Krzysztof Wojtas

Solutions to exercises and problems

from

Introduction to Algorithms
Fourth Edition

by Thomas H. Cormen
Charles E. Leiserson
Ronald L. Rivest
Clifford Stein

May 27, 2024

Disclaimer

This document is incomplete and was pre-released early to make easier access to the material for beta readers. Please be aware that the solutions in this document may contain bugs, be untested or badly formatted. Until the material reaches a mature state, I hope to receive feedback via the project's GitHub. There you can also track progress and read about the project's history and future:

`https://github.com/wojtask/clrs4e-solutions`

Contents

Part I Foundations

1 The Role of Algorithms in Computing

1.1 Algorithms

1.1-1 One real-world example that requires sorting is online marketplace search results. When users search for a specific item on e-commerce websites, the search results need to be sorted in a meaningful way to provide the most relevant and helpful information to the users.

An example that requires finding the shortest distance between two points is route planning or navigation systems. These systems are widely used in various applications, such as GPS devices, online mapping services, ride-sharing apps, and logistics management.

1.1-2 In a real-world setting, there are several measures of algorithm efficiency that go beyond just speed. Some of these include:

Memory Usage: The amount of memory required by an algorithm can significantly impact its efficiency. Algorithms that consume excessive memory may not scale well when dealing with large data sets or on systems with limited memory resources. It's important to consider the memory requirements of an algorithm to ensure it can run efficiently in different environments.

Scalability: An algorithm's ability to handle increasingly larger input sizes without a significant degradation in performance is crucial in real-world scenarios. Scalability is particularly important when dealing with big data or high-volume systems where the algorithm needs to process and analyze vast amounts of information.

Parallelizability: In modern computing environments, the ability to parallelize an algorithm and leverage multiple processors or distributed systems can greatly enhance efficiency. Algorithms that can be effectively parallelized can take advantage of hardware resources and potentially reduce overall processing time.

I/O Efficiency: Algorithms that involve reading from or writing to external storage, such as databases or files, need to be mindful of I/O efficiency. Minimizing disk accesses or network transfers and optimizing data retrieval and storage techniques can significantly impact the overall performance of an algorithm.

Energy Consumption: With the increasing emphasis on energy-efficient computing, algorithms that consume less power are desirable in many real-world settings. Optimizing an algorithm to reduce unnecessary computations or limit resource-intensive operations can help conserve energy and improve efficiency, particularly in resource-constrained environments like mobile devices or IoT devices.

Robustness and Reliability: Efficiency should not come at the cost of algorithmic robustness and reliability. In a real-world setting, algorithms need to handle various edge cases, exceptions, and error conditions gracefully. An algorithm that is efficient but lacks robustness may result in incorrect outputs, system failures, or security vulnerabilities.

Maintainability and Modularity: In many real-world scenarios, algorithms need to be maintained, extended, or integrated into existing systems. The efficiency of an algorithm can be influenced by factors like code readability, ease of understanding, and modularity, which impact the ease of maintenance and future enhancements.

1.1-3 We'll discuss the properties of ***arrays*** — one of the simplest non-trivial data structures widely used in computer programming. The strengths of arrays include:

Simplicity: Arrays are a straightforward and easy-to-understand data structure. They have a simple interface and operations like insertion and deletion are well-defined. This simplicity makes arrays a popular choice for beginners and for scenarios where the data size is known in advance.

Random Access: Arrays provide constant-time access to individual elements. Each element in an array can be accessed directly by its index, allowing for efficient random access operations. This makes arrays suitable for scenarios where quick access to elements is required.

Efficiency: Arrays have a fixed size and provide efficient memory allocation. The elements in an array are stored contiguously in memory, allowing for efficient traversal and manipulation of the array. This makes arrays well-suited for operations that involve iterating over elements or performing arithmetic computations.

Arrays also have some limitations that may make them less useful or efficient in certain problems. These are:

Fixed Size: Arrays have a fixed size, meaning they cannot be dynamically resized once created. This can be a limitation when the number of elements in the array needs to change dynamically. Resizing an array typically involves creating a new, larger array and copying the existing elements, which can be inefficient.

Wasted Space: Arrays require contiguous memory allocation, which can lead to wasted space when the array is not fully occupied. If an array is created with a larger capacity than needed, the unused space remains allocated, resulting in wasted memory. This is especially problematic when dealing with large arrays or in situations where memory is a scarce resource.

Insertion and Deletion: Inserting or deleting elements in an array can be inefficient. If an element is inserted or deleted in the middle of the array, all the subsequent elements need to be shifted to accommodate the change. This operation takes the time proportional to the number of elements in the array. Therefore, arrays are not the best choice when frequent insertions or deletions at arbitrary positions are required.

Homogeneousness: Arrays have a fixed data type, which means they can only store elements of a single type. This can be limiting when working with heterogeneous data.

1.1-4 The problems are similar in terms of optimizing routes in a graph, but differ in objectives, constraints, and computational complexity:

- The shortest-path problem finds the minimum cost path between two nodes, while the traveling-salesperson problem finds the shortest route that visits all nodes exactly once and returns to the starting node.
- The shortest-path problem has no constraints, while the traveling-salesperson problem has the constraint of visiting all nodes once.
- The shortest-path problem can be solved efficiently, while the traveling-salesperson problem is computationally challenging and has no known polynomial-time solution.

1.1-5 When providing critical medical diagnoses, we are interested in finding only the best solutions. For instance, in complex cases where a patient's life is at stake, accurate and precise diagnoses are crucial. The healthcare professionals need to identify the exact underlying condition to provide the appropriate treatment plan. In such situations, the highest level of accuracy is required to ensure the patient's well-being and potentially save lives.

A real-world problem where approximately optimal solution is sufficient is travel planning. When planning a vacation or business trip, finding the absolute best itinerary

might not always be necessary. For example, when searching for flights, accommodations, and activities, there are often multiple viable options that meet the basic criteria and provide an enjoyable experience. While it's desirable to find the most cost-effective or time-efficient solution, a reasonably good itinerary that meets most of the traveler's preferences can still lead to a satisfying trip. The goal is to strike a balance between the available options and the constraints, without necessitating an exhaustive search for the absolute best solution.

1.1-6 A real-world problem that fits this description is real-time stock market analysis and trading. Sometimes traders have access to the entire input data before making decisions, allowing them to analyze historical data and market trends. Other times, the input data arrives gradually over time, requiring traders to adapt and make decisions based on evolving real-time information such as news releases and market updates. In both cases, the goal is to make profitable trades based on the available information, but the difference lies in the availability of the data.

1.2 Algorithms as a technology

1.2-1 One example of an application that requires algorithmic content at the application level is a recommendation system for an online video or music streaming platform. These platforms use algorithms to analyze user preferences, historical data, and content metadata in order to provide personalized recommendations to their users. The algorithms involved in these recommendation systems serve several functions:

Content-Based Filtering: Content-based filtering algorithms analyze the attributes or characteristics of the content itself, such as genre, director, actors, or music genre, artist, and lyrics. By building a profile for each user based on their preferences, these algorithms recommend similar content that matches the user's preferences.

Collaborative Filtering: Collaborative filtering algorithms analyze user behavior and preferences to find similarities between users and their viewing or listening habits. This helps identify users with similar tastes and enables the system to recommend content based on what similar users have liked or consumed. Collaborative filtering algorithms can be based on user-based or item-based approaches.

Reinforcement Learning: Some advanced recommendation systems employ reinforcement learning techniques. These algorithms continuously learn and adapt based on user feedback and interactions. They optimize the recommendation process by dynamically adjusting the weights assigned to different features and learning from the user's explicit or implicit feedback (e.g., ratings, likes, skips).

Contextual Factors: Algorithms in recommendation systems also consider contextual factors such as time of day, day of the week, or a user's location. These factors help tailor recommendations to specific situations and user contexts, ensuring that the recommended content is relevant and timely.

The primary goal of these algorithms is to enhance user experience by providing personalized content recommendations. By leveraging historical data, user behavior, and content attributes, the algorithms aim to surface relevant and engaging content that matches the user's preferences and interests.

1.2-2 For natural n the inequality $8n^2 < 64n \lg n$ holds when $2 \leq n \leq 43$. Insertion sort beats merge sort for arrays of such sizes.

1.2-3 The smallest positive integer n satisfying the inequality $100n^2 < 2^n$ is $n = 15$.

Problems

1-1 ***Comparison of running times***

The results are collected in the table in Figure 1-1. Numbers greater than 10^6 are approximate. A month is assumed to have 30 days and a year is assumed to have 365 days.

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
$\lg n$	2^{10^6}	$2^{6 \cdot 10^7}$	$2^{3.6 \cdot 10^9}$	$2^{8.64 \cdot 10^{10}}$	$2^{2.59 \cdot 10^{12}}$	$2^{3.15 \cdot 10^{13}}$	$2^{3.15 \cdot 10^{15}}$
$\sqrt{n}$	10^{12}	$3.6 \cdot 10^{15}$	$1.3 \cdot 10^{19}$	$7.46 \cdot 10^{21}$	$6.72 \cdot 10^{24}$	$9.95 \cdot 10^{26}$	$9.95 \cdot 10^{30}$
n	10^6	$6 \cdot 10^7$	$3.6 \cdot 10^9$	$8.64 \cdot 10^{10}$	$2.59 \cdot 10^{12}$	$3.15 \cdot 10^{13}$	$3.15 \cdot 10^{15}$
$n \lg n$	62,746	$2.8 \cdot 10^6$	$1.33 \cdot 10^8$	$2.76 \cdot 10^9$	$7.19 \cdot 10^{10}$	$7.98 \cdot 10^{11}$	$6.86 \cdot 10^{13}$
n^2	1,000	7,745	60,000	293,938	$1.61 \cdot 10^6$	$5.62 \cdot 10^6$	$5.62 \cdot 10^7$
n^3	100	391	1,532	4,420	13,736	31,593	146,645
2^n	19	25	31	36	41	44	51
$n!$	9	11	12	13	15	16	17

Figure 1-1 Largest sizes of problems solvable in a given time by an algorithm of a given time efficiency.

2 Getting Started

2.1 Insertion sort

2.1-1 See Figure 2.1-1.

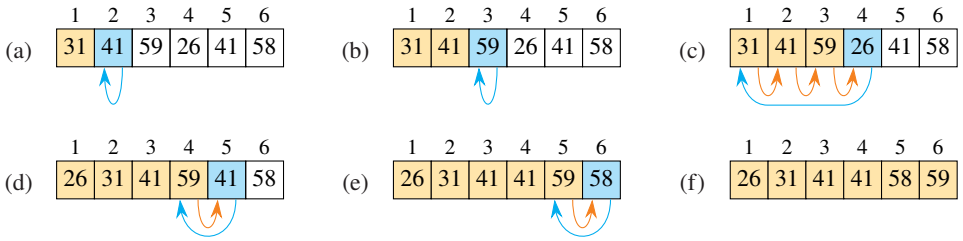


Figure 2.1-1 The operation of INSERTION-SORT(A, n) where A initially contains the sequence $\langle 31, 41, 59, 26, 41, 58\rangle$ and $n = 6$. **(a)–(e)** The iterations of the **for** loop of lines 1–8. **(f)** The final sorted array.

2.1-2 We formulate the loop invariant as follows:

> At the start of each iteration of the **for** loop of lines 2–3, *sum* is the sum of the numbers in the subarray $A[1 : i - 1]$.

Initialization: Before the first loop iteration, $i = 1$. The subarray $A[1 : i - 1]$ is empty, so the sum of its elements is 0, by definition. It is thus equal to the current value of *sum*, just after its initialization in line 1.

Maintenance: Suppose that just before the ith iteration *sum* holds the sum of the numbers in the subarray $A[1 : i - 1]$. The only operation performed in the body of the loop is increasing *sum* by the value $A[i]$, so before the next iteration *sum* contains the sum of the numbers in the subarray $A[1 : i]$. Incrementing i for the next iteration of the loop then preserves the loop invariant.

Termination: The loop terminates once i equals $n+1$. The invariant says that *sum* is now equal to the sum of numbers in the subarray $A[1:n]$. Therefore, the procedure is correct, since it immediately returns *sum* in line 4.

2.1-3 In the INSERTION-SORT procedure we can just reverse the second inequality in the **while** loop condition. In other words, replace line 5 by:

```
while j > 0 and A[j] < key
```

2.1-4 The description leads to the following pseudocode:

```
LINEAR-SEARCH(A, n, x)
1  for i = 1 to n
2      if A[i] == x
3          return i
4  return NIL
```

We'll prove the loop invariant:

> At the start of each iteration of the **for** loop of lines 1–3, the value x does not appear in the subarray $A[1:i-1]$.

Initialization: Prior to the first iteration of the loop, $i = 1$. Then, the subarray $A[1:i-1]$ is empty and as such does not contain x.

Maintenance: Suppose that just before the ith iteration the value x does not appear in the subarray $A[1:i-1]$. The loop will terminate in this iteration, if $A[i]$ equals x. Otherwise, we increment i and move on to the next iteration, so the loop invariant remains true.

Termination: The loop terminates once it encounters i such that $A[i]$ equals x, or after running for n iterations. In the first case the procedure returns the index i. In the second case $i = n+1$ and the procedure returns NIL, which is correct, since by the loop invariant, the value x does not appear in the subarray $A[1:n]$.

2.1-5

```
ADD-BINARY-INTEGERS(A, B, n)
1  let C[0 : n] be a new array
2  carry = 0
3  for i = 0 to n − 1
4       C[i] = (A[i] + B[i] + carry) mod 2
5       carry = ⌊(A[i] + B[i] + carry)/2⌋
6  C[n] = carry
7  return C
```

Once the array C has been created, the **for** loop of lines 3–5 keeps adding the individual bits of numbers a and b. More specifically, it fills array C by summing up the bits in arrays A and B at the same positions, using modular arithmetic and a carry kept in the variable *carry*. Just before returning array C the final carry is put to its last position.

2.2 Analyzing algorithms

2.2-1 $n^3/1000 + 100n^2 - 100n + 3 = \Theta(n^3)$

2.2-2

```
SELECTION-SORT(A, n)
1  for i = 1 to n − 1
2       min = i
3       for j = i + 1 to n
4            if A[j] < A[min]
5                 min = j
6       exchange A[min] with A[i]
```

The outer **for** loop maintains the following invariant:

> Just prior to the ith iteration of the **for** loop of lines 1–6, the subarray $A[1 : i-1]$ contains the $i-1$ smallest elements of array A in sorted order.

According to the invariant, once the loop terminates after performing the $n-1$ iterations, the subarray $A[1 : n-1]$ contains the $n-1$ smallest elements of A in sorted order. The last element then must be greater than or equal to each element of the subarray $A[1 : n-1]$, which means that the $n-1$ iterations suffice to sort the entire array.

In the ith iteration of the outer **for** loop we determine the smallest element of $A[i : n]$. After initializing the position of a potential minimum in line 2, we need to examine the remaining $n - i$ positions in the inner **for** loop. Let's assign statement costs for each line of the algorithm, so that line k has cost c_k. For each $i = 1, 2, \ldots, n - 1$, let t_i denote the number of times line 5 is executed during the ith iteration of the outer **for** loop. Then, the running time of the algorithm on an n-element array is

$$T(n) = c_1 n + c_2(n-1) + c_3 \sum_{i=1}^{n-1}(n-i+1) + c_4 \sum_{i=1}^{n-1}(n-i) + c_5 \sum_{i=1}^{n-1} t_i + c_6(n-1).$$

In the worst case $t_i = n - i$ for each $i = 1, 2, \ldots, n - 1$, and by the fact that

$$\begin{aligned}\sum_{i=1}^{n-1}(n-i+1) &= \sum_{i=2}^{n} i \\ &= \frac{n(n+1)}{2} - 1\end{aligned}$$

and

$$\begin{aligned}\sum_{i=1}^{n-1}(n-i) &= \sum_{i=1}^{n-1} i \\ &= \frac{n(n-1)}{2},\end{aligned}$$

the running time of SELECTION-SORT is

$$\begin{aligned}T(n) &= c_1 n + c_2(n-1) + c_3\left(\frac{n(n+1)}{2} - 1\right) + c_4\left(\frac{n(n-1)}{2}\right) \\ &\quad + c_5\left(\frac{n(n-1)}{2}\right) + c_6(n-1) \\ &= \left(\frac{c_3}{2} + \frac{c_4}{2} + \frac{c_5}{2}\right)n^2 + \left(c_1 + c_2 + \frac{c_3}{2} - \frac{c_4}{2} - \frac{c_5}{2} + c_6\right)n - (c_2 + c_3 + c_6).\end{aligned}$$

Therefore, $T(n)$ can be expressed as $an^2 + bn + c$ for constants a, b, and c that depend on the statement costs c_k, and so $T(n) = \Theta(n^2)$.

The best case differs from the worst case in the values t_i, which all are 0. But this difference only cancels out the terms involving c_5 in the above formula, and so it doesn't affect the rate of growth of the running time of the algorithm for the best case.

2.2-3 By Exercise C.3-2, linear search will check $(n + 1)/2$ elements on average. In the worst case all n elements will need to be examined. Therefore, since the algorithm terminates once either the searched for value has been found, or all elements have been examined, both the average-case and the worst-case running times of linear search are $\Theta(n)$.

2.2-4 A sorted array does not need to be sorted. Any sorting algorithm can be modified so that at the beginning it checks whether the input array is sorted, and immediately exits if it is. The extra step of detecting the ordering of the input elements requires a complete scan of the input array. Thus, the best-case running time of the algorithm for an input array of length n is $\Theta(n)$.

2.3 Designing algorithms

2.3-1 See Figure 2.3-1.

2.3-2 *Instead of $p \neq r$ it should be $p == r$ (in both places). Corrected in fourth printing.*

Lines 3–7 of MERGE-SORT are executed for subarrays $A[p:r]$ of more than one element, i.e., when $p < r$. Note that $p \leq (p+r)/2 \leq r$, and since both p and r are integers,

$$p \leq \lfloor (p+r)/2 \rfloor \leq r.$$

Therefore, the recursive call in line 4 is for a subarray of at least one element. Suppose that the procedure is called in line 5 for an empty subarray. Then, $\lfloor (p+r)/2 \rfloor + 1 > r$ and by the above inequalities, there must be $\lfloor (p+r)/2 \rfloor = r$. This equality holds only when $(p+r)/2 = r$, which implies $p = r$, contradicting the assumption that $p < r$. Therefore, replacing the test in line 1 by "**if** $p == r$" guarantees no recursive call works on an empty subarray.

