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, 2008

This is the same list of problems as distributed in Fall 2008, although of course the actual homework and exam problems are subject to change.

1 Mathematical Preliminaries and Analysis

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:
\[
\begin{align*}
\text{a) } & \quad n^2 \log_2 n, \quad 3^n, \quad 2^n, \quad \log_2 n, \quad n^2 \\
\text{b) } & \quad n = 2^\Theta(n^3) \\
\text{c) } & \quad n = 2^\Theta(n^3) \\
\end{align*}
\]

2. Arrange the expressions by asymptotic growth rate from least to fastest.

Answer:

3. Show that for any real constants \( a \) and \( b \) with \( \frac{a}{b} > 1 \),
\[n + \frac{a}{b} n^2 - O(n^2)\]

Answer: CLR, problem 2.2-2 (*).}

4. \( n = 2^\Theta(n^2) \)

Answer: CLR, problem 2.2-4 (*).

5. For each of the following, compute the answer to the question without using the algorithm. A is faster than algorithm B.

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

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

Answer: Bresenham and Barkey, problem 2.7 and 2.8.

6. 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 \( O(n^3) \). 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 \( O(n^3) \).
- \( A \) has worst-case time complexity \( O(n^2) \).
- \( A \) has worst-case time complexity \( O(n^3) \).

Answer: Skiena, problem 2.1.

7. For each of the following pairs of functions, either \( f(n) \) is in \( O(g(n)) \), \( f(n) \) is in \( \Theta(g(n)) \), or \( f(n) = \Omega(g(n)) \).

Determine which relationship is correct and briefly explain why.

(a) \( f(n) = \log_2(n^3) \) and \( g(n) = \log_2(n^4) \)

Answer: \( f(n) \) is in \( O(g(n)) \).

(b) \( f(n) = \log_2(n^3) \) and \( g(n) = \log_2(n^5) \)

Answer: \( f(n) \) is in \( O(g(n)) \).

(c) \( f(n) = \log_2(n^3) \) and \( g(n) = \log_2(n^4) \)

Answer: \( f(n) \) is in \( O(g(n)) \).

(d) \( f(n) = \log_2(n^3) \) and \( g(n) = \log_2(n^5) \)

Answer: \( f(n) \) is in \( O(g(n)) \).

(e) \( f(n) = \log_2(n^3) \) and \( g(n) = \log_2(n^5) \)

Answer: \( f(n) \) is in \( O(g(n)) \).

(f) \( f(n) = \log_2(n^3) \) and \( g(n) = \log_2(n^5) \)

Answer: \( f(n) \) is in \( O(g(n)) \).

8. List the functions from the least to the greatest order. If any two or more are of the same order, indicate which.

\[
\begin{align*}
\frac{u}{u^2} + \frac{v}{v^2} + \frac{w}{w^2} & \quad \frac{u}{u^2} + \frac{v}{v^2} + \frac{w}{w^2} \\
\frac{u}{u^2} + \frac{v}{v^2} + \frac{w}{w^2} & \quad \frac{u}{u^2} + \frac{v}{v^2} + \frac{w}{w^2} \\
\frac{u}{u^2} & \quad \frac{u}{u^2} \\
\frac{u}{u^2} & \quad \frac{u}{u^2} \\
\frac{u}{u^2} + \frac{v}{v^2} + \frac{w}{w^2} & \quad \frac{u}{u^2} + \frac{v}{v^2} + \frac{w}{w^2} \\
\frac{u}{u^2} & \quad \frac{u}{u^2} \\
\frac{u}{u^2} + \frac{v}{v^2} + \frac{w}{w^2} & \quad \frac{u}{u^2} + \frac{v}{v^2} + \frac{w}{w^2} \\
\frac{u}{u^2} & \quad \frac{u}{u^2} \\
\end{align*}
\]

Answer: Skiena, problem 3.1.
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

1.9 Estimation

1.10 Probability

1.11 Program Analysis

2 Data Structures

2.1 Simple Data Structures

1.5 Proof by Contradiction

1.6 Number Theory

1.8 Logarithms

1.9 Estimation

1.10 Probability

1.11 Program Analysis

2 Data Structures

2.1 Simple Data Structures

3. For \( f_r = i + 1 \), return \( i + 1 \)

4. For \( f_r = i + j \), return \( i + j \)

5. \( r = r + 1 \)

6. return \( \) 

5. Estimate how many cubic miles of water flow out of the mouth of the Mississippi River each day. Do not look up any exponential facts. Describe the assumptions you made in arriving at your answer.

