2$ edges such that $G$ has exactly one unique perfect matching.
5. Consider an $n \times n$ board of checkerboard $B$ of alternating black and white squares. Assume that $n$ is even. We seek to cover this checkerboard with rectangular dominoes of size $2 \times 1$.

(a) Show how to cover the board with $n \times n/2$ dominoes.

(b) Remove the upper left and lower right corners from $B$. Show that you cannot cover the remaining board with $n \times n/2 - 1$ dominoes.

(c) Remove one arbitrary black square and one arbitrary white square from $B$. Show that the rest of the board can be covered with $n \times n/2 - 1$ dominoes. (Hint: think about a Hamiltonian cycle in the dual graph of $B$).

Source: Manber, problem 2.35 (**)

6.3 BFS / DFS

1. Give a depth-first search algorithm which outputs the list of the edges in the depth-first search tree.

Source: Baase, problem 4.21

2. Present correct and efficient algorithms to convert between the following graph data structures, for an undirected graph $G$ with $n$ vertices and $m$ edges. You must give the time complexity of each algorithm.

(a) Convert from an adjacency matrix to adjacency lists.

(b) Convert from an adjacency list to an incidence matrix. An incidence matrix $M$ has a row for each vertex and a column for each edge, such that $M[i, j] = 1$ if vertex $i$ is part of edge $j$, otherwise $M[i, j] = 0$.

(c) Convert from an incidence matrix to adjacency lists.

Source: Skiena, problem 4-1

3. Prove that if $G$ is an undirected connected graph, then each of its edges is either in the depth-first search tree or is a back edge.

Source: Baase, problem 4.22

4. Give an $O(n)$ algorithm to test whether an undirected graph contains a cycle.

Source: CLR, problem 23.4-3 (*)

5. Suppose $G$ is a connected undirected graph. An edge $e$ whose removal disconnects the graph is called a bridge. Must every bridge $e$ be an edge in a depth-first search tree of $G$, or can $e$ be a back edge? Give a proof or a counterexample.

Source: Skiena, problem 4-4
6. Suppose an arithmetic expression is given as a tree. Each leaf is an integer and each internal node is one of the standard arithmetical operations ($+, -, \times, \div$). For example, the expression $2 + 3 \times 4 + (3 \times 4)/5$ could be represented by the tree in Figure

![Expression Tree](image)

Give an $O(n)$ algorithm for evaluating such an expression, where there are $n$ nodes in the tree.

**Source:** Skiena, problem 4-7

7. Suppose an arithmetic expression is given as a DAG (directed acyclic graph) with common subexpressions removed. Each leaf is an integer and each internal node is one of the standard arithmetical operations ($+, -, \times, \div$). For example, the expression $2 + 3 \times 4 + (3 \times 4)/5$ could be represented by the DAG below:

![DAG](image)

Give an $O(n + m)$ algorithm for evaluating such a DAG, where there are $n$ nodes and $m$ edges in the DAG. Hint: modify an algorithm for the tree case to achieve the desired efficiency.

**Source:** Skiena, problem 4-8 (*)

8. A *bipartite* graph is a graph whose vertices can be partitioned into two subsets such that there is no edge between any two vertices in the same subset. Give a linear-time algorithm to determine if a graph $G$ is bipartite.

**Source:** CL/R, problem 23.2-6 and Baase, problem 4.26 and 4.59.

9. Give a linear algorithm to compute the chromatic number of graphs where each vertex has degree at most 2. Must such graphs be bipartite?
10. For an undirected graph $G = (V, E)$, and three vertices $a$, $b$, and $c$, decide in linear time whether there exists a cycle in $G$ that contains both $a$ and $b$ but does not contain $c$.

Source: Manber, problem 7.84

11. The problem of testing whether a graph $G$ contains a Hamiltonian path is NP-hard, where a Hamiltonian path $P$ is a path that visits each vertex exactly once. There does not have to be an edge in $G$ from the ending vertex to the starting vertex of $P$, unlike in the Hamiltonian cycle problem.

Given a directed acyclic graph $G$ (a DAG), give an $O(n + m)$-time algorithm to test whether or not it contains a Hamiltonian path. (Hint: think about topological sorting and DFS.)

Source: Skiena, problem 6-6 and Manber, problem 7.79 (*)

12. Your job is to arrange $n$ rambunctious children in a straight line, facing front. You are given a list of $m$ statements of the form “$i$ hates $j$”. If $i$ hates $j$, then you do not want put $i$ somewhere behind $j$, because then $i$ is capable of throwing something at $j$.

(a) Give an algorithm that orders the line, (or says that it is not possible) in $O(m + n)$ time.