2.3-3 We use the following loop invariant:

> Prior to each iteration of the **while** loop of lines 12–18, the subarray $A[p:k-1]$ contains the elements of $L[0:i-1]$ and the elements of $R[0:j-1]$, all arranged in sorted order.

The **while** loop of lines 12–18 terminates once $i = n_L$ or $j = n_R$. When it happens, the above invariant says that all elements from at least one array, L or R, have been copied into the subarray $A[p:k-1]$, and that the subarray is sorted. Since both arrays L and R are sorted, all elements in $L[i:n_L-1]$, as well as all elements in $R[j:n_R-1]$, are greater than or equal to all elements in $A[p:k-1]$. To finish merging the arrays L and R into $A[p:r]$, it suffices to copy the subarrays $L[i:n_L-1]$ and $R[j:n_R-1]$ to $A[k:r]$, which is exactly what the **while** loops of lines 20–23 and 24–27 do. In total, the three **while** loops copy exactly $n_L + n_R = r - p + 1$ elements from $L[0:n_L-1]$ (copied from $A[p:q]$) and $R[0:n_R-1]$ (copied from $A[q+1:r]$) to $A[p:r]$, so the MERGE procedure works correctly.

2.3-4 Let $n = 2^k$ for $k \geq 1$. We'll do the proof by induction on k. For $k = 1$ we have $n = 2$ and $T(n) = 2 = 2 \lg 2$, so the base case holds. For the inductive step, assume that $k > 1$, or $n > 2$, and that $T(n/2) = (n/2) \lg(n/2)$. Then,

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2(n/2) \lg(n/2) + n \\ &= n(\lg n - 1) + n \\ &= n \lg n . \end{aligned}$$

2.3-5

```
RECURSIVE-INSERTION-SORT(A, n)
1  if n > 1
2      RECURSIVE-INSERTION-SORT(A, n − 1)
3  key = A[n]
4  j = n − 1
5  while j > 0 and A[j] > key
6      A[j + 1] = A[j]
7      j = j − 1
8  A[j + 1] = key
```

Let $T(n)$ be the worst-case running time of sorting an array $A[1:n]$ with the above algorithm. Because inserting $A[n]$ into the sorted subarray $A[1:n-1]$ takes at most $\Theta(n)$ time, we get the following recurrence:

$$T(n) = \begin{cases} \Theta(1), & \text{if } n = 1, \\ T(n-1) + \Theta(n), & \text{if } n > 1. \end{cases}$$

2.3-6 *Instead of v there should be x. Corrected in third printing.*

Below is recursive pseudocode implementing binary search. The procedure takes as input a sorted array $A[1:n]$, indices p and r, and a value x, and searches for x in the subarray $A[p:r]$. It returns an index q such that x equals $A[q]$ or the special value NIL if x does not appear in A.

```
BINARY-SEARCH(A, p, r, x)
1  if p > r
2       return NIL
3  q = ⌊(p + r)/2⌋
4  if x == A[q]
5       return q
6  if x < A[q]
7       return BINARY-SEARCH(A, p, q − 1, x)
8  else return BINARY-SEARCH(A, q + 1, r, x)
```

The procedure finds the midpoint of the subarray $A[p:r]$ and compares x to it in line 4. Then it discards approximately half of that subarray, and continues searching for x in the other half in the recursive call of line 7 or 8. The procedure ends when it finds x or when the search range becomes empty (i.e., $p > r$), which means that A does not contain x. The recurrence describing the worst-case running time of the algorithm is:

$$T(n) = \begin{cases} \Theta(1), & \text{if } n \le 1, \\ T(n/2) + \Theta(1), & \text{if } n > 1. \end{cases}$$

In Exercise 4.5-3 we show that $T(n) = \Theta(\lg n)$.

2.3-7 *Instead of $A[1:j-1]$ there should be $A[1:i-1]$. Corrected in third printing.*

Using a binary search we can determine the target position in the subarray $A[1:i-1]$ for the element initially occupying the position i. To be able to use here the BINARY-SEARCH procedure presented in Exercise 2.3-6, we need to modify it first so it returns p instead of NIL in line 2. However, moving $A[i]$ on its target position requires shifting a part of the array to the right, which in the worst case takes the time proportional to the length of the sorted subarray. This approach is therefore no better than running a linear search in the INSERTION-SORT procedure, and would not improve the worst-case running time of insertion sort.

2.3-8 Let's represent the set S as an array $A[1:n]$. First, we'll sort A, so we can use a binary search through it (see Exercise 2.3-6). Then we will be scanning A in the sorted order and for an element $A[i]$ we will search for another element in the subarray $A[i+1:n]$, that added to $A[i]$ gives x. That subarray is empty for $i = n$, so we can skip searching for the last item. We can also terminate this procedure once $A[i] \ge x/2$, since all elements in the subarray $A[i+1:n]$ are then greater than $x/2$, and the search will never succeed. Depending on the search result, we return a boolean value (TRUE or FALSE).

```
SUM-SEARCH(A, n, x)
1  MERGE-SORT(A, 1, n)
2  i = 1
3  while i < n and A[i] < x/2
4      if BINARY-SEARCH(A, i + 1, n, x − A[i]) ≠ NIL
5          return TRUE
6      i = i + 1
7  return FALSE
```

Sorting the array $A[1:n]$ in line 1 takes $\Theta(n \lg n)$ time. The procedure BINARY-SEARCH is called in line 4 at most $n-1$ times, for subarrays of decreasing sizes — from $n-1$ down to 1 — so the total time spent in all these calls won't exceed $n \cdot \Theta(\lg n) = \Theta(n \lg n)$.

Problems

2-1 *Insertion sort on small arrays in merge sort*

a. Insertion sort called for a sublist of length k runs in $\Theta(k^2)$ worst-case time. Applied separately to n/k such sublists takes $(n/k) \cdot \Theta(k^2) = \Theta(nk)$ in total.

b. In the modified merge sort we'll check at each recursion level if $r - p + 1 \le k$, and if so we'll sort $A[p:r]$ using insertion sort, ending this recursion branch. The sorted sublists will then be merged as usual by the MERGE procedure. If we considered a recursion tree for the modified algorithm, it would have approximately $\lg(n/k)$ levels, with each level contributing a cost of $\Theta(n)$. The total time spent on merging n/k sublists is therefore $\lg(n/k) \cdot \Theta(n) = \Theta(n \lg(n/k))$.

c. For $k = c \lg n$, where c is a positive constant,

$$\begin{aligned}\Theta(nk + n \lg(n/k)) &= \Theta(cn \lg n + n \lg(n/(c \lg n))) \\ &= \Theta((c+1) \cdot n \lg n - n \lg c - n \lg \lg n) \\ &= \Theta(n \lg n),\end{aligned}$$

since in Θ-notation we can ignore the lower-order terms and the leading term's constant coefficient. Thus, as long as k grows as fast as $c \lg n$ for a constant $c > 0$, the running time of the modified merge sort remains the same as the running time of the standard merge sort, in terms of Θ-notation.

On the other hand, if k grows faster than $c \lg n$ for any constant $c > 0$, then the total time spent on sorting sublists by insertion sort, as shown in part (a), is larger than $n \lg n$ in terms of Θ-notation, and so the whole modified algorithm runs slower than the original algorithm.

d. In practice, k should be the largest list length for which a particular implementation of insertion sort runs faster than a particular implementation of merge sort.

2-2 Correctness of bubblesort

a. We also need to show that A' is a permutation of the array A.

b. The inner loop invariant:

> At the start of each iteration of the **for** loop of lines 2–4, $A[j]$ is the smallest element of the subarray $A[j : n]$.

Initialization: Just before the first loop iteration, $j = n$, so the subarray $A[j : n]$ consists of a single element $A[j]$ and the invariant trivially holds.

Maintenance: By the loop invariant, we assume that just before an iteration, $A[j]$ is the smallest element of $A[j : n]$. Then, $A[j-1]$ is the smallest element of $A[j-1 : n]$, unless $A[j] < A[j-1]$, in which case the elements $A[j]$ and $A[j-1]$ are exchanged in line 4. Decrementing j for the next iteration preserves the invariant.

Termination: The loop terminates, since it is a **for** loop iterating from n down to $i + 1$, and $2 \le i + 1 \le n$. At termination, $j = i$, and $A[i]$ is the smallest element of $A[i : n]$.

c. The outer loop invariant:

> At the start of each iteration of the **for** loop of lines 1–4, the subarray $A[1 : i - 1]$ consists of the $i - 1$ smallest elements of A in sorted order.

Initialization: Prior to the first iteration, $i = 1$, so the subarray $A[1 : i - 1]$ is empty and therefore trivially sorted.

Maintenance: By the assumption that the subarray $A[1 : i - 1]$ is sorted, the largest element of that subarray is $A[i - 1]$. The inner **for** loop places the smallest element of the subarray $A[i : n]$ at position i (which we proved in part (b)). By the loop invariant holding before the iteration, it follows that in the subarray $A[i : n]$ there are no elements smaller than $A[i - 1]$, so in particular $A[i - 1] \le A[i]$. Hence,

the subarray $A[1:i]$ contains the i smallest elements of A in sorted order. By incrementing i for the next loop iteration, we preserve the invariant.

Termination: The loop makes exactly $n-1$ iterations, and so it terminates. When it does, $i = n$, and the subarray $A[1:i-1]$ consists of the $i-1 = n-1$ smallest elements of A in sorted order. Thus, the element at position n must be the largest element of A, so the algorithm sorts correctly.

d. For any input, the **for** loop of lines 2–4 performs $n-i$ iterations for each $i = 1, 2, \ldots, n-1$, and each iteration of that loop takes constant time. The total number of the inner loop iterations is

$$\sum_{i=1}^{n-1}(n-i) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2},$$

so the running time of BUBBLESORT is $\Theta(n^2)$. It is the same as the wost-case running time of INSERTION-SORT, but worse than its best-case running time.

2-3 *Correctness of Horner's rule*

a. The **for** loop of lines 2–3 makes $n+1$ iterations, each taking a constant time, so the running time of HORNER is $\Theta(n)$.

b. The following procedure takes the same set of parameters as HORNER and evaluates $P(x)$ using the naive method.

```
POLYNOMIAL-EVALUATE(A, n, x)
1  p = 0
2  for i = 0 to n
3       s = A[i]
4       for j = 1 to i
5            s = s · x
6       p = p + s
7  return p
```

The **for** loop of lines 4–5 iterates i times for each $i = 0, 1, \ldots, n$, for a total of $\sum_{i=0}^{n} i = n(n+1)/2$ iterations. The running time of POLYNOMIAL-EVALUATE is therefore $\Theta(n^2)$, so the algorithm is less efficient as compared to HORNER.

c. We prove the three properties of the loop invariant:

Initialization: Prior to the first iteration, $i = n$, so

$$\begin{aligned} p &= \sum_{k=0}^{n-(i+1)} A[k+i+1] \cdot x^k \\ &= \sum_{k=0}^{-1} A[k+n+1] \cdot x^k \\ &= 0, \end{aligned}$$

that matches the initial value of p.

Maintenance: During each iteration p gets the value of $A[i] + x \cdot p$. Assuming that the invariant holds before the current iteration, we have

$$\begin{aligned} p &= A[i] + \sum_{k=0}^{n-(i+1)} A[k+i+1] \cdot x^{k+1} \\ &= A[i] \cdot x^0 + \sum_{k=1}^{n-i} A[k+i] \cdot x^k \\ &= \sum_{k=0}^{n-i} A[k+i] \cdot x^k. \end{aligned}$$

After decrementing i for the next iteration, the invariant is preserved.

Termination: The loop terminates, since it performs exactly $n + 1$ iterations. At termination, $i = -1$, so

$$\begin{aligned} p &= \sum_{k=0}^{n-(i+1)} A[k+i+1] \cdot x^k \\ &= \sum_{k=0}^{n} A[k] \cdot x^k. \end{aligned}$$

Therefore, the algorithm correctly evaluates the polynomial $P(x)$ for a given x.

2-4 Inversions

a. $(1, 5)$, $(2, 5)$, $(3, 4)$, $(3, 5)$, $(4, 5)$

b. An array of length n with the most inversions is the one sorted in descending order. In such an array, every pair (i, j), where $1 \le i < j \le n$, is an inversion, and so the array has $\binom{n}{2} = n(n-1)/2$ inversions.

c. For an array $A[1:n]$, by $\text{inv}(i)$ let us denote the number of inversions (j, i) of A. In the INSERTION-SORT procedure, the **while** loop of lines 5–7 moves the element initially at position i by $\text{inv}(i)$ positions to the left, thus eliminating exactly $\text{inv}(i)$ inversions. The **while** loop runs for each $i = 2, 3, \ldots, n$, for a total of $I = \sum_{i=2}^{n} \text{inv}(i)$ iterations, which is exactly the number of inversions there were in the input array. We conclude that the running time of insertion sort is $\Theta(n + I)$.

d. Let $A[1:n]$ be an array representing the input permutation of n elements, and let a and b be two different elements. Suppose that during merge sort invoked for the array A, at some point in a call to MERGE, $L[i] = a$ and $R[j] = b$. If the test in line 13 of MERGE holds, then the elements a and b don't make an inversion. Otherwise, $a > b$, and since the arrays L and R are sorted, b is less than any so far unprocessed element in L (i.e., not yet placed in a target position of A in line 14). These unprocessed elements occupy the subarray $L[i:n_L - 1]$, so b makes exactly $n_L - i$ inversions with them. Once b is copied back to A in line 16, it will share the same merged subarray with the elements in $L[i:n_L - 1]$, so we won't count any inversion twice in subsequent recursive calls to MERGE.

In this way, by modifying merge sort we can determine the number of inversions of an array $A[1:n]$ of n distinct elements in $\Theta(n \lg n)$ worst-case time. The side effect of running the modified procedure is having the input array sorted. The only modification to the original MERGE procedure is adding a new variable for counting inversions in a given recursive call. We will initialize the variable to 0 before line 12, and will be incrementing it by $n_L - i$ in the **else** branch of lines 16–17, before returning it as a result of the modified procedure. We will use a similar counter in the modified MERGE-SORT procedure to sum up the number of inversions from the recursive calls, as well as from the call to the modified MERGE (which we call MERGE-INVERSIONS). The following pseudocode implements the modified MERGE-SORT.

```
COUNT-INVERSIONS(A, p, r)
1  inversions = 0
2  if p < r
3      q = ⌊(p + r)/2⌋
4      inversions = inversions + COUNT-INVERSIONS(A, p, q)
5      inversions = inversions + COUNT-INVERSIONS(A, q + 1, r)
6      inversions = inversions + MERGE-INVERSIONS(A, p, q, r)
7  return inversions
```

The initial call COUNT-INVERSIONS$(A, 1, n)$ returns the number of inversions of $A[1:n]$.

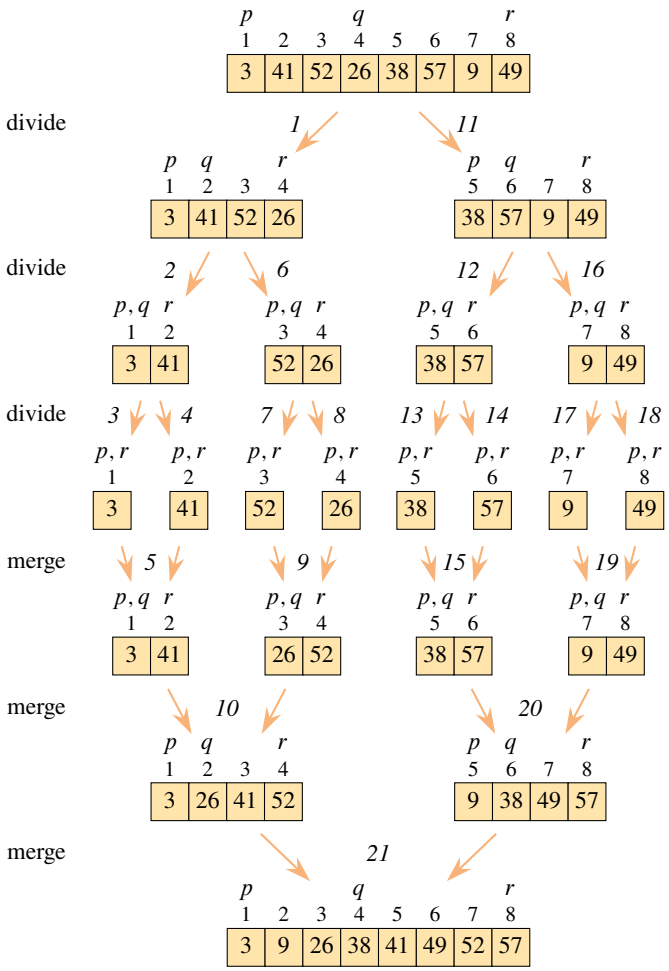


Figure 2.3-1 The operation of merge sort on the array A initially containing the sequence $\langle 3, 41, 52, 26, 38, 57, 9, 49 \rangle$.

3 Characterizing Running Times

3.1 O-notation, Ω-notation, and Θ-notation

3.1-1 Suppose that the $\lfloor n/3 \rfloor$ largest values of the input array A occupy the first $\lfloor n/3 \rfloor$ positions $A[1 : \lfloor n/3 \rfloor]$. When the array is sorted, each of these values must pass through the middle $n - 2\lfloor n/3 \rfloor$ positions $A[\lfloor n/3 \rfloor + 1 : 2\lfloor n/3 \rfloor]$, one position at a time, by at least $n - 2\lfloor n/3 \rfloor$ executions of line 6. So the time taken by INSERTION-SORT in the worst case is at least proportional to

$$\begin{aligned}
\lfloor n/3 \rfloor(n - 2\lfloor n/3 \rfloor) &\geq \lfloor n/3 \rfloor(n - 2n/3) && \text{(by inequality } \lfloor x \rfloor \leq x \text{ (3.2))} \\
&\geq ((n-2)/3)(n/3) && \text{(by inequality (3.8))} \\
&\geq (n/6)(n/3) && \text{(as long as } n \geq 4\text{)} \\
&= n^2/18,
\end{aligned}$$

which is $\Omega(n^2)$.

3.1-2 In SELECTION-SORT the outer loop runs $n - 1$ times, and the inner loop runs at most $n - 1$ times, spending a constant time per iteration. This situation is analogous to that we've seen in INSERTION-SORT, so the upper bound on the running times of both algorithms must the same: $O(n^2)$.