6. A sorting algorithm takes \( O(n^2) \) time to sort \( n \) items on your hard drive. How long will it take to sort 1000 items? Explain:
   (a) if you believe that the algorithm takes time proportional to \( n^2 \), and
   (b) if you believe that the algorithm takes time proportional to \( n \log n \).

5. Estimate how many cities and towns there are in the United States.

6. Estimate how many cubic miles of water flow out of the mouth of the Mississippi River each day. Describe your assumptions.

5. Examine how many cubic miles of water flow out of the mouth of the Mississippi River each day. Do not look up any exponential facts. Describe your assumptions.

6. A sorting algorithm takes \( O(n^2) \) time to sort \( n \) items on your hard drive. How long will it take to sort 1000 items? Explain:
   (a) if you believe that the algorithm takes time proportional to \( n^2 \), and
   (b) if you believe that the algorithm takes time proportional to \( n \).
4. (a) Give an efficient algorithm to find the second-largest key among n keys. You can do better than \( \Theta(n) \) 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: Basic, problem 4.1 and 3.2 (*)

2.3 Union-Find
1. There is a well-known data structure for union and find operations that supports these operations in \( \Theta(\alpha(n)) \) time and amortized time \( \Theta(1) \).
   Design a data structure that can perform a sequence of \( m \) union and find operations on a collection of \( n \) elements consisting of any sequence of all element unions followed by a sequence of all finds.
   Source: Ian Parberry, section 3.2, problem 4.71 (*).

2.4 Trees and Dictionary Structures
1. Design a dictionary data structure in which insertions, deletions and membership queries can be all performed 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 4.2, problem 4.70 (*).
2. Describe how to modify any balanced tree data structure such that search, insert, delete, minimum and maximum can be done in \( O(1) \) time each, but successor and predecessor now take \( O(1) \) time each. Which operations have to be modified to support this?
   Source: Problem S-4.72 (*)
3. Suppose you have recourse to a balanced binary tree data structure, which supports search, insert, delete, minimum and maximum, and predecessor and successor in \( O(1) \) time. Explain how to modify the insert and delete operations so that they still take \( O(1) \) time but now minimum and maximum take \( O(1) \) time. Hint: Think in terms of using the abstract dictionary operations, instead of working with pointers and the tree.
   Source: Shiloah, problem 4.60 (*)
4. You are given the task of making two numbers and then printing them out in sorted order. Suppose you have access to a balanced binary tree data structure, which supports each of the operations search, insert, delete, minimum, maximum, successor, and predecessor in \( O(1) \) time.
   Source: Ian Parberry, section 4.6, problem 4.61 (*)

2.2 Heaps
1. Give an algorithm for finding the minimum element of a non-empty set of \( n \) integers in \( O(\log n) \) time.
   Source: Ian Parberry, section 2.2, problem 4.68 (*)
2. Give an array-based heap on \( n \) elements and a root element \( x \), efficiently determine whether the third smallest key in the heap is greater than or equal to \( x \). Your algorithm should be \( O(\log n) \) in the worst case, independent of the size of the heap. Hint: you may need to find the fifth smallest element in a rooted subtree.
   Source: Milkov, problem 4.69 (*)
3. Give an \( O(n \log n) \) time algorithm which removes it sorted list with a total of \( n \) elements into an unordered list. Hint: use a heap to speed up the elementary \( O(n \log n) \) algorithm.
   Source: CERL, problem 7.25 (*)

(a) Give a \( \Theta(n) \) sorting algorithm using just these operations and two stacks, where you are allowed to temporarily push elements into a constant number of additional registers. [Hint: intervenes sort]
   Source: Shaller, chapter 8, problem 6.2 (*)

5. Prepare a sorting algorithm for any input reason. A simple method trying to translate a short program containing the following kind of statement, with a message complaining it has run out of temporary space:
   case id {
      1: Statement (a)
      2: Statement (b)
      3: Statement (c)
      2: Statement (a)
      4: Statement (a)
   }
   Source: Shiloah, problem 4.63 (*)

6. Design a data structure that allows one to search, insert, and delete an integer \( x \) in \( O(n) \) time on average time, independent of the total number of integers stored. Assume that \( 1 \leq x \leq n \) and that there are \( n \) unit spaces available for the symboltable, whose size is the maximum number of integers that can be in the table at any time. Hint: use two arrays \( A[1..n] \) and \( P[1..n] \).
   Source: Milkov, problem 4.70 (*)
7. Let \( T_1 \) and \( T_2 \) be binary search trees having \( n \) nodes and the exact same set of keys. Prove that by applying the rotations in \( T_1 \) to the shape can be converted to \( T_2 \), [Hint: convert to a right running chain.]
   Source: CERL, problem 10.26 (with Milkov, problem 4.71 (*)