(b) Suppose instead you want to arrange the children in rows, such that if $i$ hates $j$ then $i$ must be in a lower numbered row than $j$. Give an efficient algorithm to find the minimum number of rows needed, if it is possible.

Source: Ian Parberry, section 13.6 problem 667-668 (*)

13. A mother vertex in a directed graph $G = (V, E)$ is a vertex $v$ such that all other vertices $G$ can be reached by a directed path from $v$.

(a) Give an $O(n + m)$ algorithm to test whether a given vertex $v$ is a mother of $G$, where $n = |V|$ and $m = |E|$.

(b) Give an $O(n + m)$ algorithm to test whether graph $G$ contains a mother vertex.

Source: Baase, problem 4.57 (*)

14. An articulation vertex of a graph $G$ is a vertex whose deletion disconnects $G$. Let $G$ be a graph with $n$ vertices and $m$ edges. Give a simple $O(n + m)$ algorithm for finding a vertex of $G$ that is not an articulation vertex, i.e., whose deletion does not disconnect $G$.

Source: Skiena, problem 4-10 and Manber, problem 7.27 (*)
15. Following up on the previous problem, give an $O(n+m)$ algorithm that finds a deletion order for the $n$ vertices such that no deletion disconnects the graph. (Hint: think DFS/BFS.)

*Source:* Skiena problem 4-11 (*)

16. An undirected graph $G = (V,E)$ is a *split graph* if its vertices can be partitioned into two disjoint sets $U$ and $W$ such that the graph induced by $U$ has no edges and the graph induced by $W$ is a complete graph (i.e. all the edges are present).

(a) Give an example of a graph which is not a split graph.

(b) Design a linear-time algorithm to determine whether a given graph is a split graph.

*Source:* Manber, problem 7.117 (*)

### 6.4 Minimum Spanning Tree

1. Prove that if the weights of the edges of a connected graph $G$ are distinct, then $G$ has a unique minimum spanning tree.

*Source:* Baase, problem 4.3

2. Can Prim’s and Kruskal’s algorithm yield different minimum spanning trees? Explain why or why not.

*Source:* Shaffer, chapter 7 problem 7.20

3. Does either Prim’s and Kruskal’s algorithm work if there are negative edge weights? Explain why or why not.

*Source:* Shaffer, chapter 7 problem 7.22

4. Suppose the cost of laying a telephone cable from point $a$ to point $b$ is proportional to the Euclidean distance from $a$ to $b$. A certain number of towns are to be connected at minimum cost. Find an example where it costs less to lay the cables via an exchange situated in between the towns then to use only direct links.

*Source:* Brassard and Bratley, problem 6.8

5. Is the path between a pair of vertices in a minimum spanning tree necessarily a shortest path between the two vertices in the full graph? Give a proof or a counterexample.

*Source:* Skiena, problem 4-2

6. Suppose we are *given* the minimum spanning tree $T$ of a given graph $G$ (with $n$ vertices and $m$ edges) and a new edge $e = (u,v)$ of weight $w$ that we will add to $G$. Give an efficient algorithm to find the minimum spanning tree of the graph $G + e$. Your algorithm should run in $O(n)$ time to receive full credit, although slower but correct algorithms will receive partial credit.

*Source:* Skiena, problem 4-13 (*)
7. In the minimum product spanning tree problem, the cost of a tree is the product of all the edge weights in the tree, instead of the sum of the weights. You may assume that all edges have positive weight.

(a) Give a graph whose minimum product spanning tree is different than the minimum weight spanning tree.

(b) Give an efficient algorithm to compute the minimum product spanning tree. (Hint: think logarithms).

*Source*: Manber, problem 7.69 (*)

8. Modify Prim’s algorithm so that it runs in time $O(n \log k)$ on a graph that has only $k$ different edges costs.

*Source*: Ian Parberry, section 9.3 problem 449 (*)

### 6.5 Shortest Path

1. The single-destination shortest path problem for a directed graph is to find the shortest path from every vertex to a specified vertex $v$. Give an efficient algorithm to solve the single-destination shortest paths problem.

*Source*: Shaffer, chapter 7 problem 7.13

2. Can we modify Dijkstra’s algorithm to solve the single-source longest path problem by changing minimum to maximum? If so, then prove your algorithm correct. If not, then provide a counterexample.

*Source*: Ian Parberry, section 9.2 problem 436 (*)

3. Let $G = (V, E)$ be a weighted acyclic directed graph with possibly negative edge weights. Design a linear-time algorithm to solve the single-source shortest-paths problem from a given source $v$.

*Source*: Manber, problem 7.50 (*)

### 6.6 Design Problems

1. An undirected graph $G = (V, E)$ is called injective if the edges can be directed such that each vertex has in-degree 1.

   (a) Prove that each connected component of an injective graph must contain exactly one cycle.