Now observe that during those iterations of the outer loop, in which $j = 1, 2, \ldots, \lfloor n/2 \rfloor$, $A[min]$ is compared with all elements of $A[\lfloor n/2 \rfloor + 1 : n]$ in line 4. So in each of these $\lfloor n/2 \rfloor$ iterations of the outer loop, the inner loop iterates at least $n - \lfloor n/2 \rfloor$ times. The body of the inner loop takes a constant time per iteration, so the total running time of SELECTION-SORT in the worst case is proportional to the total number of iterations

of the inner loop, which is at least

$$\begin{aligned} \lfloor n/2 \rfloor (n - \lfloor n/2 \rfloor) &\geq ((n-1)/2)(n/2) && \text{(by inequalities (3.2) and (3.8))} \\ &\geq (n/4)(n/2) && \text{(as long as } n \geq 2) \\ &= n^2/8, \end{aligned}$$

or $\Omega(n^2)$.

Because we've shown that in all cases SELECTION-SORT runs in $O(n^2)$ time and in $\Omega(n^2)$ time, we can conclude that the running time of SELECTION-SORT is $\Theta(n^2)$.

3.1-3 If we assume that the αn largest values of the input array occupy the first αn positions, those values will need to pass through the $(1 - 2\alpha)n$ middle positions, before the array is sorted. Those middle array positions must exist, so we'll require that $1 - 2\alpha > 0$, or $\alpha < 1/2$. The number of such moves is $\alpha n(1 - 2\alpha)n$. We can find the value of α that maximizes this number for a given n, by analysing the quadratic function $f(\alpha) = \alpha(1 - 2\alpha)$. Its vertex form is $f(\alpha) = -2(\alpha - 1/4) + 1/8$, so the vertex of the parabola is $(1/4, 1/8)$, and so the function is maximized at $\alpha = 1/4$.

3.2 Asymptotic notation: formal definitions

3.2-1 Let n_0 be the smallest value of n such that $f(n) \geq 0$ and $g(n) \geq 0$. For all $n \geq n_0$,

$$0 \leq \frac{f(n) + g(n)}{2} \leq \max\{f(n), g(n)\} \leq f(n) + g(n),$$

so it suffices to let $c_1 = 1/2$ and $c_2 = 1$ in the definition of Θ-notation to obtain the desired result.

3.2-2 Let $T(n)$ be the running time of algorithm A. The statement "$T(n)$ is at least $O(n^2)$" means that $T(n) \geq f(n)$, where $f(n) = O(n^2)$. In particular, we can choose $f(n) = 0$, and the statement will remain true for any $T(n)$. Thus, the statement does not characterize $T(n)$ in any meaningful way.

3.2-3 Let's determine positive constants c and n_0 such that $0 \leq 2^{n+1} \leq c2^n$ for all $n \geq n_0$. Since $2^{n+1} = 2 \cdot 2^n$, we can pick $c = 2$ and $n_0 = 1$. So $2^{n+1} = O(2^n)$.

Now let's assume that c and n_0 are positive constants satisfying $0 \leq 2^{2n} \leq c2^n$ for all $n \geq n_0$. Then $2^{2n} = 2^n \cdot 2^n \leq c2^n$, which implies that $c \geq 2^n$. The last inequality, however, does not hold regardless of a value we would choose for c, because 2^n becomes arbitrarily large as n gets large. Hence, $2^{2n} \neq O(2^n)$.

3.2-4 By the definition of Θ-notation, $f(n) = \Theta(g(n))$ whenever there exist positive constants c_1, c_2, and n_0 such that

$$0 \le c_1 g(n) \le f(n) \le c_2 g(n)$$

for all $n \ge n_0$. This condition can be decomposed into the combination of inequalities $0 \le c_1 g(n) \le f(n)$ and $0 \le f(n) \le c_2 g(n)$, both holding for all $n \ge n_0$. From the former follows $f(n) = \Omega(g(n))$ and from the latter follows $f(n) = O(g(n))$.

For the proof of the opposite direction, suppose that $f(n) = \Omega(g(n))$ and $f(n) = O(g(n))$. The former means that there exist positive constants c_1 and n_1 such that $0 \le c_1 g(n) \le f(n)$ for all $n \ge n_1$, and the latter means that there exist positive constants c_2 and n_2 such that $0 \le f(n) \le c_2 g(n)$ for all $n \ge n_2$. Then, by letting $n_0 = \max\{n_1, n_2\}$ and merging both conditions, we get that

$$0 \le c_1 g(n) \le f(n) \le c_2 g(n)$$

for all $n \ge n_0$. Thus, $f(n) = \Theta(g(n))$.

3.2-5 For the input size n let $T(n)$ be the running time of an algorithm, and let $T_{\text{worst}}(n)$ and $T_{\text{best}}(n)$ be its worst-case running time and its best-case running time, respectively.

Suppose that $T(n) = \Theta(g(n))$. Among all inputs of size n, there is one that is the worst case and one that is the best case for the algorithm in terms of time required to run on these inputs. Since $T(n)$ describes the running time for *all* cases, it also describes the running time for the worst case, and the running time for the best case. Thus, $T_{\text{worst}}(n) = \Theta(g(n))$ and $T_{\text{best}}(n) = \Theta(g(n))$, and by Theorem 3.1 we have that $T_{\text{worst}}(n) = O(g(n))$ and $T_{\text{best}}(n) = \Omega(g(n))$.

Now suppose that $T_{\text{best}}(n) = \Omega(g(n))$ and $T_{\text{worst}}(n) = O(g(n))$. Let c_1 and n_1 be positive constants such that $0 \le c_1 g(n) \le T_{\text{best}}(n)$ for all $n \ge n_1$, and let c_2 and n_2 be positive constants such that $0 \le T_{\text{worst}}(n) \le c_2 g(n)$ for all $n \ge n_2$. Of course, for any input size n, $T_{\text{best}}(n) \le T(n) \le T_{\text{worst}}(n)$. Then for all $n \ge \max\{n_1, n_2\}$,

$$0 \le c_1 g(n) \le T_{\text{best}}(n) \le T(n) \le T_{\text{worst}}(n) \le c_2 g(n),$$

so $T(n) = \Theta(g(n))$.

3.2-6 Suppose the set is nonempty and let $f(n) \in o(g(n)) \cap \omega(g(n))$. Then both relations $f(n) = o(g(n))$ and $f(n) = \omega(g(n))$ hold, which means that for any positive constants c_1 and c_2 there exists a positive constant n_0 such that $c_1 g(n) < f(n) < c_2 g(n)$ for all $n \ge n_0$. This statement is a contradiction, because it is not true that $c_1 < c_2$ for every two numbers c_1 and c_2. Hence, $o(g(n)) \cap \omega(g(n)) = \emptyset$.

3.2-7 $\Omega(g(n,m)) = \{f(n,m) :$ there exist positive constants c, n_0, and m_0 such that $0 \le cg(n,m) \le f(n,m)$ for all $n \ge n_0$ or $m \ge m_0\}$,

$\Theta(g(n,m)) = \{f(n,m) :$ there exist positive constants c_1, c_2, n_0, and m_0 such that $0 \le c_1 g(n,m) \le f(n,m) \le c_2 g(n,m)$ for all $n \ge n_0$ or $m \ge m_0\}$.

3.3 Standard notations and common functions

3.3-1 Let $n \le m$. Then, $f(n) \le f(m)$ and $g(n) \le g(m)$. If we combine these inequalities by adding one to the other, we obtain

$$f(n) + g(n) \le f(m) + g(m),$$

which means that the function $f(n) + g(n)$ is monotonically increasing. By plugging the values of $g(n)$ into the function $f(n)$, we have that $g(n) \le g(m)$ implies $f(g(n)) \le f(g(m))$, and so $f(g(n))$ is monotonically increasing. If we additionally assume that $f(n)$ and $g(n)$ are nonnegative, we can combine the inequalities $f(n) \le f(m)$ and $g(n) \le g(m)$ by multiplying one with the other:

$$f(n) \cdot g(n) \le f(m) \cdot g(m).$$

Thus, $f(n) \cdot g(n)$ is monotonically increasing.

3.3-2
$$\begin{aligned}
\lfloor \alpha n \rfloor + \lceil (1-\alpha)n \rceil &= \lfloor \alpha n \rfloor + \lceil n + (-\alpha n) \rceil \\
&= \lfloor \alpha n \rfloor + n + \lceil -\alpha n \rceil && \text{(by equation (3.10))} \\
&= \lfloor \alpha n \rfloor + n - \lfloor \alpha n \rfloor && \text{(by equation (3.3))} \\
&= n
\end{aligned}$$

3.3-3 Let $f(n) = o(n)$. Then, in particular, there exists a positive constant n_0 such that $0 \le f(n) < n/2$ for all $n \ge n_0$. If $k \ge 0$, the function n^k is monotonically increasing, so we have

$$(n + f(n))^k \ge n^k$$

and

$$\begin{aligned}
(n + f(n))^k &\le (n + n/2)^k \\
&= (3/2)^k n^k.
\end{aligned}$$

We can satisfy the condition in the definition of Θ-notation for all $n \geq n_0$, by letting $c_1 = 1$ and $c_2 = (3/2)^k$. In case where $k < 0$, the function n^k is monotonically decreasing, and the above inequalities need to be reversed:

$$(3/2)^k n^k \leq (n + f(n)) \leq n^k.$$

Then it suffices to plug $c_1 = (3/2)^k$ and $c_2 = 1$ to make the condition true. Hence, $(n + o(n))^k = \Theta(n^k)$.

Similarly, we can show that $(n - o(n))^k = \Theta(n^k)$. Suppose that $k \geq 0$. For any $f(n) = o(n)$ satisfying $0 \leq f(n) < n/2$ for all $n \geq n_0$, where n_0 is a positive constant, we have

$$\begin{aligned}(n - f(n))^k &\geq (n - n/2)^k \\ &= (1/2)^k n^k\end{aligned}$$

and

$$(n - f(n))^k \leq n^k.$$

In the definition of Θ-notation we let $c_1 = (1/2)^k$ and $c_2 = 1$, so that the condition is met for all $n \geq n_0$. The case where $k < 0$ is symmetric.

From inequalities (3.2), if $k \geq 0$, then

$$(n-1)^k < \lceil n \rceil^k < (n+1)^k,$$
$$(n-1)^k < \lfloor n \rfloor^k < (n+1)^k,$$

and symmetrically, if $k < 0$. Then, by the fact shown above, we have $\lceil n \rceil^k = \Theta(n^k)$ and $\lfloor n \rfloor^k = \Theta(n^k)$.

3.3-4 ***a.*** By applying equation (3.17) twice, we get

$$\begin{aligned}a^{\log_b c} &= (b^{\log_b a})^{\log_b c} \\ &= (b^{\log_b c})^{\log_b a} \\ &= c^{\log_b a}.\end{aligned}$$

b. We'll prove that

$$\lim_{n \to \infty} \frac{n!}{2^n} = \infty.$$

Note that in the product

$$\frac{n!}{2^n} = \left(\frac{1}{2}\right)\left(\frac{2}{2}\right)\cdots\left(\frac{n}{2}\right)$$

all the factors on the right-hand side are positive and, as n increases, the final factors can grow arbitrarily large, so the product diverges to ∞.

Similarly, we'll prove that

$$\lim_{n\to\infty} \frac{n!}{n^n} = 0.$$

We have

$$\frac{n!}{n^n} = \left(\frac{1}{n}\right)\left(\frac{2}{n}\right)\cdots\left(\frac{n}{n}\right).$$

Each factor on the right-hand side is positive and does not exceed 1. For n large enough the initial factors can be arbitrarily close to 0, so the product approaches its limit of 0.

For the proof of equation (3.28), observe that since $n! \le n^n$, the upper bound on $\lg(n!)$ is $\lg n^n = n \lg n$. We'll find an asymptotic lower bound by using Stirling's approximation. Let $f(n) = 1 + \Theta(1/n) = \Theta(1)$ be the last factor on the right-hand side of formula (3.25). Because the function $\sqrt{2\pi n}$ is strictly increasing, there exists a positive constant n_0 such that $\sqrt{2\pi n}\, f(n) \ge 1$ for all $n \ge n_0$. Also, we'll use the fact that $e < 3$, so $e < \sqrt{n}$ when $n \ge 9$. For all $n \ge \max\{n_0, 9\}$ we have

$$\begin{aligned}
\lg(n!) &= \lg\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n f(n)\right) \\
&\ge \lg\left(\frac{n}{e}\right)^n \\
&> \lg\left(\sqrt{n}\right)^n \\
&= \frac{n \lg n}{2}.
\end{aligned}$$

Combining both upper and lower bounds, we obtain $\lg(n!) = \Theta(n \lg n)$.

c. Let $f(n) = \Theta(n)$. By the definition of Θ-notation, there exist positive constants c_1, c_2, and n_0 such that

$$c_1 n \le f(n) \le c_2 n$$

for all $n \ge n_0$. Taking logarithms of each side of the above inequalities yields

$$\begin{aligned}
\lg f(n) &\ge \lg(c_1 n) \\
&= \lg c_1 + \lg n \\
&\ge \lg\left(\frac{1}{\sqrt{n}}\right) + \lg n && \text{(as long as } n \ge 1/c_1^2\text{)} \\
&= \lg\sqrt{n} \\
&= \frac{\lg n}{2}
\end{aligned}$$

and

$$\begin{aligned}\lg f(n) &\le \lg(c_2 n)\\ &= \lg c_2 + \lg n\\ &\le 2\lg n && \text{(as long as } n \ge c_2).\end{aligned}$$

So, $\lg f(n) = \Theta(\lg n)$.

3.3-5 ★ Let a function $f(n)$ be polynomially bounded. Then $f(n) = O(n^k)$ for some constant k. As we show in Problem 3-4(c), this fact implies that $\lg f(n) = O(\lg n^k) = O(\lg n)$. Now suppose that $\lg f(n) = O(\lg n)$. Then there exist positive constants c and n_0 such that $0 \le \lg f(n) \le c \lg n$ for all $n \ge n_0$. Then, we have

$$\begin{aligned}0 &\le f(n)\\ &= 2^{\lg f(n)}\\ &\le 2^{c \lg n}\\ &= (2^{\lg n})^c\\ &= n^c\end{aligned}$$

for all $n \ge n_0$, so that $f(n)$ is polynomially bounded.

In the following proofs we use equation (3.28) proved in Exercise 3.3-4 and equation $\lceil n \rceil = \Theta(n)$ proved in Exercise 3.3-3, after substituting different functions for n in those equations. We also rely on the observations and proofs from the solution to Problem 3-3.

We have

$$\begin{aligned}\lg(\lceil \lg n \rceil !) &= \Theta(\lceil \lg n \rceil \lg \lceil \lg n \rceil)\\ &= \Theta(\lg n \lg \lg n)\\ &= \omega(\lg n).\end{aligned}$$

We've shown that $\lg(\lceil \lg n \rceil !) \ne O(\lg n)$, which means that the function $\lceil \lg n \rceil !$ is not polynomially bounded.

On the other hand,

$$\begin{aligned}\lg(\lceil \lg \lg n \rceil !) &= \Theta(\lceil \lg \lg n \rceil \lg \lceil \lg \lg n \rceil)\\ &= \Theta(\lg \lg n \lg \lg \lg n)\\ &= o((\lg \lg n)^2)\\ &= o(\lg n),\end{aligned}$$

where the last step results from equation (3.24) for $a = 1$ and $b = 2$ and after substituting $\lg n$ for n. The obtained result implies in particular that $\lg(\lceil \lg \lg n \rceil !) =$

$O(\lg n)$, so the function $\lceil \lg \lg n \rceil !$ is polynomially bounded.

3.3-6 ★ As stated in the text on page 67, any positive polynomial function grows faster than any polylogarithmic function, in particular $m - 1 = \omega(\lg m)$. So for any positive constant c there is a positive constant m_0 such that $0 \le c \lg m < m - 1$ for all $m \ge m_0$. Let

$$n_0 = \underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{m_0}.$$

Notice that $\lg^* n_0 = m_0$ and since the function $\lg^* n$ is monotonically increasing, for $n \ge n_0$ it takes all integer values greater than or equal to m_0. We can therefore substitute $\lg^* n$ for m in the previous inequalities. More precisely, it is true that for any positive constant c there is a positive constant n_0 such that

$$\begin{aligned} 0 &\le c \lg(\lg^* n) \\ &< \lg^* n - 1 \\ &= \lg^*(\lg n) \end{aligned}$$

for all $n \ge n_0$. Therefore, $\lg^*(\lg n) = \omega(\lg(\lg^* n))$.

3.3-7 *ϕ and $\widehat{\phi}$ should be given by equations (3.32) and (3.33), respectively. Corrected in fourth printing.*

We have

$$\begin{aligned} \phi^2 &= \left(\frac{1+\sqrt{5}}{2}\right)^2 \\ &= \frac{1+2\sqrt{5}+5}{4} \\ &= \frac{3+\sqrt{5}}{2} \\ &= \frac{1+\sqrt{5}}{2} + 1 \\ &= \phi + 1 \end{aligned}$$

and

$$\begin{aligned}\widehat{\phi}^2 &= \left(\frac{1-\sqrt{5}}{2}\right)^2 \\ &= \frac{1-2\sqrt{5}+5}{4} \\ &= \frac{3-\sqrt{5}}{2} \\ &= \frac{1-\sqrt{5}}{2}+1 \\ &= \widehat{\phi}+1,\end{aligned}$$

so both ϕ and $\widehat{\phi}$ are the roots of the equation $x^2 = x + 1$, and therefore are the golden ratio and its conjugate, respectively.

3.3-8 *ϕ and $\widehat{\phi}$ should be given by equations (3.32) and (3.33), respectively. Corrected in fourth printing.*

We have

$$\begin{aligned}\frac{\phi^0-\widehat{\phi}^0}{\sqrt{5}} &= 0 \\ &= F_0\end{aligned}$$

and

$$\begin{aligned}\frac{\phi^1-\widehat{\phi}^1}{\sqrt{5}} &= \frac{1+\sqrt{5}-\left(1-\sqrt{5}\right)}{2\sqrt{5}} \\ &= 1 \\ &= F_1,\end{aligned}$$

so the base case holds. Now suppose that $i \geq 2$. The inductive hypothesis is that

$$F_{i-2} = \frac{\phi^{i-2}-\widehat{\phi}^{i-2}}{\sqrt{5}}$$

and

$$F_{i-1} = \frac{\phi^{i-1}-\widehat{\phi}^{i-1}}{\sqrt{5}}.$$

Then, we have

$$\begin{aligned}
F_i &= F_{i-2} + F_{i-1} \\
&= \frac{\phi^{i-2} - \widehat{\phi}^{i-2}}{\sqrt{5}} + \frac{\phi^{i-1} - \widehat{\phi}^{i-1}}{\sqrt{5}} \\
&= \frac{\phi^{i-2}(1+\phi) - \widehat{\phi}^{i-2}\left(1+\widehat{\phi}\right)}{\sqrt{5}} \\
&= \frac{\phi^{i} - \widehat{\phi}^{i}}{\sqrt{5}} && \text{(by Exercise 3.3-7),}
\end{aligned}$$

so the formula for F_i holds for all $i \geq 0$.