(a) Explain how you can use this dictionary to sort in \( O(n \log n) \) time using only the following abstract operations: minimum, maximum, insert, delete, search.
   Source: Shiloah, problem 4.64 (*)

5. Given the predecesor and successor pointers for 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 preorder and postorder traversal.
   Source: Shiloah, problem 4.65
6. A binary tree 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. Given an algorithm to concentrate binary search tree into one binary search tree. The worst-case running time should be \( O(n) \), where \( k \) is the maximum height of the tree.
   Source: Milkov, problem 4.69
7. Let \( T_1 \) and \( T_2 \) be binary search trees each having \( n \) nodes and the exact same set of keys. Prove that by applying the rotations in \( T_1 \), the shape can be converted to \( T_2 \), [Hint: convert to a right running chain.]
   Source: CERL, problem 10.26 and Milkov, problem 4.71 (*)
8. Design a data structure to support the following operations:
   (a) insert \( x \) into \( T \),
   (b) delete the smallest element from \( T \),
   (c) return the largest element from \( T \).
   All operations must take \( O(\log n) \) time on an unordered set.
   Source: Ian Parberry, section 4.6, problem 4.76 (*)

9. In the heaps algorithm, we see a given meta-data, each weighing between zero and one. We also have a collection of heaps, but fragile-exist. Our goal is to find the smallest number of tellings that will hold the objects, with no losing neither more than one element.
   The step-by-step order for his problem is as follows. Take each of the objects in the order in which they are given. For each object, place it into a partially filled bin which has room. If no such exists, start a new bin. Design an algorithm that implements the above heuristic by taking as input the weights of each object, and outputting the number of bins needed when using \( \Theta(n) \) in \( O(\log n) \) time,
Repeat the above using the back derivative, where we put the next object in the partially filled bin with the smallest amount of extra mass after the object is inserted.

Repeat the above using the front derivative, where we put the next object in the partially filled bin with the largest amount of extra mass after the object is inserted.

Let $\mathbf{A} = [a_1, a_2, \ldots, a_n]$ be an array of real numbers. Design an algorithm to perform any sequence of the following operations:

- **$A[k]$** - add the value $k$ to the list.
- **$P[i]$** - return the sum of the first $i$ numbers, i.e., $\sum_{j=1}^{i} a_j$.

There are no insertions or deletions: the only change is in 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: Mehlhorn, problem 2.6.1†

2.6.1 Extend the data structure of the previous problem to support insertions and deletions.

Each element now has a key $k$ and a value $v$. An element is accessed by its key. The additional operation applies to the value, but the location is accessed by its key. The procedure to update an element $i'$ to $i$ is different.

- **$A[k]$** - add the value $k$ to the list with key $k$.
- **$I[k]$** - insert a new item with key $k$ and value $v$.
- **$D[k]$** - delete the item with key $k$.
- **$P[i]$** - return the sum of all the elements currently in those slots whose keys are less than or equal to $i$.

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

Source: Mehlhorn, problem 2.6.2†

2.6.2 Certain libraries can be modeled as an ordered list of full books $b_1, b_2, \ldots, b_n$, where $b_i$ has a full of $n_i$ books whenever passing fullness $n_i$, for all $1 \leq i \leq n$. Design a data structure that performs all the following operations in $O(\log n)$ time:

- **$I[i]$** - insert a new book behind $b_i$ with full $n_i$.
- **$D[i]$** - delete book $b_i$.
- **$R(i)$** - return the total full of books through index $i$.
- **$B(i, j)$** - add $b_i$ to the full for books $i$ through $j$, inclusive.

Source: Ian Parberry, section 12.2 problems 54.3.†

2.5.2 Programming Exercises

3 Sorting and Searching

3.1 Applications of Sorting

3.1.1 Olivia's 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 which are due, pay them.

(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 these 30 publishers.

(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 times each person checked out at least one book.

Source: Baeza, problem 2.30

2.30 Next Gleipnir is given the job of partitioning 3n players into two teams of $n$ players each. Each player has a numerical rating that measures how good he/she is at the game. Next seeks to divide the players as $n/2$ as possible to make the smallest possible talent imbalance between teams $A$ and team $B$. Show how Next can do this in $O(n \log n)$ time.

Source: Sloman, problem 2.1

2.1 Given two sets $S_1$ and $S_2$ of equal 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 $x$. (The partial credit gives an $O(n^2)$ algorithm for this problem.