   (b) Give a linear-time algorithm to so orient the edges of an injective graph.

*Source*: Manber, problem 7.35
2. A matching in a graph is a set of disjoint edges, i.e. edges which do not share any
vertices in common. Give a linear-time algorithm to find a maximum matching in a
tree.

Source: Manber, problem 7.95

3. Given an undirected graph $G$ with $n$ vertices and $m$ edges, and an integer $k$, give an
$O(m + n)$ algorithm that finds the maximum induced subgraph $H$ of $G$ such that each
vertex in $H$ has degree $\geq k$, or prove that no such graph exists. An induced subgraph
$F = (U, R)$ of a graph $G = (V, E)$ is a subset of $U$ of the vertices $V$ of $G$, and all edges
$R$ of $G$ such that both vertices of each edge are in $U$.

Source: Skiena, Problem 4-9 (*)

4. The square of a directed graph $G = (V, E)$ is the graph $G^2 = (V, E^2)$ such that
$(u, w) \in E^2$ iff for some $v \in V$, both $(u, v) \in E$ and $(v, w) \in E$; i.e. there is a path of
exactly two edges.

Give efficient algorithms for both adjacency lists and matrices.

Source: CLR, problem 23.1-5 (*)

5. A vertex cover of an undirected graph $G = (V, E)$ is a set of vertices $U$ such that each
edge in $E$ is incident on at least one vertex of $U$.

(a) Give an efficient algorithm to find a minimum-size vertex cover if $G$ is a tree.

(b) Let $G = (V, E)$ be a tree with weights associated with the vertices such that
the weight of each vertex is equal to the degree of that vertex. Give an efficient
algorithm to find a minimum-weight vertex cover of $G$.

(c) Let $G = (V, E)$ be a tree with arbitrary weights associated with the vertices. Give
an efficient algorithm to find a minimum-weight vertex cover of $G$.

Source: Manber, problem 7.108 and 7.109, and CLR, problem 37.1-3

6. An interval graph is an undirected graph $G = (V, E)$ whose vertices correspond to
intervals on the real line, where each interval is specified by a leftmost value $v_1$ and
a rightmost value $v_2$. Two vertices in $G$ are connected iff the corresponding intervals
overlap. Let $G$ be an interval graph whose corresponding intervals are provided. Give
an efficient algorithm to find a maximum independent set in $G$.

Source: Manber, problem 7.116

7. Let $G = (V, E)$ be a binary tree. The distance between two vertices in $G$ is the length of
the path connecting these two vertices, and the diameter of $G$ is the maximal distance
over all pairs of vertices. Give a linear-time algorithm to find the diameter of a given
tree.

Source: Manber, problem 5.15 (*)
8. When a vertex and its incident edges are removed from a tree, a collection of subtrees remains, as in the example below:

![Diagram of a tree with a removed vertex]

Give a linear-time algorithm that, for any tree with $n$ vertices, finds a vertex whose removal leaves no subtree with more than $n/2$ vertices.

Source: Baase, problem 4.60 (*)

9. A directed graph $G = (V, E)$ is called unilateral if for all pairs of vertices $v$ and $w$ in $G$, at least one of them is reachable from the other.

(a) Prove that every strongly connected graph is unilateral.

(b) Give an example of a graph which is unilateral but not strongly connected.

(c) Give a linear-time algorithm to determine whether a given directed graph $G$ is unilateral. Hint: Consider the strongly connected components graph.

Source: Manber, problem 7.93 (*)

6.7 Other Graph Problems

1. A permutation matrix is an $n \times n$ binary matrix where each row and column has exactly one nonzero entry. A permutation matrix can be represented by a permutation $P$ such that $P[i] = j$ if the $i$th row contains its 1 in the $j$th column.

(a) Prove that the product of two permutation matrices is another permutation matrix.

(b) Give a $O(n)$ algorithm to multiply two permutation matrices, where the input and output use the representation above.

Source: Manber, problem 9.22 (*)

6.8 Programming Exercises

7 Special Topics

7.1 Lower Bounds

1. Mr. B. C. Dull claims to have developed a new data structure for priority queues that supports the operations Insert, Maximum, and Extract-Max, all in $O(1)$ worst-case time. Prove that he is mistaken. (Hint: the argument does not involve a lot of gory
details—just think about what this would imply about the $\Omega(n \log n)$ lower bound for sorting.)

Source: Skiena, problem 2-10  (*)

2. Show that there is no sorting algorithm which sorts at least $(1/2^n) \times n!$ instances in $O(n)$ time. Show that there is no sorting algorithm which sorts at least $(1/2^n) \times n!$ instances in $O(n)$ time.