3.3-9 Suppose that $k \lg k = \Theta(n)$. By the definition of Θ-notation for functions with two parameters (see Exercise 3.2-7), there exist positive constants c_1, c_2, k_0, and n_0 such that

$$c_1 n \leq k \lg k \leq c_2 n$$

for all $k \geq k_0$ or $n \geq n_0$. By placing the additional restriction of $k \geq 2$ and taking logarithms on all parts of the above compound inequality, we get

$$\lg c_1 + \lg n \leq \lg k + \lg \lg k \leq \lg c_2 + \lg n.$$

Then, as long as $n \geq c_2$, $\lg c_2 \leq \lg n$, and so

$$\begin{aligned}
\lg k &\leq \lg c_2 + \lg n - \lg \lg k \\
&\leq \lg n + \lg n \\
&= 2 \lg n,
\end{aligned}$$

or $1 \geq \lg k / 2 \lg n$. Therefore,

$$\begin{aligned}
k &\geq \frac{k \lg k}{2 \lg n} \\
&\geq \frac{c_1}{2} \cdot \frac{n}{\lg n}
\end{aligned}$$

for all $k \geq \max\{k_0, 2\}$ or $n \geq \max\{n_0, c_2\}$. Hence, $k = \Omega(n / \lg n)$.

Now, let's also assume that $k \geq 16$, which implies $\lg k \leq \sqrt{k}$, or $\lg \lg k \leq (\lg k)/2$. As long as $n \geq 1/c_1^2$, $\lg c_1 \geq -(\lg n)/2$. Then,

$$\begin{aligned}
\lg k &\geq \lg c_1 + \lg n - \lg \lg k \\
&\geq -(\lg n)/2 + \lg n - (\lg k)/2 \\
&= (\lg n)/2 - (\lg k)/2
\end{aligned}$$

so $\lg k \geq (\lg n)/3$ or, equivalently, $1/3 \leq \lg k / \lg n$. Then,

$$\begin{aligned} k &\leq \frac{3k \lg k}{\lg n} \\ &\leq 3c_2 \cdot \frac{n}{\lg n} \end{aligned}$$

for all $k \geq \max\{k_0, 16\}$ or $n \geq \max\left\{n_0, 1/c_1^2\right\}$. Hence, $k = O(n/\lg n)$, and by Theorem 3.1, we obtain $k = \Theta(n/\lg n)$.

Problems

3-1 *Asymptotic behavior of polynomials*

In this problem we rely on the fact noted in the text on page 65 — for an asymptotically positive polynomial $p(n)$ of degree d, $p(n) = \Theta(n^d)$. The proofs refer to the positive constants c_1, c_2, and n_0 such that $0 \leq c_1 n^d \leq p(n) \leq c_2 n^d$ for all $n \geq n_0$.

a. $k \geq d$ implies that $n^k \geq n^d$ for all $n \geq 1$, so

$$0 \leq p(n) \leq c_2 n^d \leq c_2 n^k$$

for all $n \geq \max\{n_0, 1\}$. Hence, $p(n) = O(n^k)$.

b. Symmetrically, $k \leq d$ implies that $n^k \leq n^d$ for all $n \geq 1$, so

$$0 \leq c_1 n^k \leq c_1 n^d \leq p(n)$$

for all $n \geq \max\{n_0, 1\}$. Hence, $p(n) = \Omega(n^k)$.

c. For $k = d$ the equality $p(n) = \Theta(n^d) = \Theta(n^k)$ holds trivially.

d. Let b_2 be a positive constant. The inequality $c_2 n^d < b_2 n^k$ cas be rewritten as $c_2/b_2 < n^{k-d}$. If we let $m = (c_2/b_2)^{1/(k-d)}$ we have that, since $k > d$, the inequality holds for all $n > m$. Thus, for any $b_2 > 0$,

$$0 \leq p(n) \leq c_2 n^d < b_2 n^k$$

for all $n \geq \max\{n_0, m + 1\}$. Hence, $p(n) = o(n^k)$.

e. The proof is symmetric to the one from part (d). Let b_1 be a positive constant and let $m = (b_1/c_1)^{1/(d-k)}$. Then, since $k < d$, the inequality $b_1 n^k < c_1 n^d$ holds for all $n > m$. Thus, for any $b_1 > 0$,

$$0 \leq b_1 n^k < c_1 n^d \leq p(n)$$

for all $n \geq \max\{n_0, m + 1\}$. Hence, $p(n) = \omega(n^k)$.

3-2 Relative asymptotic growths

a. By equation (3.24), $\lg^k n = o(n^\epsilon)$, which also implies $\lg^k n = O(n^\epsilon)$.

b. By equation (3.13), $n^k = o(c^n)$, which also implies $n^k = O(c^n)$.

c. The functions $\sqrt{n}$ and $n^{\sin n}$ cannot be compared using asymptotic notation, because the value of the exponent in $n^{\sin n}$ takes all values between 0 and 1, while $\sqrt{n} = n^{1/2}$.

d. We have

$$\begin{aligned}\lim_{n\to\infty} \frac{2^n}{2^{n/2}} &= \lim_{n\to\infty} 2^{n/2} \\ &= \infty,\end{aligned}$$

so $2^n = \omega(2^{n/2})$, which also implies $2^n = \Omega(2^{n/2})$.

e. By equation (3.21), the functions $n^{\lg c}$ and $c^{\lg n}$ are equivalent.

f. By equation (3.28), $\lg(n!) = \Theta(n \lg n)$. Because $\lg(n^n) = n \lg n = \Theta(n \lg n)$, we have $\lg(n!) = \Theta(\lg(n^n))$.

The table in Figure 3-2 collects the obtained results.

3-3 Ordering by asymptotic growth rates

a. In our justifications we will make use of the following lemmas:

Lemma 1
If a function $f(n)$ is asymptotically positive and if $g(n) = \omega(h(n))$, then $f(n)g(n) = \omega(f(n)h(n))$.

A	B	O	o	Ω	ω	Θ
$\lg^k n$	n^ϵ	yes	yes	no	no	no
n^k	c^n	yes	yes	no	no	no
$\sqrt{n}$	$n^{\sin n}$	no	no	no	no	no
2^n	$2^{n/2}$	no	no	yes	yes	no
$n^{\lg c}$	$c^{\lg n}$	yes	no	yes	no	yes
$\lg(n!)$	$\lg(n^n)$	yes	no	yes	no	yes

Figure 3-2 Comparison of asymptotic growths of pairs of functions.

Proof Pick n_1, so that $f(n) > 0$ for all $n \geq n_1$. The relation $g(n) = \omega(h(n))$ implies that for any constant $c > 0$ there exists a constant $n_2 > 0$ such that

$$0 \leq ch(n) < g(n)$$

for all $n \geq n_2$. If we let $n \geq \max\{n_1, n_2\}$ and multiply all parts of the above compound inequality by $f(n)$, we obtain the result. ■

Lemma 2
If $f(n) = \omega(g(n))$ and $\lim_{n\to\infty} g(n) = \infty$, then $2^{f(n)} = \omega\left(2^{g(n)}\right)$.

Proof By the definition of ω-notation, for any constant $c > 0$ there is a constant $n_0 > 0$ such that $f(n) > cg(n)$ for all $n \geq n_0$. For fixed c, let d be a number satisfying the inequality $cg(n) \geq d + g(n)$ for all $n \geq n_0$. Then $d \leq (c-1)g(n)$, and because $g(n)$ grows arbitrarily large as n grows, d can take any real value, and so 2^d can take any positive value. Thus,

$$\begin{aligned} 2^{f(n)} &> 2^{cg(n)} \\ &\geq 2^{d+g(n)} \\ &= 2^d \cdot 2^{g(n)} \\ &> 0, \end{aligned}$$

so $2^{f(n)} = \omega\left(2^{g(n)}\right)$. ■

Lemma 3
For $f(n) = \Theta(g(n))$, $h(n) = \omega(f(n))$ if and only if $h(n) = \omega(g(n))$.

Proof Let c and n_0 be positive constants such that for all $n \geq n_0$,

$$0 \leq cg(n) \leq f(n).$$

If $h(n) = \omega(f(n))$, then for any constant $a > 0$ there exists a constant $n_a > 0$ such that $0 \leq af(n) < h(n)$ for all $n \geq n_a$. For any a and all $n \geq \max\{n_0, n_a\}$,

$$\begin{aligned} h(n) &> af(n) \\ &\geq acg(n) \\ &\geq 0, \end{aligned}$$

or $h(n) = \omega(g(n))$, because the factor ac can take any positive value.

By symmetry of Θ-notation, $g(n) = \Theta(f(n))$, so the proof in the backward direction is symmetric, with $f(n)$ and $g(n)$ swapped. ■

The following justifications show that either $f(n) = \omega(g(n))$ — which implies that $f(n) = \Omega(g(n))$ — or $f(n) = \Theta(g(n))$.

$2^{2^{n+1}} = \omega\left(2^{2^n}\right)$
By Lemma 1 for $f(n) = g(n) = 2^{2^n}$ and $h(n) = 1$.

$2^{2^n} = \omega((n+1)!)$
By equation (3.13), $2^n = \omega(n^2)$, and by Lemma 1, $n^2 = \omega(n \lg n)$, because $n = \omega(\lg n)$. Then, by transitivity of ω-notation, $2^n = \omega(n \lg n)$. By equation (3.28) and the fact that $0 < \lg(n+1) < \lg(n!)$ for $n \geq 3$,

$$\begin{aligned} \lg((n+1)!) &= \lg(n+1) + \lg(n!) \\ &= \Theta(n \lg n), \end{aligned}$$

so thanks to Lemma 3 we have $2^n = \omega(\lg(n+1)!)$ and the result follows from Lemma 2.

$(n+1)! = \omega(n!)$
Immediate from Lemma 1 for $f(n) = n!$, $g(n) = n + 1$, and $h(n) = 1$.

$n! = \omega(e^n)$
By equation (3.28) and Lemma 1, $\lg(n!) = \Theta(n \lg n) = \omega(n)$, while $\lg e^n = \Theta(n)$. Thanks to Lemma 3 we have $\lg(n!) = \omega(\lg e^n)$ and the relation follows from Lemma 2.

$e^n = \omega(n \cdot 2^n)$
By equation (3.13), $(e/2)^n = \omega(n)$. Then, the relation follows from Lemma 1 for $f(n) = 2^n$, $g(n) = (e/2)^n$, and $h(n) = n$.

$n \cdot 2^n = \omega(2^n)$
Immediate from Lemma 1 for $f(n) = 2^n$, $g(n) = n$, and $h(n) = 1$.

$2^n = \omega((3/2)^n)$
Immediate from Lemma 1 for $f(n) = (3/2)^n$, $g(n) = (4/3)^n$, and $h(n) = 1$.

$(3/2)^n = \omega(n^{\lg\lg n})$

$\lg(3/2)^n = n\lg(3/2)$ and

$$\begin{aligned}\lg(n^{\lg\lg n}) &= \lg n \lg\lg n \\ &< \lg n \lg n \\ &= \lg^2 n .\end{aligned}$$

By equality (3.24), $n\lg(3/2) = \omega(\lg^2 n)$, and so $\lg(3/2)^n = \omega(\lg(n^{\lg\lg n}))$. Then, the result follows from Lemma 2.

$n^{\lg\lg n} = \Theta((\lg n)^{\lg n})$

Immediate from equation (3.21).

$(\lg n)^{\lg n} = \omega((\lg n)!)$

Using Stirling's approximation, we get

$$\begin{aligned}(\lg n)! &= \Theta\left(\sqrt{2\pi\lg n}\left(\frac{\lg n}{e}\right)^{\lg n}\right) \\ &= \Theta\left((\lg n)^{\lg n}\frac{\sqrt{\lg n}}{n^{\lg e}}\right).\end{aligned}$$

Let

$$f(n) = (\lg n)^{\lg n}\frac{\sqrt{\lg n}}{n^{\lg e}}$$

and let

$$g(n) = \frac{n^{\lg e}}{\sqrt{\lg n}}.$$

Then, $g(n) = \omega(1)$, so by Lemma 1, $f(n)g(n) = (\lg n)^{\lg n} = \omega(f(n))$. Finally, $(\lg n)^{\lg n} = \omega((\lg n)!)$ follows from Lemma 3.

$(\lg n)! = \omega(n^3)$

By equation (3.28) and Lemma 1, we get $\lg((\lg n)!) = \Theta(\lg n \lg\lg n) = \omega(\lg n)$ and $\lg n^3 = 3\lg n$, so $\lg((\lg n)!) = \omega(\lg n^3)$ and now it suffices to apply Lemma 2.

$n^3 = \omega(n^2)$

Immediate from Problem 3-1(e).

$n^2 = \Theta(4^{\lg n})$

Immediate from equation (3.21).

$4^{\lg n} = \omega(n\lg n)$

By equation (3.21), $4^{\lg n} = n^2$, so we apply Lemma 1 for $f(n) = g(n) = n$ and $h(n) = \lg n$.

$n \lg n = \Theta(\lg(n!))$

Immediate from equation (3.28).

$\lg(n!) = \omega(n)$

By equation (3.28) and Lemma 1, $\lg(n!) = \Theta(n \lg n) = \omega(n)$.

$n = \Theta(2^{\lg n})$

Immediate from equation (3.21).

$2^{\lg n} = \omega\left(\left(\sqrt{2}\right)^{\lg n}\right)$

By equation (3.21), $2^{\lg n} = n$, as well as $\left(\sqrt{2}\right)^{\lg n} = n^{\lg\sqrt{2}} = \sqrt{n}$. Thus, the relation follows from Problem 3-1(e).

$\left(\sqrt{2}\right)^{\lg n} = \omega\left(2^{\sqrt{2\lg n}}\right)$

It holds that

$$\begin{aligned}\lg\left(\sqrt{2}\right)^{\lg n} &= \lg n \lg\sqrt{2} \\ &= \frac{\lg n}{2} \\ &= \sqrt{2\lg n} \cdot \frac{\sqrt{\lg n}}{2\sqrt{2}} \\ &= \sqrt{2\lg n} \cdot \omega(1),\end{aligned}$$

so Lemma 1 implies $\lg\left(\sqrt{2}\right)^{\lg n} = \omega\left(\lg 2^{\sqrt{2\lg n}}\right)$. Now it suffices to use Lemma 2.

$2^{\sqrt{2\lg n}} = \omega(\lg^2 n)$

Consider $\lg\lg 2^{\sqrt{2\lg n}} = \Theta(\lg\lg n)$ and $\lg\lg\lg^2 n = \Theta(\lg\lg\lg n)$. We can show that $\lg\lg n = \omega(\lg\lg\lg n)$ by substituting $\lg\lg n$ for n in equation (3.24). Then we have $\lg\lg 2^{\sqrt{2\lg n}} = \omega(\lg\lg\lg^2 n)$, and we apply Lemma 2 twice.

$\lg^2 n = \omega(\ln n)$

By Lemma 1, $\lg^2 n = \omega(\lg n)$. Also, $\ln n = \Theta(\lg n)$, so the relation follows from Lemma 3.

$\ln n = \omega\left(\sqrt{\lg n}\right)$

We have

$$\begin{aligned}\ln n &= \frac{\lg n}{\lg e} \\ &= \sqrt{\lg n} \cdot \frac{\sqrt{\lg n}}{\lg e} \\ &= \sqrt{\lg n} \cdot \omega(1),\end{aligned}$$

so the relation follows from Lemma 1.

$\sqrt{\lg n} = \omega(\ln \ln n)$

We have $\lg \sqrt{\lg n} = \Theta(\lg \lg n)$ and $\lg \ln \ln n = \Theta(\lg \lg \lg n)$. By equation (3.24), after substituting $\lg \lg n$ for n, $\lg \lg n = \omega(\lg \lg \lg n)$, so $\lg \sqrt{\lg n} = \omega(\lg \ln \ln n)$, and the relation follows from Lemma 2.

$\ln \ln n = \omega\left(2^{\lg^* n}\right)$

$\lg \ln \ln n = \Theta(\lg \lg \lg n) = \omega(\lg^* n)$, so we apply Lemma 2.

$2^{\lg^* n} = \omega(\lg^* n)$

$\lg 2^{\lg^* n} = \lg^* n$ and it holds that $\lg^* n = \omega(\lg(\lg^* n))$, which we prove in the following justifications. Then the relation follows from Lemma 2.

$\lg^* n = \Theta(\lg^*(\lg n))$

$\lg^*(\lg n) = \lg^* n - 1$ holds for all $n \geq 2$.

$\lg^*(\lg n) = \omega(\lg(\lg^* n))$

Immediate from Exercise 3.3-6.

$\lg(\lg^* n) = \omega(n^{1/\lg n})$

By equation (3.19), $1/\lg n = \log_n 2$, and by (3.21),

$$\begin{aligned} n^{1/\lg n} &= n^{\log_n 2} \\ &= 2^{\log_n n} \\ &= 2 \\ &= \Theta(1). \end{aligned}$$

On the other hand, $\lg(\lg^* n) = \omega(1)$, and so the relation follows from Lemma 3.

$n^{1/\lg n} = \Theta(1)$

Immediate from the above justification.

Below is the list of all examined functions $g_1, g_2, \ldots, g_{30}$ ordered according to relations between them in terms of Ω-notation. Each line represents a different equivalence class of Θ-notation.

$g_1(n) = 2^{2^{n+1}}$

$g_2(n) = 2^{2^n}$

$g_3(n) = (n+1)!$

$g_4(n) = n!$

$g_5(n) = e^n$

$g_6(n) = n \cdot 2^n$

$g_7(n) = 2^n$

$g_8(n) = (3/2)^n$

$g_9(n) = n^{\lg\lg n},\ g_{10}(n) = (\lg n)^{\lg n}$

$g_{11}(n) = (\lg n)!$

$g_{12}(n) = n^3$

$g_{13}(n) = n^2,\ g_{14}(n) = 4^{\lg n}$

$g_{15}(n) = n \lg n$

$g_{16}(n) = \lg(n!)$

$g_{17}(n) = n,\ g_{18}(n) = 2^{\lg n}$

$g_{19}(n) = \left(\sqrt{2}\right)^{\lg n}$

$g_{20}(n) = 2^{\sqrt{2\lg n}}$

$g_{21}(n) = \lg^2 n$

$g_{22}(n) = \ln n$

$g_{23}(n) = \sqrt{\lg n}$

$g_{24}(n) = \ln\ln n$

$g_{25}(n) = 2^{\lg^* n}$

$g_{26}(n) = \lg^* n,\ g_{27}(n) = \lg^*(\lg n)$

$g_{28}(n) = \lg(\lg^* n)$

$g_{29}(n) = n^{1/\lg n},\ g_{30}(n) = 1$

b. Here is an example of a function that has the desired property:

$$f(n) = \left(2^{2^{n+1}}\right)^{n \bmod 2}.$$

Observe that $f(n) = g_1(n)$ when n is odd, and $f(n) = g_{30}(n)$ when n is even.