Source: Sloman, problem 2.4

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 pass one argument from the book as an argument. For example, $S \in \mathbb{R}$, $(S, \mathbb{N}, \leq)$, $x \in S$ means that $x$ is a number in the set.

(a) Let $S$ be an unsorted list of integers. Give an algorithm that finds the pair $x, y \in S$ such that $x + y = 0$. Your algorithm must run in $O(n \log n)$ time.

(b) Let $S$ be an array of integers. Give an algorithm that finds the pair $x, y \in S$ such that $x + y = n/2$. Your algorithm must run in $O(n \log n)$ time.

(c) Let $S$ be an unsorted array of integers. Give an algorithm that finds the pair $x, y \in S$ such that $x + y = n$. Your algorithm must run in $O(n \log n)$ time.

(d) Let $S$ be an array of integers. Give an algorithm that finds the pair $x, y \in S$ such that $x + y = 2n$. Your algorithm must run in $O(n \log n)$ time.

Source: Sloman, problem 3.3.1

5. Give an efficient algorithm to determine the number of empty and full lists. Show 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: Mehlhorn, problem 4.2

6. The next and kth problem is defined as follows. You are given a collection of $n$ lists of different lengths, and $m$ corresponding units. You can take either a given list and build together from which you know whether the list is large or small, or any small to form the list. The differences in size between pairs of lists on each can be too small to be seen as real, so you can only compute the sums of entire lists or two lists directly. You are to match each list to each other.

(a) Give an $O(n^2)$ algorithm to solve the next and kth problem.

(b) Suppose that instead of matching all of the lists and kth, you wish to find the smallest list and its corresponding unit. Show that this can be done in only $O(n)$ comparisons.

(c) Show that any algorithm for the next and kth problems must take $O(n \log n)$ comparisons in the worst case. (Hint: sorting rearranges $n \log n$ comparisons in the worst case.

Source: Ian Parberry, section 12.3 problems 6 and 6.5.†

3.2 Quicksort

3.2.1 The running time for Quicksort depends on 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 $O(n^2)$ time if the data is already sorted.

(a) Suppose we always pick the pivot element to be the last from the last position of the要素。 On the worst case, does Quicksort now take $O(n \log n)$, or $O(n^2)$?

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

(c) Suppose we always pick the pivot element to be the index of the median element of the first $n/2$ elements of the要素。 On a sorted array, does Quicksort now take $O(n \log n)$, or $O(n^2)$?

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

Source: Sloman, problem 2.9-21

2.9-21 Suppose an array $A$ consists of $n$ elements, each of which is red, green, or blue. We seek to sort the elements so that all the same color before all the others, which came before all the rest. The only operation permitted on the array are

- $\operatorname{Examine}(i)$ - report the color of the $i$th element of $A$.
- $\operatorname{Set}(i, c)$ - make the $i$th element of $A$ to color $c$.

A correct and efficient algorithm for color-based sorting. There is a linear-time solution.

Source: Baeza, problem 2.25†

3.3 MergeSort

3.4 Other Sorting Algorithms

3.4.1 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 $\pi$ have exactly $n(n-1)/2$ inversions?

(b) Let $P$ and $P'$ be the reversal of this permutation. Show that $P$ and $P'$ 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: Baeza, problem 2.26
3.5 Searching

1. Consider the minimum spanning tree problem. In this game, player 1 thinks of a number in the range $0, 1, 2, ..., 9$. Player 2 has to guess the number by asking the least number of yes/no questions. Assume that nobody lies.

(a) What is an optimal strategy if Player 2 is forced to play?

(b) What is a good strategy if Player 2 is not forced to play?

Source: Miners, problem 62.1 (2)

3.6 Programming Exercises

4 Divide and Conquer

4.1 Binary Search

1. Suppose you are given an array $A$ of sorted numbers that have been circularly shifted $k$ positions to the right. For example, $[2, 3, 4, 5, 6]$ is a sorted array that has been circularly shifted 2 positions, while $[6, 7, 8, 9, 0]$ has been circularly shifted 3 positions.

(a) Suppose you know what $k$ is. Given an $O(1)$ algorithm to find the largest number in $A$.

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

Source: Skiena, problem 36.1 (2)

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

Source: Skiena, problem 36.2 (2)

3. Given an array $A$ of numbers ordered by value, modify the binary search routine to return the position of the first value with value $x$ in the structure where $x$ can appear multiple times in the array. Be sure that your algorithm is $O(lg n)$, that is, do not count a searchMarkers since an occurrence of $x$ is found.