Source: CLR, problem 9.1-3  (*)

7.2 Computational Geometry

7.3 Programming Exercises

8 Backtracking and Combinatorial Search

8.1 Backtracking

8.2 Combinatorial Generation

8.3 Programming Exercises

9 Intractable Problems and Approximation Algorithms

9.1 Reductions

1. Draw the graph that results from the reduction of 3-SAT to vertex cover for the expression

\[ (x + \overline{y} + z) \cdot (\overline{x} + y + \overline{z}) \cdot (\overline{x} + y + z) \cdot (x + \overline{y} + \overline{x}) \]

Source: Manber, problem 11.5

2. The baseball card collector problem is as follows. Given packets $P_1, \ldots, P_m$, each of which contains a subset of that year’s baseball cards, is it possible to collect all the year’s cards by buying $\leq k$ packets?

For example, if the players are \{Aaron, Mays, Ruth, Skiena\} and the packets are

\{\{Aaron, Mays\}, \{Mays, Ruth\}, \{Skiena\}, \{Mays, Skiena\}\},

there does not exist a solution for $k = 2$ but there does for $k = 3$, such as

\{Aaron, Mays\}, \{Mays, Ruth\}, \{Skiena\}

Prove that the baseball card collector problem is NP-hard using a reduction from vertex cover.

Source: Skiena, problem 6-3
3. An *Eulerian cycle* is a tour that visits every edge in a graph exactly once. An *Eulerian subgraph* is a subset of the edges and vertices of a graph that has an Eulerian cycle. Prove that the problem of finding the number of edges in the largest Eulerian subgraph of a graph is NP-hard. (Hint: the Hamiltonian circuit problem is NP-hard even if each vertex in the graph is incident upon exactly three edges.)

*Source:* Skiena, problem 6-5 (*)

4. The *low degree spanning tree problem* is as follows. Given a graph $G$ and an integer $k$, does $G$ contain a spanning tree such that all vertices in the tree have degree *at most* $k$ (obviously, only tree edges count towards the degree)? For example, in the following graph, there is no spanning tree such that all vertices have degree less than three.

   lowdegree2.0in

   (a) Prove that the low degree spanning tree problem is NP-hard with a reduction from Hamiltonian *path*.

   (b) Now consider the *high degree spanning tree problem*, which is as follows. Given a graph $G$ and an integer $k$, does $G$ contain a spanning tree whose highest degree vertex is *at least* $k$? In the previous example, there exists a spanning tree of highest degree 8. Give an efficient algorithm to solve the high degree spanning tree problem, and an analysis of its time complexity.

   *Source:* Skiena, problem 6-5

5. Prove that subgraph isomorphism is *NP-complete*.

   *Source:* CLR, problem 36.5-1 (*)

6. Prove that the *clique, no-clique* problem is NP-hard:

   *Input:* An undirected graph $G = (V, E)$ and an integer $k$.

   *Output:* Does $G$ contain a clique of size $k$ and an independent set of size $k$.

   *Source:* Manber, problem 11.13

7. (a) Give a linear-time reduction of the problem of finding the maximum element in an array to sorting. What does this reduction tell us about the upper and lower bounds to the problem of finding the maximum element in a sequence?

   (b) Can we not reduce the problem of sorting to that of finding the maximum element in linear time?

   *Source:* Shaffer, problem 15.6

8. Show that the *strong 3-SAT* problem can be solved in polynomial time:

   *Input:* A Boolean formula $F$ in conjunctive normal form with at most three literals per clause.

35
Output: Is there a truth assignment to the variables of F such that at least two literals in each clause of F are true?

Source: Ian Parberry, section 12.1 problem 581 (*)

9.2 Complexity Theory
9.3 Approximation Algorithms
9.4 Programming Exercises
10 Randomized Algorithms
10.1 Random Numbers
10.2 Design Problems
10.3 Programming Exercises
11 Implementation Issues
11.1 The RAM Model

1. Suppose you optimize the performance of a program by tuning up the most heavily-used parts of the code. However, you are careful not to change the underlying algorithm. Would you expect the efficiency gain to be (a) by a constant factor, whatever the problem being solved or (b) should it increase proportionally greater as the problem size increases? Justify your answer.

Source: Brassard and Bratley, problem 2.5

2. By using virtual memory, a programmer can build programs and data structures which are larger than the actual memory available on his machine. Does this mean that the amount of storage used by an algorithm is not important in practice? Justify your answer.

Source: Brassard and Bratley, problem 2.4

11.2 Thought Problems

1. With computers getting faster and faster, and programmers getting more and more expensive, does it ever really pay to optimize programs for efficiency? List several applications where increased efficiency is important, and others where it doesn’t really matter.

Source: Brassard and Bratley, problem 2.3

11.3 Programming Exercises