3-4 *Asymptotic notation properties*

a. False. Let $f(n) = n$ and $g(n) = n^2$. Then $f(n) = O(g(n))$ but $g(n) \neq O(f(n))$.

b. False. Let $f(n) = n$ and $g(n) = n^2$. For all $n \geq 1$, $\min\{f(n), g(n)\} = f(n)$, so $f(n) + g(n) = O(g(n)) \neq O(f(n))$.

c. True. Suppose that $f(n) = O(g(n))$. Let c and n_0 be positive constants such that $1 \leq f(n) \leq cg(n)$ and $\lg g(n) \geq 1$ for all $n \geq n_0$. Then,

$$\begin{aligned}\lg f(n) &\leq \lg c + \lg g(n)\\ &\leq \lg c \cdot \lg g(n) + \lg g(n)\\ &= (\lg c + 1)\lg g(n)\\ &= O(\lg g(n)).\end{aligned}$$

d. False. Let $f(n) = 2n$ and $g(n) = n$. It holds $f(n) = O(g(n))$, but $2^{f(n)} \neq O\left(2^{g(n)}\right)$ (see Exercise 3.2-3).

e. False. If $f(n) = 1/n$, then $f^2(n) = 1/n^2$. Since there doesn't exist any positive constant c such that $1/n \leq c/n^2$ for arbitrarily large n, then $f(n) \neq O(f^2(n))$.

f. True. Suppose that $f(n) = O(g(n))$. Let c and n_0 be positive constants such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$. Dividing all parts of the inequality by c yields $0 \leq f(n)/c \leq g(n)$, and since $1/c > 0$, then $g(n) = \Omega(f(n))$.

g. False. Let $f(n) = 2^n$, so $f(n/2) = 2^{n/2} = \sqrt{2^n}$. Suppose that $f(n) = O(f(n/2))$. Then for a positive constant c and for sufficiently large n, it holds $2^n \leq c\sqrt{2^n}$. But then $c \geq \sqrt{2^n}$ and c cannot be a constant. Therefore, $f(n) \neq O(f(n/2))$, which implies $f(n) \neq \Theta(f(n/2))$.

h. True. Let $h(n) = o(f(n))$. Then, for any positive constant c there exists a positive constant n_0 such that $0 \leq h(n) < cf(n)$ for all $n \geq n_0$. This implies that

$$\begin{aligned}f(n) &\leq f(n) + o(f(n))\\ &= f(n) + h(n)\\ &< (c+1)f(n)\\ &< 2f(n),\end{aligned}$$

so $f(n) + o(f(n)) = \Theta(f(n))$.

3-5 *Manipulating asymptotic notation*

a. Let $p(n) = \Theta(f(n))$ be the function on the left-hand side of the equation under the outer Θ, and let c_1, c_2, and n_p be positive constants such that

$$0 \le c_1 f(n) \le p(n) \le c_2 f(n)$$

for all $n \ge n_p$. Also, let $q(n) = \Theta(p(n))$ and let d_1, d_2, and n_q be positive constants such that

$$0 \le d_1 p(n) \le q(n) \le d_2 q(n)$$

for all $n \ge n_q$. Then, for all $n \ge \max\{n_p, n_q\}$,

$$\begin{aligned} 0 &\le c_1 d_1 f(n) \\ &\le d_1 p(n) \\ &\le q(n) \\ &\le d_2 p(n) \\ &\le c_2 d_2 f(n), \end{aligned}$$

which implies that $q(n) = \Theta(f(n))$.

b. Let $p(n) = \Theta(f(n))$ and $q(n) = O(f(n))$ be the two functions on the left-hand side of the equation. There exist positive constants c_1, c_2, d, n_p, and n_q such that

$$0 \le c_1 f(n) \le p(n) \le c_2 f(n)$$

for all $n \ge n_p$, and

$$0 \le q(n) \le df(n)$$

for all $n \ge n_q$. Then, for all $n \ge \max\{n_p, n_q\}$,

$$\begin{aligned} 0 &\le c_1 f(n) \\ &\le p(n) \\ &\le p(n) + q(n) \\ &\le c_2 f(n) + df(n) \\ &= (c_2 + d) f(n), \end{aligned}$$

which implies that $p(n) + q(n) = \Theta(f(n))$.

c. Let $p(n) = \Theta(f(n))$ and $q(n) = \Theta(g(n))$ be the two functions on the left-hand side of the equation. There exist positive constants c_1, c_2, d_1, d_2, n_p, and n_q such that

$$0 \le c_1 f(n) \le p(n) \le c_2 f(n)$$

for all $n \geq n_p$, and

$$0 \leq d_1 g(n) \leq q(n) \leq d_2 g(n)$$

for all $n \geq n_q$. Then, for all $n \geq \max\{n_p, n_q\}$,

$$\begin{aligned}
0 &\leq \min\{c_1, d_1\}\,(f(n) + g(n)) \\
&\leq c_1 f(n) + d_1 g(n) \\
&\leq p(n) + q(n) \\
&\leq c_2 f(n) + d_2 g(n) \\
&\leq \max\{c_2, d_2\}\,(f(n) + g(n)),
\end{aligned}$$

which implies that $p(n) + q(n) = \Theta(f(n) + g(n))$.

d. For the same functions and constants as in the previous part, it is true that for all $n \geq \max\{n_p, n_q\}$,

$$\begin{aligned}
0 &\leq \min\{c_1, d_1\}^2\,(f(n) \cdot g(n)) \\
&\leq c_1 f(n) \cdot d_1 g(n) \\
&\leq p(n) \cdot q(n) \\
&\leq c_2 f(n) \cdot d_2 g(n) \\
&\leq \max\{c_2, d_2\}^2\,(f(n) \cdot g(n)),
\end{aligned}$$

which implies that $p(n) \cdot q(n) = \Theta(f(n) \cdot g(n))$.

e. *Instead of a_1, $b_1 > 0$ there should be a_1, $a_2 > 0$. Corrected in fourth printing.*

We show the asymptotic upper bound and the asymptotic lower bound separately. For all $n \geq a_2$, $\lg a_2 \leq \lg n$, so

$$\begin{aligned}
(a_1 n)^{k_1} \lg^{k_2}(a_2 n) &= a_1^{k_1} n^{k_1} (\lg a_2 + \lg n)^{k_2} \\
&\leq a_1^{k_1} n^{k_1} (2 \lg n)^{k_2} \\
&= a_1^{k_1} 2^{k_2} \cdot n^{k_1} \lg^{k_2} n \\
&= O(n^{k_1} \lg^{k_2} n),
\end{aligned}$$

because the factor $a_1^{k_1} 2^{k_2}$ certainly is a positive constant.

Now, if we let $n \geq 1/a_2^2$, then $\lg n \geq -2 \lg a_2$, or $\lg a_2 \geq -(\lg n)/2$. Then,

$$\begin{aligned}(a_1 n)^{k_1} \lg^{k_2}(a_2 n) &= a_1^{k_1} n^{k_1} (\lg a_2 + \lg n)^{k_2} \\ &\geq a_1^{k_1} n^{k_1} ((\lg n)/2)^{k_2} \\ &= (a_1^{k_1}/2^{k_2}) \cdot n^{k_1} \lg^{k_2} n \\ &= \Omega(n^{k_1} \lg^{k_2} n),\end{aligned}$$

since $a_1^{k_1}/2^{k_2}$ is a positive constant.

★ ***f.*** *The statement in this part is flawed and cannot be solved. First, it lacks the assumption that both sums converge to positive numbers. Second, it fails to clarify the variable that the right-hand side Θ-notation applies to, making the equation either trivial or unambiguous. Removed in fourth printing.*

★ ***g.*** *Similar as above.*

3-6 *Variations on O and Ω*

a. Let c be a positive constant. Suppose that $f(n) \geq cg(n)$ for n in some set S. If S is infinite, then since $g(n)$ is asymptotically nonnegative, $f(n) = \overset{\infty}{\Omega}(g(n))$. Now suppose S is finite and let $n_0 \in S$ be the largest value for which the above inequality holds. Then, $cg(n) > f(n)$ for all $n \geq n_0 + 1$, and since $f(n)$ is asymptotically nonnegative, $f(n) = O(g(n))$.

Theorem 3.1 remains true if we substitute $\overset{\infty}{\Omega}$ for Ω.

b. See the example given in Problem 3-3(b).

c. The advantage of using $\overset{\infty}{\Omega}$-notation instead of Ω-notation is the fact shown in part (a), that all asymptotically nonnegative functions become asymptotically comparable. In practice, if we can prove that a function $f(n)$ is not $O(g(n))$, then we automatically get that $f(n) = \overset{\infty}{\Omega}(g(n))$.

On the other hand, the relation $f(n) = \overset{\infty}{\Omega}(g(n))$ doesn't necessarily say that the function $f(n)$ „dominates" over the function $g(n)$ for sufficiently large n, because for any positive constant c there can still be $f(n) < cg(n)$ for infinitely many integers n. For example, $n^{1+\sin n} = \overset{\infty}{\Omega}(n)$, since for infinitely many n the value of the exponent in $n^{1+\sin n}$ is greater than 1, but at the same time for infinitely many n that exponent can take values less than 1.

Besides, from the practical point of view, the infinitely many integers referred to in the definition of $\overset{\infty}{\Omega}$-notation can be far beyond the maximum input size of the algorithm whose running time we're trying to describe using this notation.

d. The relation $f(n) = \Theta(g(n))$ implies that the function $f(n)$ is asymptotically nonnegative, or $f(n) = |f(n)|$ for sufficiently large n, so $f(n) = O'(g(n))$ follows from $f(n) = O(g(n))$, and the „only if" direction remains true.

To examine the „if" direction, suppose that $f(n) = O'(g(n))$ and $f(n) = \Omega(g(n))$. By definition, if $f(n) = O'(g(n))$ then there exist positive constants c and n_0 such that for all $n \geq n_0$,

$$\begin{cases} 0 \leq \quad f(n) \leq cg(n), & \text{if } f(n) \geq 0, \\ 0 \leq -f(n) \leq cg(n), & \text{if } f(n) < 0. \end{cases}$$

Suppose there is no $n_1 \geq n_0$ such that $f(n) \geq 0$ for all $n \geq n_1$. But then, for any positive constant d the inequalities $0 \leq dg(n) \leq f(n)$ cannot hold for all sufficiently large n, which means that $f(n) \neq \Omega(g(n))$ leading to a contradiction. Hence, the function $f(n)$ is asymptotically nonnegative, as so $f(n) = O(g(n))$.

As we can see, by substituting O' for O, we aren't invalidating Theorem 3.1.

e. We define $\widetilde{\Omega}$ and $\widetilde{\Theta}$ as follows:

$$\widetilde{\Omega}(g(n)) = \{f(n) : \text{ there exist positive constants } c, k, \text{ and } n_0 \text{ such that } 0 \leq cg(n)\lg^k n \leq f(n) \text{ for all } n \geq n_0\},$$

$$\widetilde{\Theta}(g(n)) = \{f(n) : \text{ there exist positive constants } c_1, c_2, k_1, k_2, \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n)\lg^{k_1} n \leq f(n) \leq c_2 g(n)\lg^{k_2} n \text{ for all } n \geq n_0\}.$$

The proof of Theorem 3.1 for $\widetilde{O}$-, $\widetilde{\Omega}$-, and $\widetilde{\Theta}$-notations is analogous to the proof in Exercise 3.2-4 to the original theorem.

By definition, $f(n) = \widetilde{\Theta}(g(n))$ whenever there exist positive constants c_1, c_2, k_1, k_2, and n_0 such that

$$0 \leq c_1 g(n)\lg^{k_1} n \leq f(n) \leq c_2 g(n)\lg^{k_2} n$$

for all $n \geq n_0$. This condition can be decomposed into the combination of inequalities $0 \leq c_1 g(n)\lg^{k_1} n \leq f(n)$ and $0 \leq f(n) \leq c_2 g(n)\lg^{k_2} n$, both true for all $n \geq n_0$. The former implies $f(n) = \widetilde{\Omega}(g(n))$ and the latter implies $f(n) = \widetilde{O}(g(n))$.

For the proof of the opposite direction, suppose that $f(n) = \widetilde{\Omega}(g(n))$ and $f(n) = \widetilde{O}(g(n))$. Then, there exist positive constants c_1, k_1, and n_1 such that

$$0 \leq c_1 g(n)\lg^{k_1} n \leq f(n)$$

for all $n \geq n_1$, and there exist positive constants c_2, k_2, and n_2 such that

$0 \leq f(n) \leq c_2 g(n) \lg^{k_2} n$

for all $n \geq n_2$. Merging both inequalities yields

$0 \leq c_1 g(n) \lg^{k_1} n \leq f(n) \leq c_2 g(n) \lg^{k_2} n$

for all $n \geq \max\{n_1, n_2\}$. Hence, $f(n) = \widetilde{\Theta}(g(n))$.

3-7 *Iterated functions*

a. Since $(n-1)^{(i)} = n - i$, we have $f_0^*(n) = \max\{0, \lceil n \rceil\}$, so $f_0^*(n) = \Theta(n)$.

b. By the definition of the iterated logarithm, $f_1^*(n) = \lg^* n = \Theta(\lg^* n)$.

c. $(n/2)^{(i)} = n/2^i$, so

$$f_1^*(n) = \begin{cases} 0 & \text{if } n \leq 1, \\ \lceil \lg n \rceil & \text{if } n > 1. \end{cases}$$

Hence, $f_1^*(n) = \Theta(\lg n)$.

d. As in part (c),

$$f_2^*(n) = \begin{cases} 0 & \text{if } n \leq 2, \\ \lceil \lg n \rceil - 1 & \text{if } n > 2. \end{cases}$$

Hence, $f_2^*(n) = \Theta(\lg n)$.

e. $\sqrt{n} = n^{1/2}$, so $\left(\sqrt{n}\right)^{(i)} = n^{1/2^i}$. For $n > 2$, the solution to inequality $n^{1/2^i} \leq 2$ with respect to i is $i \geq \lg \lg n$, so

$$f_2^*(n) = \begin{cases} 0 & \text{if } 0 \leq n \leq 2, \\ \lceil \lg \lg n \rceil & \text{if } n > 2. \end{cases}$$

Hence, $f_2^*(n) = \Theta(\lg \lg n)$.

f. If $0 \leq n \leq 1$, $f_1^*(n) = 0$, and if $n > 1$, $f_1^*(n)$ is undefined, because for all integers $i \geq 0$, $\left(\sqrt{n}\right)^{(i)} > 1$.

g. This case is analogous to the one from part (e). Since $(n^{1/3})^{(i)} = n^{1/3^i}$,

$$f_2^*(n) = \begin{cases} 0 & \text{if } n \leq 2, \\ \lceil \log_3 \lg n \rceil & \text{if } n > 2. \end{cases}$$

Hence, $f_2^*(n) = \Theta(\lg \lg n)$.

The table in Figure 3-7 collects the obtained results.

$f(n)$	c	$f_c^*(n)$
$n-1$	0	$\Theta(n)$
$\lg n$	1	$\Theta(\lg^* n)$
$n/2$	1	$\Theta(\lg n)$
$n/2$	2	$\Theta(\lg n)$
$\sqrt{n}$	2	$\Theta(\lg \lg n)$
$\sqrt{n}$	1	undefined
$n^{1/3}$	2	$\Theta(\lg \lg n)$

Figure 3-7 Asymptotic bounds of iterated functions.

4 Divide-and-Conquer

4.1 Multiplying square matrices

4.1-1 Observe, that for $n \times n$ matrices A and B,

$$\begin{pmatrix} A & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} B & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} AB & 0 \\ 0 & 0 \end{pmatrix},$$

where 0 denotes zero matrices. In other words, when multiplying the square matrices A and B, we can pad them with zeros to obtain $m \times m$ matrices $A' = \begin{pmatrix} A & 0 \\ 0 & 0 \end{pmatrix}$ and $B' = \begin{pmatrix} B & 0 \\ 0 & 0 \end{pmatrix}$, and multiply A' by B' instead. The submatrix formed by taking the intersection of the first n rows and the first n columns of $A'B'$ is AB.

We implement this trick in the generalized version of the divide-and-conquer algorithm for multiplying $n \times n$ matrices. First we find m — the least power of 2 greater than of equal to n. Next we create the $m \times m$ matrices A' and B', by copying the corresponding entries of the matrices A and B to the upper left $n \times n$ submatrices of A' and B', while setting the remaining entries to 0. For a given $n \times n$ matrix C, where the matrix product AB will be added, we similarly pad it with zeros to create the $m \times m$ matrix C'. Then we can call MATRIX-MULTIPLY-RECURSIVE(A', B', C', m) and copy the relevant entries of the matrix C' to C.

Creating each matrix A', B', and C' takes $\Theta(m^2)$ time, and copying the result to the matrix C takes $\Theta(n^2)$ time. Let $T(n)$ and $T'(n)$ be the worst-case time to multiply two $n \times n$ matrices using the standard MATRIX-MULTIPLY-RECURSIVE procedure, and its generalized version, respectively. Then, omitting the base case, we get

$$T'(n) = T(m) + \Theta(m^2) + \Theta(n^2).$$

Note that $n \le m < 2n$, so

$$\begin{aligned} T'(n) &\ge T(n) + \Theta(n^2) + \Theta(n^2) \\ &= \Omega(n^3) \end{aligned}$$

and

$$\begin{aligned} T'(n) &\le T(2n) + \Theta((2n)^2) + \Theta(n^2) \\ &= O((2n)^3) \\ &= O(n^3). \end{aligned}$$

Hence, $T'(n) = \Theta(n^3)$.