Source: Slader, chapter 3.3.2 (2)

4. Suppose that $S$ and $F$ are sorted sets, and containing $n$ distinct elements.

(a) Given an O($n$) algorithm to find the ith smallest of the $n$ elements.

(b) Give a linear faster algorithm for the problem.

Source: Slader, problem 3.1.5 (2)

4.2 Recursive Algorithms

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

Source: Slader, problem 2.2.3 (2)

2. A recursive algorithm finds both the largest and smallest elements in an array of $n$ values. Design and analyze a divide-and-conquer recursive algorithm that uses $\Theta(lg n)$ comparisons for any integer $n$. Before considering the case where $n$ is a power of 2.

Source: Ian Parberry, section 7.1, problem 37.5 and Bauer, problem 3.2.1 (2)

3. Give an efficient divide-and-conquer algorithm to find the $k$ largest elements in the merge of two sorted sequences $S_1$ and $S_2$. The best algorithm runs in time $O(k \log n + 2k)$.

Source: Ian Parberry, section 7.2, problem 37.6 (2)

4. Given an array of $n$ and numbers, consider the problem of finding the maximum sum in any contiguous subarray of the input. For example, in the array

$$ [21, -5, -7, 2, 10, -3, -10, 2] $$

the maximum is achieved by summing the third through seventh elements, where $2 + 10 + (-3) + (-10) + 2 = 0$. When all numbers are positive, the entire array is the answer, while when all numbers are negative, an empty array maximizes the total at 0.

(a) Give a simple, clear, and correct $O(n)$-time algorithm to find the maximum continuous subarray.

(b) Give a $O(n)$ outer 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, section 36.5, Bauer, problem 2.2.7 and Ian Parberry, section 7.6 problem 37.7 (2)

5. Let $X = \{x_1, x_2, ..., x_n\}$ be a sequence of arbitrary real numbers. Give a linear time algorithm to find the subsequence of maximum sum $x_{i_1}, x_{i_2}, ..., x_{i_k}$ whose product is maximum over all consecutive subsequences. The product of the empty subsequence is defined as 1. Observe that $X$ can contain only real less than 1, including negative numbers.

Source: Miners, problem 5.2.7 (2)

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 coins at most until it is equal to $n$, then return the remaining change. For each of the following change amounts, determine 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 coins, which consists of half-dollars (50 cents), quarters (25 cents), dimes (10 cents), nickels (5 cents), and pennies.

(b) English coins before the decimalisation, which consisted of farthings (1/4 pence), florins (20 pence), shillings (12 pence), crown (5 pence), half-crowns (25 pence), and farthings (1/4 pence).

Source: Slader and Slader, problem 62.1, 62.2, and 63.1 (2)

2. In the reservoir-carrying problem, we are given a set $A$ of $n$ jobs. Each job $i$ has a processing time $t_i$ and a deadline $d_i$. The jobs are all scheduled on one processor of size $D < d_i$ for all $i$, and the deadlines are all respected by the latest of any earlier than $d_i$, or after the latest time. The greedy algorithm for this problem is to sort the jobs by earliest deadline first, then give the feasible schedule exists, then the schedule produced by this greedy algorithm is feasible.

Source: Slader, section 7.2, problem 67.1 (2)

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 cannot be rearranged. Therefore, we can speak of a book $b_i$ where $i \geq 0$, that has a thickness $t_i$ and height $h_i$. The length of each bookshelf is fixed to be $L = E$.

(a) Suppose all the books have the same height: $h_0 = h_1 = h_2$ for all 3, and the shelves are all separated by a distance greater than $d_i$, or any book fits on any shelf. The greedy algorithm would fill the shelf first with as many books as we can until we get the smallest book that $b_3$ does not fit, and then repeat with subsequent books.

(b) Suppose that the greedy algorithm always finds the optimal shelf placement, and analyze its time complexity.

Source: Slader, problem 34 (2)

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

(a) Give a complete tree that the greedy algorithm for storing each shelf as tall as possible does not always give the minimum overall height.

(b) Give an algorithm for this problem, and analyze its time complexity. Hint: use dynamic programming.

Source: Slader, problem 38 (2)
5.2 Fibonacci Numbers

5.3 Approximate and Exact String Matching

1. The longest common substring (or 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 "banana" and "orange" is simply "an".

(a) Let \( m = |X| \) and \( n = |Y| \). Then a dynamic programming algorithm for longest common subsequence based on the longest common subsequence fails.

(b) Give a simpler dynamic algorithm which does not rely on dynamic programming.

(c) Briefly Given an O(n^2) algorithm for longest common subsequence in suffix trees.