4.1-2 Let $A_1, A_2, \ldots, A_k$ and $B_1, B_2, \ldots, B_k$ be $n \times n$ matrices. Then

$$A = \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_k \end{pmatrix}$$

is a $kn \times n$ matrix,

$$B = \begin{pmatrix} B_1 & B_2 & \cdots & B_k \end{pmatrix}$$

is an $n \times kn$ matrix, and

$$AB = \begin{pmatrix} A_1B_1 & A_1B_2 & \cdots & A_1B_k \\ A_2B_1 & A_2B_2 & \cdots & A_2B_k \\ \vdots & \vdots & \ddots & \vdots \\ A_kB_1 & A_kB_2 & \cdots & A_kB_k \end{pmatrix}.$$

Each submatrix product A_iB_j in the right-hand side matrix can be computed in $\Theta(n^3)$ time using MATRIX-MULTIPLY-RECURSIVE, so the overall time to multiply A by B is $\Theta(k^2n^3)$.

On the other hand,

$$BA = \sum_{i=1}^{k} B_iA_i,$$

and we can compute it by computing the k products B_iA_i in total time of $\Theta(kn^3)$, then adding them together in $\Theta(kn^2)$ time.

Therefore, multiplying an $n \times kn$ matrix by a $kn \times n$ matrix is asymptotically faster than multiplying these matrices in reverse order, by a factor of k.

4.1-3 At each recursive step the time spent on partitioning the matrices and copying the results is $\Theta(n^2)$, so the recurrence (4.9) changes to

$$T(n) = 8T(n/2) + \Theta(n^2).$$

To show it has the same solution as the original recurrence, let's use the master method from Section 4.5. The above is a master recurrence with $a = 8$, $b = 2$, and driving function $f(n) = \Theta(n^2)$. Since $n^{\log_b a} = n^{\lg 8} = n^3$ and $f(n) = O(n^{3-\epsilon})$ for any $0 < \epsilon \le 1$, we can apply case 1 of the master theorem to get $T(n) = \Theta(n^3)$.

4.1-4 We'll assume that n is an exact power of 2. The procedure MATRIX-ADD-RECURSIVE takes three $n \times n$ matrices A, B and C, and computes $C = C + (A + B)$. Like MATRIX-MULTIPLY-RECURSIVE, in the divide step it partitions the matrices as in equation (4.2). But in the conquer step it adds each submatrix of A to the corresponding submatrix of B and adds the sum to the corresponding submatrix of C:

$$C_{11} = C_{11} + A_{11} + B_{11},$$
$$C_{12} = C_{12} + A_{12} + B_{12},$$
$$C_{21} = C_{21} + A_{21} + B_{21},$$
$$C_{22} = C_{22} + A_{22} + B_{22},$$

which involves four recursive calls.

```
MATRIX-ADD-RECURSIVE(A, B, C, n)
 1  if n == 1
 2      // Base case.
 3      c11 = c11 + a11 + b11
 4      return
 5  // Divide.
 6  partition A, B, and C into n/2 × n/2 submatrices
        A11, A12, A21, A22; B11, B12, B21, B22;
        and C11, C12, C21, C22; respectively
 7  // Conquer.
 8  MATRIX-ADD-RECURSIVE(A11, B11, C11, n/2)
 9  MATRIX-ADD-RECURSIVE(A12, B12, C12, n/2)
10  MATRIX-ADD-RECURSIVE(A21, B21, C21, n/2)
11  MATRIX-ADD-RECURSIVE(A22, B22, C22, n/2)
```

Let $T(n)$ be the worst-case time required by this algorithm to add two $n \times n$ matrices. Each recursive call in lines 8–11 adds two $n/2 \times n/2$ matrices contributing $T(n/2)$ to the overall running time, so all four recursive calls take $4T(n/2)$ time. If we assume that

the algorithm uses index calculations to partition the matrices in line 6, the recurrence for its running time is

$$T(n) = 4T(n/2) + \Theta(1).$$

This is a master recurrence with $a = 4$, $b = 2$, and driving function $f(n) = \Theta(1)$. We have $n^{\log_b a} = n^{\lg 4} = n^2$, and $f(n) = O(n^{2-\epsilon})$, where $0 < \epsilon \leq 2$. Applying case 1 of the master theorem, we obtain the solution $T(n) = \Theta(n^2)$.

However, if the matrices are partitioned by copying, the recurrence changes to

$$T(n) = 4T(n/2) + \Theta(n^2).$$

The watershed function remains unchanged, but the driving function is different. Since $f(n) = \Theta(n^2) = \Theta(n^2 \lg^0 n)$, case 2 of the master theorem applies here. The solution to the recurrence is $T(n) = \Theta(n^2 \lg n)$.

4.2 Strassen's algorithm for matrix multiplication

4.2-1 To simplify the notation we'll omit the parentheses surrounding the single entry of 1×1 matrices and effectively treat such matrices as scalars.

We run Strassen's algorithm on the given 2×2 input matrices A and B, and the 2×2 output matrix C, initialized to 0. In step 1 the algorithm partitions the input matrices into the submatrices:

$$A_{11} = 1,$$
$$A_{12} = 3,$$
$$A_{21} = 7,$$
$$A_{22} = 5,$$
$$B_{11} = 6,$$
$$B_{12} = 8,$$
$$B_{21} = 4,$$
$$B_{22} = 2.$$

Then, in step 2 the following matrices are created:

$$\begin{aligned}
S_1 &= 8 - 2 = 6,\\
S_2 &= 1 + 3 = 4,\\
S_3 &= 7 + 5 = 12,\\
S_4 &= 4 - 6 = -2,\\
S_5 &= 1 + 5 = 6,\\
S_6 &= 6 + 2 = 8,\\
S_7 &= 3 - 5 = -2,\\
S_8 &= 4 + 2 = 6,\\
S_9 &= 1 - 7 = -6,\\
S_{10} &= 6 + 8 = 14.
\end{aligned}$$

In step 3 the algorithm calls itself recursively to compute the matrices $P_1, P_2, \ldots, P_7$. They are all products of two 1×1 matrices, so each recursive call will return right after step 1:

$$\begin{aligned}
P_1 &= 1 \cdot 6 = 6,\\
P_2 &= 4 \cdot 2 = 8,\\
P_3 &= 12 \cdot 6 = 72,\\
P_4 &= 5 \cdot (-2) = -10,\\
P_5 &= 6 \cdot 8 = 48,\\
P_6 &= (-2) \cdot 6 = -12,\\
P_7 &= (-6) \cdot 14 = -84.
\end{aligned}$$

Finally, in step 4 the algorithm updates the four submatrices of the matrix C:

$$\begin{aligned}
C_{11} &= 48 + (-10) - 8 + (-12) = 18,\\
C_{12} &= 6 + 8 = 14,\\
C_{21} &= 72 + (-10) = 62,\\
C_{22} &= 48 + 6 - 72 - (-84) = 66.
\end{aligned}$$

The result of calling Strassen's algorithm on the given input is

$$C = \begin{pmatrix} 18 & 14 \\ 62 & 66 \end{pmatrix}.$$

4.2-2 The pseudocode below calls two auxiliary procedures MATRIX-ADD(A, B, C, n) and MATRIX-SUBTRACT(A, B, C, n). The former implements matrix addition on two $n \times n$

matrices A and B, and accumulates the result into the third matrix: $C = C + (A + B)$. The latter subtracts the second matrix from the first in a similar fashion to compute $C = C + (A - B)$.

```
STRASSEN(A, B, C, n)
 1  if n == 1
 2      c11 = c11 + a11 · b11
 3      return
 4  partition A, B, and C into n/2 × n/2 submatrices
        A11, A12, A21, A22; B11, B12, B21, B22;
        and C11, C12, C21, C22; respectively
 5  create n/2 × n/2 matrices S1, S2, ..., S10
 6  create and zero the entries of n/2 × n/2 matrices P1, P2, ..., P7
 7  MATRIX-SUBTRACT(B12, B22, S1, n/2)
 8  MATRIX-ADD(A11, A12, S2, n/2)
 9  MATRIX-ADD(A21, A22, S3, n/2)
10  MATRIX-SUBTRACT(B21, B11, S4, n/2)
11  MATRIX-ADD(A11, A22, S5, n/2)
12  MATRIX-ADD(B11, B22, S6, n/2)
13  MATRIX-SUBTRACT(A12, A22, S7, n/2)
14  MATRIX-ADD(B21, B22, S8, n/2)
15  MATRIX-SUBTRACT(A11, A21, S9, n/2)
16  MATRIX-ADD(B11, B12, S10, n/2)
17  STRASSEN(A11, S1, P1, n/2)
18  STRASSEN(S2, B22, P2, n/2)
19  STRASSEN(S3, B11, P3, n/2)
20  STRASSEN(A22, S4, P4, n/2)
21  STRASSEN(S5, S6, P5, n/2)
22  STRASSEN(S7, S8, P6, n/2)
23  STRASSEN(S9, S10, P7, n/2)
24  MATRIX-ADD(P5, P4, C11, n/2)
25  MATRIX-SUBTRACT(P6, P2, C11, n/2)
26  MATRIX-ADD(P1, P2, C12, n/2)
27  MATRIX-ADD(P3, P4, C21, n/2)
28  MATRIX-SUBTRACT(P5, P3, C22, n/2)
29  MATRIX-SUBTRACT(P1, P7, C22, n/2)
```

4.2-3 For convenience, we'll assume that n is an exact power of 3. Similarly to equation (4.2), let's partition the matrices into nine $n/3 \times n/3$ submatrices. We can then treat the input matrices as two 3×3 matrices that we multiply using k multiplications. Each of those is in fact a multiplication of two $n/3 \times n/3$ matrices and can be performed by running the algorithm recursively.

Let $T(n)$ be the worst-case time to multiply two $n \times n$ matrices by this algorithm. If we did the partitioning using index calculations, then we get the recurrence

$$T(n) = kT(n/3) + \Theta(1).$$

To solve it, we'll use the master method. We have $a = k$, $b = 3$, and driving function $f(n) = \Theta(1)$. Since $n^{\log_b a} = n^{\log_3 k}$ and $f(n) = O(n^{\log_3 k - \epsilon})$ for $0 < \epsilon \le \log_3 k$, we apply case 1 of the master theorem to get $T(n) = \Theta(n^{\log_3 k})$. According to Problem 3-1(d), $T(n) = o(n^{\lg 7})$ if $\log_3 k < \lg 7$, which gives us $k < 3^{\lg 7} \approx 21.85$. Hence, the largest integer value for k is 21 and then the running time of the algorithm is $T(n) = \Theta(n^{\log_3 21}) = O(n^{2.78})$.

4.2-4 Like in the previous exercise, we can partition the $n \times n$ matrices into either $n/68 \times n/68$, $n/70 \times n/70$, or $n/72 \times n/72$ submatrices, and use a corresponding method to multiply the partitioned matrices as if they were either 68×68, 70×70, or 72×72 matrices. Each multiplication of two submatrices made while applying these methods can be performed by invoking the algorithm recursively.

Let $T_{68}(n)$, $T_{70}(n)$, and $T_{72}(n)$ be the worst-case times to multiply two $n \times n$ matrices using this algorithm, when using each given method. Then, assuming partitioning is done using index calculations, we get the recurrences:

$$T_{68}(n) = 132{,}464 \cdot T_{68}(n/68) + \Theta(1),$$
$$T_{70}(n) = 143{,}640 \cdot T_{70}(n/70) + \Theta(1),$$
$$T_{72}(n) = 155{,}424 \cdot T_{72}(n/72) + \Theta(1).$$

By the master method, their solutions are:

$$T_{68}(n) = \Theta(n^{\log_{68} 132{,}464}) = O(n^{2.795128488}),$$
$$T_{70}(n) = \Theta(n^{\log_{70} 143{,}640}) = O(n^{2.795122690}),$$
$$T_{72}(n) = \Theta(n^{\log_{72} 155{,}424}) = O(n^{2.795147392}).$$

Of all those methods, the one involving multiplying 70×70 matrices yields the best asymptotic running time, and with this choice the algorithm performs better than the $O(n^{2.81})$ Strassen's algorithm.

4.2-5 The algorithm will compute the three values:

$$\alpha = ac,$$
$$\beta = bd,$$
$$\gamma = (a+b)(c+d),$$

each requiring one multiplication of real numbers. Then,

$$\begin{aligned}(a+bi)(c+di) &= (ac-bd)+(ad+bc)i \\ &= (ac-bd)+((a+b)(c+d)-ac-bd)i \\ &= (\alpha-\beta)+(\gamma-\alpha-\beta)i,\end{aligned}$$

so the algorithm will produce $\alpha - \beta$ as the real component, and $\gamma - \alpha - \beta$ as the imaginary component.

4.2-6 Given two $n \times n$ matrices, A and B, let's create a new $2n \times 2n$ matrix

$$D = \begin{pmatrix} A & B \\ 0 & 0 \end{pmatrix}$$

in $\Theta((2n)^2) = \Theta(n^2)$ time. Then we can use the $\Theta(n^\alpha)$-time algorithm for squaring $n \times n$ matrices to compute D^2 in $\Theta((2n)^\alpha) = \Theta(n^\alpha)$ time. The product AB is the upper right $n \times n$ submatrix of D^2:

$$\begin{aligned}D^2 &= \begin{pmatrix} A & B \\ 0 & 0 \end{pmatrix}\begin{pmatrix} A & B \\ 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} A^2 & AB \\ 0 & 0 \end{pmatrix}.\end{aligned}$$

Since $\alpha \geq 2$, the overall running time of the algorithm is $\Theta(n^2 + n^\alpha) = \Theta(n^\alpha)$.

4.3 The substitution method for solving recurrences

4.3-1 ***a.*** Let's adopt the inductive hypothesis that $T(n) \leq cn^2$ for all $n \geq n_0$, where c, $n_0 > 0$ are constants. Assume by induction that this bound holds for all numbers at least as big as n_0 and less than n. In particular, if $n \geq n_0 + 1$, it holds for $n - 1$, yielding

$T(n-1) \le c(n-1)^2$. Substituting into the recurrence yields

$$\begin{aligned} T(n) &\le c(n-1)^2 + n \\ &= cn^2 - 2cn + c + n \\ &= cn^2 + (c - 1/2)(1 - 2n) + 1/2 \\ &\le cn^2, \end{aligned}$$

where the last step holds as long as $(c-1/2)(1-2n)+1/2 \le 0$. If we further constrain n_0 to be no less than 1 (so that $n \ge 2$), we can satisfy this inequality by choosing any $c \ge 2/3$.

The inductive hypothesis holds for the inductive case, and now we'll establish it for the base case, i.e., when $n_0 \le n < n_0 + 1$. Let's pick $n_0 = 1$. By our convention, $T(n)$ is algorithmic, so $T(1)$ is constant, and choosing $c = \max\{2/3, T(1)\}$ satisfies the inequality $T(1) \le c \cdot 1^2$, completing the proof.

We've shown that $T(n) \le cn^2$ for all $n \ge 1$, which implies that the recurrence has the solution $T(n) = O(n^2)$.

b. We adopt the inductive hypothesis that $T(n) \le c \lg n$ for all $n \ge n_0$, where c, $n_0 > 0$ are constants. Assume by induction that this bound holds for all numbers at least as big as n_0 and less than n. Let $n \ge 2n_0$. Then $T(n/2) \le c \lg(n/2)$, and substituting it into the formula for $T(n)$ yields

$$\begin{aligned} T(n) &\le c \lg(n/2) + \Theta(1) \\ &= c \lg n - c \lg 2 + \Theta(1) \\ &= c \lg n - c + \Theta(1) \\ &\le c \lg n, \end{aligned}$$

where the last step holds if we constrain the constant c to be sufficiently large that for $n \ge 2n_0$, c dominates the anonymous constant hidden by the $\Theta(1)$ term.

Now let's pick $n_0 = 2$, so that $\lg n > 0$ for all $n \ge n_0$. Both $T(2)$ and $T(3)$ are constants, so we can choose c large enough to satisfy the inequalities $T(2) \le c \lg 2$ and $T(3) \le c \lg 3$, establishing the inductive hypothesis for the base cases, when $n_0 \le n < 2n_0$.

We've proven that $T(n) \le c \lg n$ for all $n \ge 2$, so $T(n) = O(\lg n)$.

c. We'll prove each bound separately, starting from the lower bound. Consider the inductive hypothesis that $T(n) \ge c_1 n \lg n$ for all $n \ge n_1$, where $c_1, n_1 > 0$ are constants and assume that this bound holds for all numbers no less than n_1 and less than n. Let

$n \geq 2n_1$. Then $T(n/2) \geq c_1(n/2)\lg(n/2)$, and we get

$$\begin{aligned} T(n) &\geq 2c_1(n/2)\lg(n/2) + n \\ &= c_1 n \lg n - c_1 n + n \\ &\geq c_1 n \lg n \qquad \text{(as long as } c_1 \leq 1\text{).} \end{aligned}$$

Now it suffices to show that the bound holds when $n_1 \leq n < 2n_1$. Let's pick $n_1 = 2$, so that $\lg n > 0$ for all $n \geq n_1$. Both $T(2)$ and $T(3)$ are constants, so both inequalities $T(2) \geq c_1(2\lg 2)$ and $T(3) \geq c_1(3\lg 3)$ can be satisfied by picking $c_1 = \min\{1, T(2)/2, T(3)/(3\lg 3)\}$, establishing the inductive hypothesis for the base cases.

We proceed similarly for the upper bound, by considering the inductive hypothesis $T(n) \leq c_2 n \lg n$ for all $n \geq n_2$, where c_2, $n_2 > 0$ are constants. Assume that the bound holds for all numbers no less than n_2 and less than n. If we let $n \geq 2n_2$, we have $T(n/2) \leq c_2(n/2)\lg(n/2)$, and by substitution we obtain

$$\begin{aligned} T(n) &\leq 2c_2(n/2)\lg(n/2) + n \\ &= c_2 n \lg n - c_2 n + n \\ &\leq c_2 n \lg n \qquad \text{(as long as } c_2 \geq 1\text{).} \end{aligned}$$

Let's pick $n_2 = 2$ so we only need to verify that the bound holds when $n = 2$ or $n = 3$. Letting $c_2 = \max\{1, T(2)/2, T(3)/(3\lg 3)\}$ handles the base cases, as it satisfies both $T(2) \leq c_2(2\lg 2)$ and $T(3) \leq c_2(3\lg 3)$.