Source: Baase, problem 6.26 [*]

2. Let \( A = a_1, a_2, \ldots, a_n \) and \( B = b_1, b_2, \ldots, b_m \) be two character strings of the same length. Give an O(mn) algorithm to determine whether \( B \) is a cyclic shift of \( A \). If \( B \) is a cyclic shift of \( A \) then there exists an index \( 1 \leq k \leq n \) such that \( A_{k+1} = b_1, A_{k+2} = b_2, \ldots, A_n = b_{m-k+1}, A_1 = b_{m-k+2} \).

Note that testing whether string \( S \) is a shifting of \( T \) can be done in \( O(|S|) \) time.

Source: Manber, problem 6.46 and Baase, problem 5.28 [*]

3. Give an O(n) algorithm to find the longest maximally increasing sequence in a sequence of \( n \) numbers.

Source: CLR, problem 30.25 [*]

4. Suppose you are given three strings of characters: \( X, Y, \) and \( Z \). Give \( [X \rightarrow Y], [Y \rightarrow Z], \) and \( [Z \rightarrow Y, Z \rightarrow X] \). Note that these subshifts \( X \) to \( Y \) and \( Y \) to \( Z \) can be formed by interchanging the characters from \( X, Y, \) and \( Z \) in any way that maintains the left-to-right ordering of the characters from each string.

Note that this algorithm is a subshift of characters and looks, but does not interchange it.

Give an efficient dynamic programming algorithm that determines whether \( Z \) is a subshift of \( X \) and \( Y \). Hint: The values the dynamic programming entry you construct should be 0/1 rather than 0/1.

Source: Baase, problem 6.27 [*]

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. We are given an \( X \times Y \) matrix \( B \), where \( B[i,j] = 1 \) if and only if the intersection between streets \( i \) and \( j \) is somewhere we wish to avoid.

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

(b) Give an \( O(Y^2) \) 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 equal length. For partial credit, give a \( O(XY) \) algorithm.

Source: Skiena, problem 3.1.1 [*]

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. Unfortunately, the city has bad neighborhoods, which are defined as intersections we do not want to walk. We are given an \( X \times Y \) matrix \( B \), where \( B[i,j] = 1 \) if and only if the intersection between streets \( i \) and \( j \) is somewhere we wish to avoid.

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

Give an algorithm that takes the array \( B \) and returns the number of safe paths of length \( X + Y - 1 \). For full credit, your algorithm must run in \( O(1) \).

Source: Skiena, problem 3.1.2 [*]

3. The longest palindromic substring is a substring of \( S \) that is also a palindrome. Given a string of length \( n \) and a palindromic root \( R \), find a subset of \( S \) which adds up exactly to \( R \). For example, within \( S \), \( (123, 5, 4) \) is a subset which adds up to \( 8 \) but not \( 7 \).

Find a solution to each of the following algorithms for the longest palindromic substring problem. That is, given a string \( T \) and \( k \) that shows where the longest palindromic substring of length \( k \) is located, determine whether the longest palindromic substring is simply a substring of \( T \) or must it be a substring of \( T \)?

(a) Give the algorithm that simply checks the length of \( k \) to determine whether the longest palindromic substring is simply a substring of \( T \) or must it be a substring of \( T \)?

(b) Give the algorithm that determines the longest palindromic substring of length \( k \) located at \( T \).

(c) Give the algorithm that determines the longest palindromic substring of length \( k \) located at \( T \).

Note for a given dynamic programming algorithm which runs in \( O(n^2) \) time.

Source: Ian Parbery, section 6.2 problem 6.49-67

4. Multiplying an X by Y matrix \( A \) by a Y by Z matrix \( B \) using the standard, simple algorithm takes \( X \times Y \times Z \) multiplications. Since matrix multiplication is commutative, we can paralellize an ordered chain of matrix multiplications between vectors and matrices to yield the same result. The cost of doing the multiplication, however, can differ substantially.

(a) Compute the cost of the three possible parallelizations of the matrix product \( A \times B \times C \times D \), where \( |A| = 3 \times 2 \), \( |B| = 2 \times 5 \), \( |C| = 5 \times 10 \), and \( |D| = 10 \times 2 \).

(b) Construct an example with only dense or sparse matrices in which the most factorizability does not yield the best performance.

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

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

(e) Suppose \( d \) is the largest dimension of a matrix in the ordering. Start by multiplying the two matrices which share the dimension, i.e., matrix \( C \) to matrix \( D \). Repeat this until the product has been evaluated. Give a counterexample to the claim that this yields the best performance.