We've proven that $c_1 n \lg n \leq T(n) \leq c_2 n \lg n$ for all $n \geq 2$, so $T(n) = \Theta(n \lg n)$.

d. We adopt the inductive hypothesis that $T(n) \leq c(n-34)\lg(n-34)$ for all $n \geq n_0$, where $c > 0$, $n_0 > 34$ are constants. Assume that the inductive hypothesis holds for all values no less than n_0, but less than n. Let $n \geq 2n_0 - 34$. Then $T(n/2 + 17) \leq c(n/2 + 17 - 34)\lg(n/2 + 17 - 34)$, so we have

$$\begin{aligned} T(n) &\leq 2c(n/2 + 17 - 34)\lg(n/2 + 17 - 34) + n \\ &= c(n-34)\lg(n/2 - 17) + n \\ &= c(n-34)\lg(n-34) - c(n-34) + n \\ &\leq c(n-34)\lg(n-34). \end{aligned}$$

The last step holds as long as $c(n-34) \geq n$, or $c \geq n/(n-34)$. If $n \geq 35$, then $n/(n-34) \leq 35$, so there must be $c \geq 35$.

Now let $n_0 \leq n < 2n_0 - 34$. The expression $c(n-34)\lg(n-34)$ is positive for all $n > 35$, so let's pick $n_0 = 36$. We satisfy the inequalities $T(36) \leq c(2\lg 2)$ and $T(37) \leq c(3\lg 3)$ by letting $c = \max\{35, T(36)/2, T(37)/(3\lg 3)\}$, establishing the inductive hypothesis for the base cases.

Given the fact that the function $f(n) = n \lg n$ defined over $n \geq 1$ is strictly increasing, we have in particular that for $n > 34$, $f(n-34) < f(n)$. By this observation and by the upper bound shown above, we get that for all $n \geq 36$,

$$\begin{aligned} T(n) &\leq c(n-34)\lg(n-34) \\ &< cn \lg n, \end{aligned}$$

so $T(n) = O(n \lg n)$.

e. In the proof of the lower bound we adopt the inductive hypothesis that $T(n) \geq c_1 n$ for all $n \geq n_1$, where c_1, $n_1 > 0$ are constants. By induction, let the inductive hypothesis hold for all values no less than n_1, but less than n. Also, let $n \geq 3n_1$, which implies that $T(n/3) \geq c_1(n/3)$. Then,

$$\begin{aligned} T(n) &\geq 2c_1(n/3) + \Theta(n) \\ &= c_1 n - (c_1/3)n + \Theta(n) \\ &\geq c_1 n, \end{aligned}$$

where the last step holds if we let c_1 be small enough that for $n \geq 3n_1$, the anonymous function hidden by the $\Theta(n)$ term dominates the quantity $(c_1/3)n$.

Now let $n_1 \leq n < 3n_1$. If we pick $n_1 = 1$, we only need to prove the bound for $n = 1$ and $n = 2$. By further decreasing c_1 we can satisfy both $T(1) \geq c_1 \cdot 1$ and $T(2) \geq c_1 \cdot 2$, handling the base cases.

We can show the upper bound in a similar manner. Our inductive hypothesis is that $T(n) \leq c_2 n$ for all $n \geq n_2$, where c_2, $n_2 > 0$ are constants. By induction, let the inductive hypothesis hold for all values no less than n_2, but less than n. Also, let $n \geq 3n_2$, which implies that $T(n/3) \leq c_2(n/3)$. Then,

$$\begin{aligned} T(n) &\leq 2c_2(n/3) + \Theta(n) \\ &= c_2 n - (c_2/3)n + \Theta(n) \\ &\leq c_2 n, \end{aligned}$$

where for the last step to hold we must choose c_2 large enough, so that for all $n \geq 3n_2$, the quantity $(c_2/3)n$ dominates the anonymous function hidden by the $\Theta(n)$ term.

Pick $n_2 = 1$. We can now increase c_2 sufficiently in order to satisfy $T(1) \leq c_2 \cdot 1$ and $T(2) \leq c_2 \cdot 2$, handling the base cases.

We've proven that $c_1 n \leq T(n) \leq c_2 n$ for all $n \geq 1$, so $T(n) = \Theta(n)$.

f. To show that $T(n) = \Omega(n^2)$ we adopt the inductive hypothesis that $T(n) \geq cn^2$ for all $n \geq n_0$, where $c, n_0 > 0$ are constants, and we assume that this bound holds for all

numbers no less than n_0 and less than n. If $n \geq 2n_0$, we have $T(n) \geq c(n/2)^2$, so

$$\begin{aligned} T(n) &\geq 4c(n/2)^2 + \Theta(n) \\ &= cn^2 + \Theta(n) \\ &\geq cn^2. \end{aligned}$$

The last step holds if we pick n_0 such that the function hidden by the $\Theta(n)$ term is nonnegative for all $n \geq n_0$.

Now let $n_0 \leq n < 2n_0$. By our convention about algorithmic recurrences, all such $T(n)$ are constants, so we can satisfy the inequalities $T(n) \geq cn^2$ for all $n_0 \leq n < 2n_0$ by finding a suitable value for c.

The upper bound of $O(n^2)$ for this recurrence is shown in Exercise 4.3-2, therefore $T(n) = \Theta(n^2)$.

4.3-2 We'll instead consider the recurrence $T(n) = 4T(n/2) + \Theta(n)$, of which the original recurrence is a special case, and we'll show that $T(n) = O(n^2)$.

Our first guess is that $T(n) \leq cn^2$ for all $n \geq n_0$, where c, $n_0 > 0$ are constants. Letting $n \geq 2n_0$ and substituting the inductive hypothesis applied to $T(n/2)$, yields

$$\begin{aligned} T(n) &\leq 4c(n/2)^2 + \Theta(n) \\ &= cn^2 + \Theta(n), \end{aligned}$$

but that does not imply that $T(n) \leq cn^2$ for any choice of c and for any function represented by the $\Theta(n)$ term.

Let's then improve our guess by subtracting a lower-order term: $T(n) \leq cn^2 - dn$, where $d \geq 0$ is another constant. Assume by induction that the bound holds for all values at least as big as n_0, but less than n. For $n \geq 2n_0$ we have $T(n/2) \leq c(n/2)^2 - d(n/2)$, and so

$$\begin{aligned} T(n) &\leq 4(c(n/2)^2 - d(n/2)) + \Theta(n) \\ &= cn^2 - 2dn + \Theta(n) \\ &= cn^2 - dn - (dn - \Theta(n)) \\ &\leq cn^2 - dn, \end{aligned}$$

where the last step holds as long as for $n \geq 2n_0$, the quantity dn dominates the anonymous function hidden by the $\Theta(n)$ term.

Now let $n_0 \leq n < 2n_0$. Let's pick $n_0 = 1$. Choosing $c = T(1) + d$ satisfies the condition $T(1) \leq c \cdot 1^2 - d \cdot 1$, handling the base case and completing the proof.

4.3-3 Our first guess is that $T(n) \leq c2^n$ for all $n \geq n_0$, where c, $n_0 > 0$ are constants. Letting $n \geq n_0 + 1$ and substituting the inductive hypothesis applied to $T(n-1)$, yields

$$\begin{aligned} T(n) &\leq 2c2^{n-1} + 1 \\ &= c2^n + 1, \end{aligned}$$

but that does not imply that $T(n) \leq c2^n$ for any choice of c.

Let's then improve our guess by subtracting a lower-order term: $T(n) \leq c2^n - d$, where $d \geq 0$ is another constant. Assume by induction that the bound holds for all values at least as big as n_0, but less than n. For $n \geq n_0+1$ we have $T(n-1) \leq c2^{n-1}-d$, and so

$$\begin{aligned} T(n) &\leq 2(c2^{n-1} - d) + 1 \\ &= c2^n - 2d + 1 \\ &= c2^n - d - (d - 1) \\ &\leq c2^n - d \qquad \text{(as long as } d \geq 1\text{)}. \end{aligned}$$

Now let $n_0 \leq n < n_0 + 1$. Let's pick $n_0 = 1$. Choosing $c = (T(1) + d)/2$ satisfies the condition $T(1) \leq c \cdot 2^1 - d$, handling the base case and completing the proof.

4.4 The recursion-tree method for solving recurrences

4.4-1 Figure 4.4-1 shows the recursion trees for the recurrences solved in this exercise.

a. For simplicity, assume that n is an exact power of 2 and that the base case is $T(1) = \Theta(1)$. At each level, the problem is reduced to a subproblem of exactly half the size, until the problem size hits $n = 1$, so the height of the recursion tree is $\lg n$. Each level of the tree has a single node which, at depth $i = 0, 1, \ldots, \lg n - 1$, incurs the cost of $(n/2^i)^3 = n^3/8^i$, and the leaf at depth $\lg n$ incurs the cost of $\Theta(1)$. Therefore, the

cost of the whole tree is

$$\begin{aligned} T(n) &= \sum_{i=0}^{\lg n-1} n^3/8^i + \Theta(1) \\ &< \sum_{i=0}^{\infty} n^3/8^i + \Theta(1) \\ &= \frac{1}{1-(1/8)}\, n^3 + \Theta(1) && \text{(by equation (A.7))} \\ &= (8/7)n^3 + \Theta(1) \\ &= O(n^3). \end{aligned}$$

To prove this result, we'll use the substitution method. Let's adopt the inductive hypothesis that $T(n) \le cn^3$ for all $n \ge n_0$, where $c, n_0 > 0$ are constants, and assume that the inductive hypothesis holds for all values at least as big as n_0 and less than n. Let $n \ge 2n_0$, so that $T(n/2) \le c(n/2)^3$. Substituting into the recurrence yields

$$\begin{aligned} T(n) &\le c(n/2)^3 + n^3 \\ &= cn^3/8 + n^3 \\ &= (c/8 + 1)n^3 \\ &\le cn^3, \end{aligned}$$

where the last step holds as long as $c/8 + 1 \le c$, or $c \ge 8/7$.

Let's pick $n_0 = 1$. Picking $c = \max\{8/7, T(1)\}$ satisfies $T(1) \le c \cdot 1^3$, establishing the inductive hypothesis for the base case.

We've shown that $T(n) \le cn^3$ for all $n \ge 1$, which implies that the solution to the recurrence is $T(n) = O(n^3)$.

b. Assume that n is an exact power of 3 and that $T(1) = \Theta(1)$. The subproblem size for a node at depth i is $n/3^i$, and the bottom level of the tree is at depth i such that $n/3^i = 1$ or, equivalently, $i = \log_3 n$. Each level below the top has four times as many nodes as the level above, and so the number of nodes at depth i is 4^i. The total cost of all nodes at a given depth i is $4^i(n/3^i) = (4/3)^i n$. The bottom level contains $4^{\log_3 n} = n^{\log_3 4}$ leaves, each contributing $\Theta(1)$, leading to a total leaf cost of

$\Theta(n^{\log_3 4})$. Therefore, the cost of the entire tree is

$$\begin{aligned}
T(n) &= \sum_{i=0}^{\log_3 n - 1} (4/3)^i n + \Theta(n^{\log_3 4}) \\
&= \frac{(4/3)^{\log_3 n} - 1}{4/3 - 1}\, n + \Theta(n^{\log_3 4}) && \text{(by equation (A.6))} \\
&< 3n(4/3)^{\log_3 n} + \Theta(n^{\log_3 4}) \\
&= 3n\,\frac{n^{\log_3 4}}{n^{\log_3 3}} + \Theta(n^{\log_3 4}) && \text{(by equation (3.21))} \\
&= 3n^{\log_3 4} + \Theta(n^{\log_3 4}) \\
&= O(n^{\log_3 4}).
\end{aligned}$$

To verify this bound using the substitution method, we adopt the inductive hypothesis that $T(n) \le cn^{\log_3 4} - dn$ for all $n \ge n_0$, where $c, n_0 > 0$ and $d \ge 0$ are some constants. Assume that the inductive hypothesis holds for all numbers at least as big as n_0 and less than n. If we let $n \ge 3n_0$, we have $T(n/3) \le c(n/3)^{\log_3 4} - d(n/3)$. Substituting into the recurrence yields

$$\begin{aligned}
T(n) &\le 4\left(c(n/3)^{\log_3 4} - d(n/3)\right) + n \\
&= 4cn^{\log_3 4}/3^{\log_3 4} - 4dn/3 + n \\
&= 4cn^{\log_3 4}/4^{\log_3 3} - 4dn/3 + n \\
&= cn^{\log_3 4} - dn + n(1 - d/3) \\
&\le cn^{\log_3 4} - dn,
\end{aligned}$$

where the last step holds as long as $1 - d/3 \le 0$, or $d \ge 3$.

Let's pick $n_0 = 1$. Picking $c = \max\left\{T(1) + d, (T(2) + 2d)/2^{\log_3 4}\right\}$ satisfies both conditions $T(1) \le c \cdot 1^{\log_3 4} - d \cdot 1$ and $T(2) \le c \cdot 2^{\log_3 4} - d \cdot 2$, establishing the inductive hypothesis for the base cases.

We've shown that $T(n) \le cn^{\log_3 4} - dn \le cn^{\log_3 4}$ for all $n \ge 1$, which implies that the solution to the recurrence is $T(n) = O(n^{\log_3 4})$.

c. Assume that n is an exact power of 2 and that $T(1) = \Theta(1)$. The height of the recursion tree is $\lg n$. At depth i, for $i = 0, 1, \ldots, \lg n - 1$, there are 4^i nodes, each contributing $n/2^i$, so the total cost of all nodes at depth i is $4^i(n/2^i) = 2^i n$. The bottom level, at depth $\lg n$, contains $4^{\lg n} = n^2$ leaves, so the total leaf cost is $\Theta(n^2)$.

Thus, we get

$$\begin{aligned}
T(n) &= \sum_{i=0}^{\lg n-1} 2^i n + \Theta(n^2) \\
&= \frac{2^{\lg n} - 1}{2 - 1} n + \Theta(n^2) && \text{(by equation (A.6))} \\
&< 2^{\lg n} n + \Theta(n^2) \\
&= n^2 + \Theta(n^2) \\
&= O(n^2).
\end{aligned}$$

To formally prove this bound we adopt the inductive hypothesis that $T(n) \le cn^2 - dn$ for all $n \ge n_0$, where $c, n_0 > 0$ and $d \ge 0$ are constants, and assume that the inductive hypothesis holds for all numbers at least as big as n_0 and less than n. When $n \ge 2n_0$ it holds that $T(n/2) \le c(n/2)^2 - d(n/2)$, so

$$\begin{aligned}
T(n) &\le 4(c(n/2)^2 - d(n/2)) + n \\
&= cn^2 - 2dn + n \\
&= cn^2 - dn + n(1 - d) \\
&\le cn^2 - dn && \text{(as long as } d \ge 1\text{)}.
\end{aligned}$$

If we now let $n_0 = 1$, we can choose $c = T(1) + d$ to satisfy $T(1) \le c \cdot 1^2 - d \cdot 1$, establishing the bound for the base case.

We've shown that $T(n) \le cn^2 - dn \le cn^2$ for all $n \ge 1$, which implies that the solution to the recurrence is $T(n) = O(n^2)$.

d. Assume that $T(1) = \Theta(1)$. The height of the recursion tree is n. At depth i, for $i = 0, 1, \ldots, n - 1$, there are 3^i nodes incurring a unit cost each, so the total cost of all nodes at depth i is 3^i. The bottom level, at depth n, contains 3^n leaves, so the total leaf cost is $\Theta(3^n)$. Thus, we get

$$\begin{aligned}
T(n) &= \sum_{i=0}^{n-1} 3^i + \Theta(3^n) \\
&= \frac{3^n - 1}{3 - 1} + \Theta(3^n) && \text{(by equation (A.6))} \\
&< 2 \cdot 3^n + \Theta(3^n) \\
&= O(3^n).
\end{aligned}$$

To prove this bound we adopt the inductive hypothesis that $T(n) \le c3^n - d$ for all $n \ge n_0$, where c, $n_0 > 0$ and $d \ge 0$ are constants, and assume that the inductive

hypothesis holds for all numbers at least as big as n_0 and less than n. When $n \geq n_0 + 1$ it holds that $T(n-1) \leq c3^{n-1} - d$, so

$$\begin{aligned} T(n) &\leq 3(c3^{n-1} - d) + 1 \\ &= c3^n - 3d + 1 \\ &= c3^n - d + (1 - 2d) \\ &\leq c3^n - d \qquad \text{(as long as } d \geq 1/2\text{).} \end{aligned}$$

If we now let $n_0 = 1$, we can choose $c = (T(1) + d)/3$ to satisfy $T(1) \leq c \cdot 3^1 - d$, establishing the bound for the base case.

We've shown that $T(n) \leq c3^n - d \leq c3^n$ for all $n \geq 1$, which implies that the solution to the recurrence is $T(n) = O(3^n)$.

4.4-2 Using the inductive hypothesis $L(n) \geq cn$ for some constant $c > 0$, and assuming that the inductive hypothesis holds for all values less than n, we have

$$\begin{aligned} L(n) &= L(n/3) + L(2n/3) \\ &\geq c(n/3) + c(2n/3) \\ &= cn, \end{aligned}$$

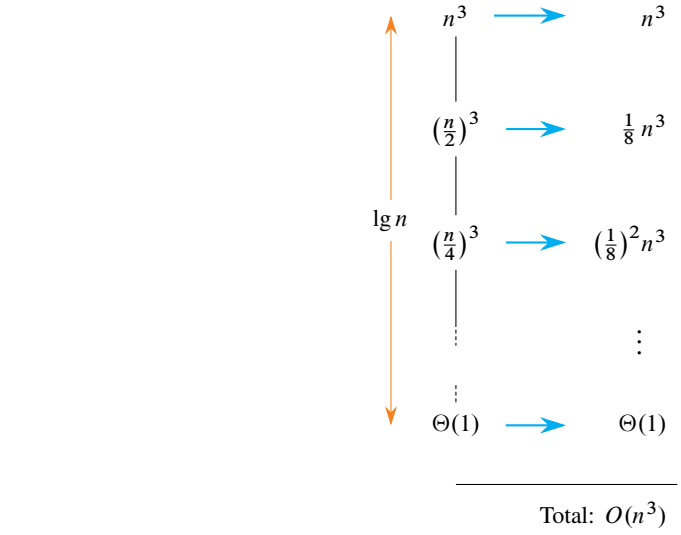
which holds for any $c > 0$. We can now choose $c = 1$ to handle the base case $L(n) = 1$ for $0 < n < n_0$, thereby completing the substitution method for the lower bound of the recurrence.

We've shown that $L(n) \geq cn$ for all $n > 0$, so $L(n) = \Omega(n)$. Combining this result with the upper bound shown in the book, we have that $L(n) = \Theta(n)$.