(f) Suppose \( d \) is the smallest dimension of a matrix in the ordering. Start by multiplying the two matrices which share the dimension, i.e., matrix \( C \) to matrix \( D \). Repeat this until the product has been evaluated. Give a counterexample to the claim that this yields the best performance.

Source: Ian Parbery, section 6.1 problem 6.56 [*]

5. We wish to compute the best order to multiply a matrix chain in a parallel processing system using two processors. Here, this involves taking the old string of \( n \) + 1 elements of time to insert a string of \( n \) characters into two spaces. Suppose a programmer wants to break a string into many flips. 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, 10, 18, and 20. If the breaks are made in left-right order, the first break costs 20 units of time, the second break costs 13 units of time and the third break costs 12 units of time, a total of 45 units of time. If the breaks are made in right-left order, the first break costs 20 units of time, the second break costs 13 units of time, and the third break costs 12 units of time, a total of only 35 units of time.

Give a dynamic programming algorithm that, given the list of character positions after which a break is to be inserted, determines the cheapest way to break the string into \( n \) segments.

Source: Ian Parbery, section 6.2 problem 6.32 [*]

6. Give an algorithm with time polynomial in \( n \) and \( k \), to check whether a boolean expression \( e \) is tautology.

Source: Skiena, problem 3.1.4 [*]

5.5 Programming Exercises

6 Graph Algorithms

6.1 Simulating Graph Algorithms

1. For the following graph \( G \):

(a) Give the adjacency matrix of the graph.

(b) Give the adjacency list of the graph.

(c) Give the adjacency set of the graph.

(d) Give the adjacency tuple of the graph.

(e) Give the adjacency relationship of the graph.

(f) Give the adjacency component of the graph.

(g) Give the adjacency function of the graph.

(h) Give the adjacency property of the graph.

(i) Give the adjacency operation of the graph.

(j) Give the adjacency method of the graph.

(k) Give the adjacency class of the graph.

(l) Give the adjacency interface of the graph.

(m) Give the adjacency protocol of the graph.

(n) Give the adjacency method signature of the graph.

(o) Give the adjacency method implementation of the graph.

(p) Give the adjacency method delegation of the graph.

(q) Give the adjacency method inheritance of the graph.

(r) Give the adjacency method polymorphism of the graph.

(s) Give the adjacency method overloading of the graph.

(t) Give the adjacency method overriding of the graph.

(u) Give the adjacency method overrides of the graph.

(v) Give the adjacency method delegations of the graph.

(w) Give the adjacency method delegations of the graph.

(x) Give the adjacency method delegations of the graph.

(y) Give the adjacency method delegations of the graph.

(z) Give the adjacency method delegations of the graph.

[Diagram of the graph]

24
6.3 BFS / DFS

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

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 list to adjacency list.
   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 list.

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

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

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

6. Suppose an arithmetic expression is given as a tree. Each leaf is an integer and each internal node is one of the standard arithmetic operations (+,-,\times,\div). For example, the expression 2+3*4+5 could be represented by the tree in Figure 25.

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

7. Suppose an arithmetic expression is given as a DAG (directed acyclic graph) with one operation per operation, where each leaf is an integer and each internal node is one of the standard arithmetic operations (+,-,\times,\div). For example, the expression 2+3*4+5 could be represented by the tree in Figure 25.

Given an O(n+m) algorithm for evaluating such a DAG, where there are n nodes and m edges in the DAG. But now design an algorithm for determining which operations achieve the desired efficiency.

Source: Skiena, problem 6–7

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: CLR, problem 22.4–4 and Baase, problem 7.60 and 7.62.

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

Sharma, problem 7.28

The problem of testing whether a graph $G$ contains a Hamiltonian path is NP-complete, where a Hamiltonian path in $G$ is a path that visits each vertex exactly once. There does not have to be an edge in $G$ directing the vertices to the starting vertices of graphs in the Hamiltonian cycle problem.

Given a directed acyclic graph $G = (V,E)$, give an $O(n^2)$ time algorithm to test whether or not $G$ contains a Hamiltonian cycle. (Hint: think about topological sorting and DFS).

Sharma, problems 7.29 and 7.30

6.5 Shortest Path

1. The graph $G = (V,E)$ is an undirected graph with a weight function $d: E \to \mathbb{R}$ that assigns each edge a real number $d_e$. A weight function $d$ is called a metric if it satisfies the triangle inequality $d_{uv} + d_{vw} \geq d_{uw}$ for all $u, v, w \in V$. Give an efficient algorithm to find the shortest path from a specified vertex $s$ to every vertex $v$. (Hint: think Dijkstra's algorithm.)

Sharma, chapter 7, problem 7.33

2. If the graph $G$ in problem 1 is directed, then you must modify Dijkstra's algorithm to so that it considers the direction of the edges. Give the shortest path from $s$ to every vertex $v$. (Hint: think Bellman-Ford algorithm.)

Sharma, problem 7.34

6.6 Design Problems

1. An undirected graph $G = (V,E)$ is called acyclic if there is no cycle in $G$. Give an algorithm to check whether a graph $G$ is acyclic.

Sharma, problem 7.27

4. A matching in a graph $G$ 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.

Sharma, problem 7.25

6. Given an undirected graph $G = (V,E)$ and a weight function $d: E \to \mathbb{R}$ that assigns each edge a real number $d_e$. Give an algorithm to find the minimum weight path from vertex $s$ to vertex $t$ in $G$. (Hint: think Dijkstra's algorithm.)

Sharma, problem 8.40

7. In the minimum path spanning tree problem, the cost of a tree is the sum of the weights of all the edges in the tree. Solve the minimum spanning tree problem for an undirected graph $G = (V,E)$ with non-negative edge weights $d_e$. Give an algorithm to find the minimum spanning tree of a graph $G = (V,E)$. Your algorithm should run in $O(n^2)$ time to receive full credit, although slower but correct algorithms will receive partial credit.

Sharma, problem 8.21

8. Suppose we are given a minimum spanning tree $T$ of a given graph $G$ with vertices $V$ and edges $E$. Give an algorithm to find a minimum spanning tree of the graph $G - e$. Your algorithm should run in $O(n^2)$ time to receive full credit, although slower but correct algorithms will receive partial credit.

Sharma, problem 8.22

9. 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.

Sharma, problem 7.26

10. Given an undirected graph $G = (V,E)$ and a weight function $d: E \to \mathbb{R}$ that assigns each edge a real number $d_e$. Give an algorithm to find the minimum weight path from vertex $s$ to vertex $t$ in $G$. (Hint: think Dijkstra's algorithm.)

Sharma, problem 7.34
6. When a vertex and its incident edges are removed from a tree, a collection of sub-trees remains, as in the example below:

```
           1
          / \
         2   3
        /   /\ \
       4   5   6
```

Given a tree-like algorithm that, for any tree with n vertices, finds a vertex whose removal eliminates an sub-tree with more than n/2 vertices.

Source: Bjarne problem 667 (?)

6.7 Other Graph Problems
1. A permutation matrix is an n x n binary matrix whose rows and columns have exactly one non-zero entry. A permutation matrix can be represented by a permutation P such that F(j) = i if the j-th row contains i in the algorithm.
2. Given a list of permutation matrices, compute the product of the matrices.
3. Given a set of permutation matrices, determine whether there exists a non-empty subset of the matrices whose product is the identity matrix.

Source: Bjarne problem 667 (?)

6.8 Programming Exercises

7 Special Topics
7.1 Lower Bounds

Source: Bjarne problem 667 (?)

7.2 Computational Geometry
7.3 Programming Exercises

8 Backtracking and Combinatorial Search
8.1 Backtracking
8.2 Combinatorial Generation

9 Intractable Problems and Approximation Algorithms
9.1 Reductions

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

\[ e \lor \neg e \lor (e \land \neg f) \lor (f \land \neg e) \lor (e \land \neg f) \lor (f \land \neg e) \lor (e \land \neg f) \lor (f \land \neg e) \lor (e \land \neg f) \lor (f \land \neg e) \]

Source: Bjarne problem 667 (?)

2. The shortest path problem is as follows. Given a directed graph G = (V, E) and a non-negative weight function w: E \to \mathbb{R}_+, find a path from vertex s to vertex t with minimum weight.

Source: CLRS problem 9.3-1 (?)

9.2 Complexity Theory

9.3 Approximation Algorithms

9.4 Programming Exercises

10 Randomized Algorithms
10.1 Random Numbers
10.2 Design Problems

11 Programming Exercises

11.1 The RAM Model

1. Suppose you optimize the performance of a program by tuning each loop until the best-case time is achieved. You choose to tune the largest loop of the algorithm. Would you expect the efficiency gain to be 3x by a constant factor, whenever the problem size is doubled? Justify your answer.

Source: Bjarne and Bjarne, problem 678 (?)

11.2 Thought Problems

1. With computers getting faster and faster, and programmers getting more and more efficient, does it ever really pay to optimize? For example, in practice, there is a significant performance difference for an efficient algorithm with increased efficiency important, and others in which it doesn't really matter.

Source: Bjarne and Bjarne, problem 678 (?)

11.3 Programming Exercises