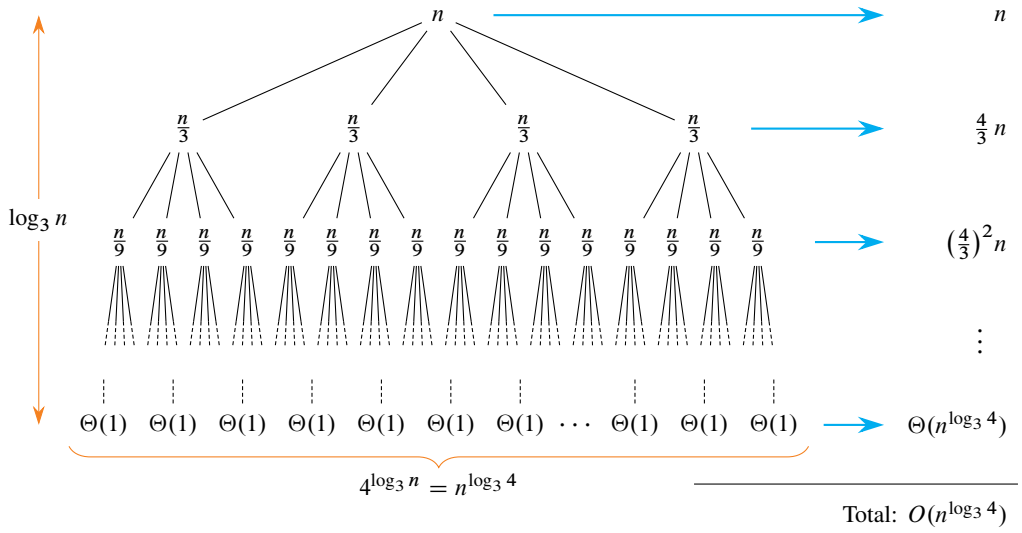
4.4-3 We'll use the inductive hypothesis that $T(n) \geq dn \lg n$ for all $n \geq n_0$, where $d, n_0 > 0$ are some constants. Assume that the inductive hypothesis holds for all numbers at least as big as n_0 and less than n. Let $n \geq 3n_0$. Then, $2n/3 \geq n/3 \geq n_0$, so both $T(n/3) \geq d(n/3)\lg(n/3)$ and $T(2n/3) \geq d(2n/3)\lg(2n/3)$ hold. Substituting into the recurrence yields

$$\begin{aligned} T(n) &\geq d(n/3)\lg(n/3) + d(2n/3)\lg(2n/3) + \Theta(n) \\ &= d(n/3)(\lg n - \lg 3) + 2d(n/3)(\lg n + \lg 2 - \lg 3) + \Theta(n) \\ &= d(n/3)(3\lg n + 2 - 3\lg 3) + \Theta(n) \\ &= dn\lg n - (\lg 3 - 2/3)dn + \Theta(n) \\ &\geq dn\lg n. \end{aligned}$$

The last step holds if we constrain the constant d to be sufficiently small that for $n \geq 3n_0$, the anonymous function hidden by the $\Theta(n)$ term dominates the quantity $(\lg 3 - 2/3)dn$.

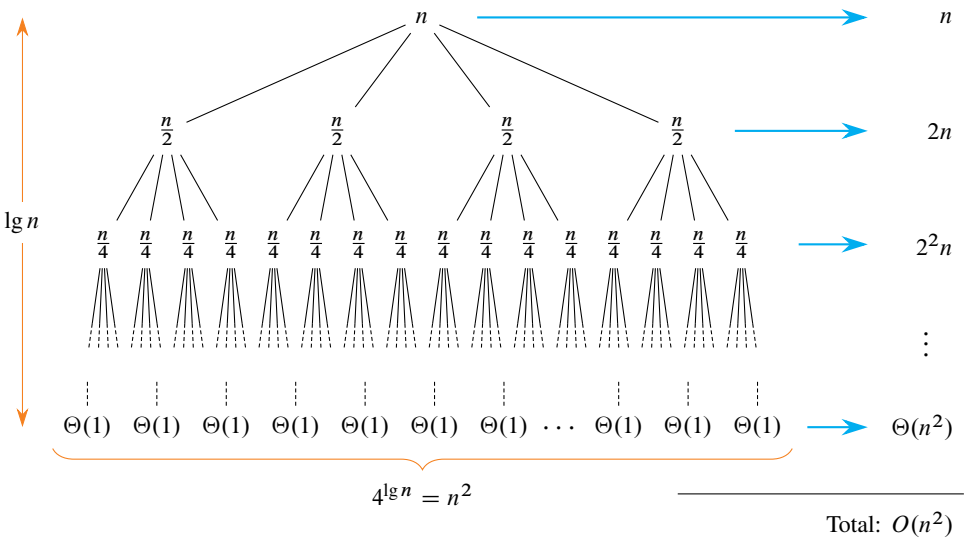


(a)

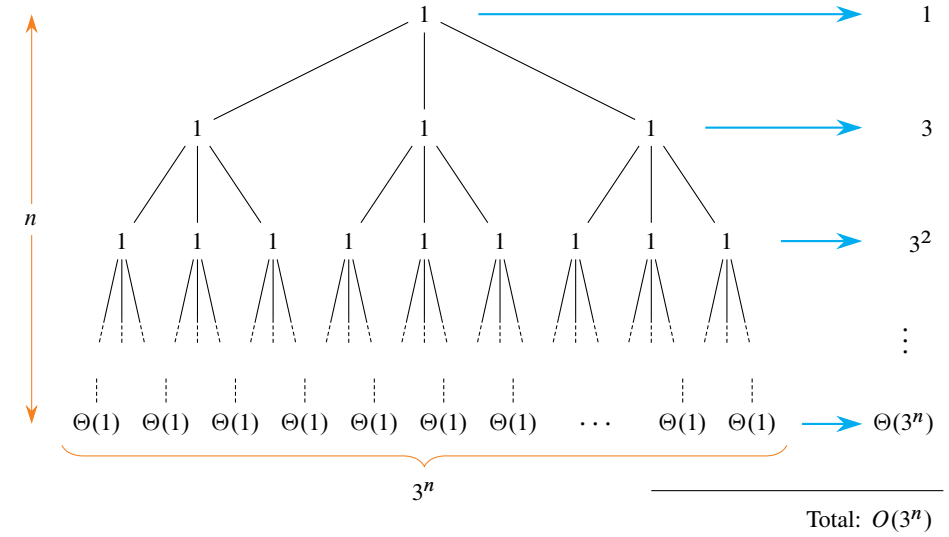


(b)

Figure 4.4-1 **(a)** A recursion tree for the recurrence $T(n) = T(n/2) + n^3$. **(b)** A recursion tree for the recurrence $T(n) = 4T(n/3) + n$.



(c)



(d)

Figure 4.4-1, continued **(c)** A recursion tree for the recurrence $T(n) = 4T(n/2) + n$. **(d)** A recursion tree for the recurrence $T(n) = 3T(n-1) + 1$.

Let's pick $n_0 = 2$, so that $\lg n > 0$ for all $n \geq n_0$. We can decrease d enough, so to satisfy both $T(2) \geq d(2 \lg 2)$ and $T(3) \geq d(3 \lg 3)$, establishing the inductive hypothesis for the base cases.

We've shown that $T(n) \geq dn \lg n$ for all $n \geq 2$, so $T(n) = \Omega(n \lg n)$. Combining this result with the upper bound shown in the book, we get $T(n) = \Theta(n \lg n)$.

4.4-4 We'll use the recursion-tree method to guess an upper bound on $T(n)$. Let's assume that $0 < \alpha \leq 1/2$, because the case when $1/2 \leq \alpha < 1$ is symmetric, and as we'll see, the choice of α doesn't affect the order of growth of an upper bound on $T(n)$. Figure 4.4-4 shows such a tree.

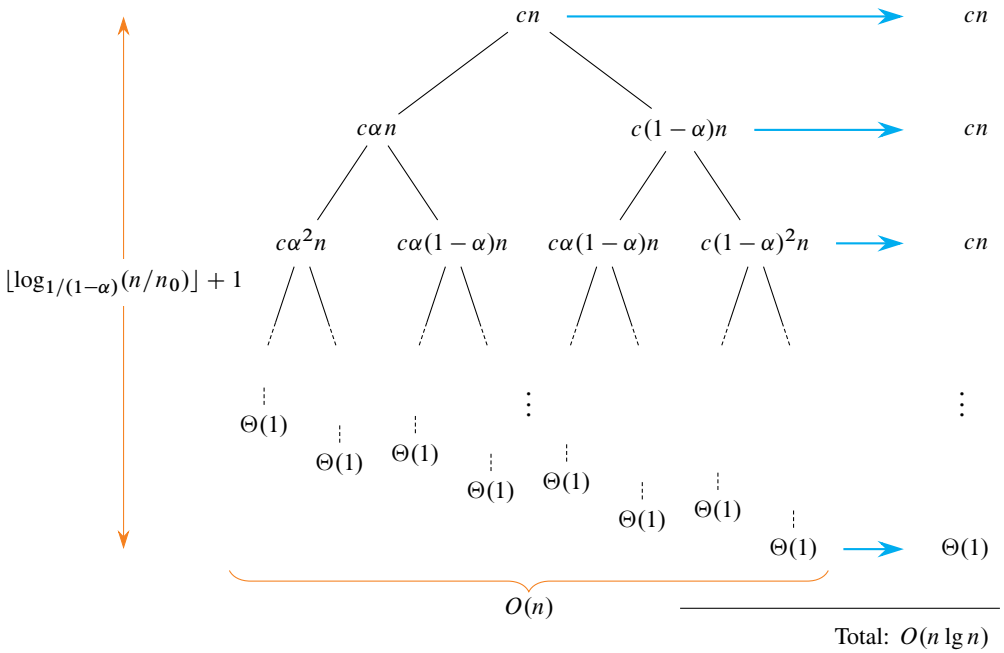


Figure 4.4-4 A recursion tree for the recurrence $T(n) = T(\alpha n) + T((1-\alpha)n) + cn$, where $0 < \alpha \leq 1/2$.

Let $n_0 > 0$ be the implicit threshold constant such that $T(n) = \Theta(1)$ for $0 < n < n_0$, and let c represent the upper-bound constant hidden by the $\Theta(n)$ term for $n \geq n_0$. The root-to-leaf path formed by right-going edges is the longest, where the node at depth i represents a subproblem of size $(1 - \alpha)^i n$. Let h be the height of the tree. At depth h the rightmost node is a leaf, so $(1 - \alpha)^h n < n_0 \leq (1 - \alpha)^{h-1} n$, which gives $h = \lfloor \log_{1/(1-\alpha)}(n/n_0) \rfloor + 1$, and thus $h = \Theta(\lg n)$. The recursion divides the problem into two subproblems, whose sizes sum up to the size of the problem. Since the cost

incurred by a problem of size m is cm, the sum of costs incurred by the subproblems is also cm. Therefore, the total cost of all nodes at any depth is bound by cn. Summing the costs of internal nodes across each level, we have at most cn per level times $h = \Theta(\lg n)$ for a total cost of $O(n \lg n)$ for all internal nodes.

The number of leaves of the tree is described by the following recurrence:

$$L(n) = \begin{cases} 1 & \text{if } n < n_0, \\ L(\alpha n) + L((1-\alpha)n) & \text{if } n \geq n_0. \end{cases}$$

We can apply the substitution method to show it has solution $L(n) = O(n)$ for any $0 < \alpha < 1$. Using the inductive hypothesis $L(n) \leq dn$ for some constant $d > 0$, and assuming that the inductive hypothesis holds for all values less than n, we have

$$\begin{aligned} L(n) &= L(\alpha n) + L((1-\alpha)n) \\ &\leq d\alpha n + d(1-\alpha)n \\ &= dn, \end{aligned}$$

which holds for any $d > 0$ and any $0 < \alpha < 1$. Picking $d = 1$ suffices to handle the base case $L(1) = 1$ for $0 < n < n_0$.

Returning to recurrence $T(n)$, we obtain its upper bound of $O(n \lg n) + O(n) = O(n \lg n)$.

4.5 The master method for solving recurrences

4.5-1 In all recurrences studied in this exercise we have $a = 2$ and $b = 4$, so the recurrences share the same watershed function $n^{\log_b a} = n^{\log_4 2} = n^{1/2} = \sqrt{n}$, but differ in driving functions $f(n)$.

a. Here we have $f(n) = 1 = O(n^{1/2-\epsilon})$ for any $\epsilon \leq 1/2$, so we can apply case 1 of the master theorem to conclude that the solution is $T(n) = \Theta(\sqrt{n})$.

b. Since $f(n) = \sqrt{n} = \Theta(n^{1/2} \lg^0 n)$, we apply case 2 to obtain the solution $T(n) = \Theta(\sqrt{n} \lg n)$.

c. Since $f(n) = \sqrt{n} \lg^2 n = \Theta(n^{1/2} \lg^2 n)$, we apply case 2 to obtain the solution $T(n) = \Theta(\sqrt{n} \lg^3 n)$.

d. We have $f(n) = n = \Omega(n^{1/2+\epsilon})$ for any $\epsilon \geq 1/2$. Also, it holds that $2f(n/4) = 2(n/4) = n/2 \leq cf(n)$ for any $c \geq 1/2$ and all sufficiently large n, satisfying the regularity condition. By case 3, the solution to the recurrence is $T(n) = \Theta(n)$.

e. We have $f(n) = n^2 = \Omega(n^{1/2+\epsilon})$ for any $\epsilon \geq 3/2$. Also, it holds that $2f(n/4) = 2(n/4)^2 = n^2/8 \leq cf(n)$ for any $c \geq 1/8$ and all n, satisfying the regularity condition. By case 3, the solution to the recurrence is $T(n) = \Theta(n^2)$.

4.5-2 The recurrence describing the worst-case running time of the professor's algorithm is

$$T(n) = aT(n/4) + \Theta(n^2).$$

For this algorithm to run asymptotically faster than Strassen's algorithm, it should be $T(n) = o(n^{\lg 7})$. Observe that neither case 2 nor 3 of the master theorem can't resolve $T(n)$ to be asymptotically less than the driving function $f(n)$. Since $f(n) = \Theta(n^2)$, neither of those cases is interesting for us.

That leaves us with case 1, for which to apply there should be $\Theta(n^2) = O(n^{\log_4 a - \epsilon})$ for some $\epsilon > 0$, which holds for $a > 16$. Then the solution to the recurrence is $T(n) = \Theta(n^{\log_4 a})$, so

$$\begin{aligned}
\log_4 a &= \frac{\lg a}{\lg 4} \\
&= \frac{\lg a}{2} \\
&= \lg \sqrt{a} \\
&< \lg 7,
\end{aligned}$$

holds when $a < 49$.

The largest integer a we are looking for, is $a = 48$.

4.5-3 We have $a = 1$ and $b = 2$, so the watershed function is $n^{\log_b a} = n^{\lg 1} = 1$. Since $f(n) = \Theta(1) = \Theta(\lg^0 n)$, case 2 of the master theorem applies, and the solution is $T(n) = \Theta(\lg n)$.

4.5-4 Let c be a constant satisfying the regularity condition for the given constants a and b, and the function $f(n)$. Then,

$$\begin{aligned}
af(n/b) &= \lg(n/2) \\
&= \lg n - 1 \\
&= (1 - 1/\lg n)\lg n \\
&\leq c \lg n.
\end{aligned}$$

The quantity $1 - 1/\lg n$ is less than 1 for $n > 1$, and can be made arbitrarily close to 1 if n is sufficiently large. Therefore, $c \geq 1$.

Furthermore, by equation (3.24), $\lg n = o(n^\epsilon)$ for any $\epsilon > 0$, which means that $\lg n \neq \Omega(n^\epsilon) = \Omega(n^{\log_b a + \epsilon})$.

4.5-5 As long as $b > 1$, $0 < a < b$, and $\epsilon > 0$ is sufficiently small, it holds that

$$\begin{aligned} f(n) &\geq 2^{\lg n} \\ &= n \\ &= \Omega(n^{\log_b a + \epsilon}). \end{aligned}$$

Furthermore,

$$\begin{aligned} a f(n/b) &= a 2^{\lceil \lg(n/b) \rceil} \\ &< a 2^{\lg n - \lg b + 1} && \text{(by inequality (3.2))} \\ &= (2a/b) 2^{\lg n} \\ &\leq (2a/b) f(n), \end{aligned}$$

so the regularity condition isn't satisfied for all sufficiently large n, if $2a/b \geq 1$.

Therefore, by choosing any constants a, b and ϵ, such that $1/2 < b/2 \leq a < b$ and $0 < \epsilon \leq 1 - \log_b a$, we satisfy all the conditions in case 3, except the regularity condition.

⋆ 4.6 Proof of the continuous master theorem

4.6-1

$$\begin{aligned} \sum_{j=0}^{\lfloor \log_b n \rfloor} (\log_b n - j)^k &\geq \sum_{j=0}^{\lfloor \log_b n \rfloor} (\lfloor \log_b n \rfloor - j)^k \\ &= \sum_{j=0}^{\lfloor \log_b n \rfloor} j^k && \text{(reindexing)} \\ &= \Omega(\lfloor \log_b n \rfloor^{k+1}) && \text{(by Exercise A.1-5)} \\ &= \Omega(\log_b^{k+1} n) && \text{(by Exercise 3.3-3)} \end{aligned}$$

4.6-2 ⋆ *Note that the function $f(n) = 0$ satisfies the regularity condition for any constants $a > 0$, $b > 1$, and $c < 1$, but it's not true that $f(n) = \Omega(n^{\log_b a + \epsilon})$ for any $\epsilon > 0$. Hence, we need to additionally assume that $f(n)$ is asymptotically positive.*

Let $f(n)$ be a driving function and let $a > 0$, $b > 1$, and $n_0 > 0$ be constants, such that $f(n) > 0$ for all $n \geq n_0$, and that $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all $n \geq bn_0$. Let $j = \lfloor \log_b(n/n_0) \rfloor$, so that $n/b^{j-1} \geq bn_0$ and $n/b^j < bn_0$. Then, for $n \geq n_0$,

$$\begin{aligned} f(n) &\geq (a/c) f(n/b) \\ &\geq (a/c)(a/c) f((n/b)/b) \\ &\quad \vdots \\ &\geq (a/c)^j f(n/b^j) \\ &= (a/c)^{\lfloor \log_b(n/n_0) \rfloor} f\left(n/b^{\lfloor \log_b(n/n_0) \rfloor}\right). \end{aligned}$$

By inequality (3.2),

$$\begin{aligned} \frac{n}{b^{\lfloor \log_b(n/n_0) \rfloor}} &\geq \frac{n}{b^{\log_b(n/n_0)}} \\ &= \frac{n}{n/n_0} \\ &= n_0, \end{aligned}$$

and

$$\begin{aligned} \frac{n}{b^{\lfloor \log_b(n/n_0) \rfloor}} &\leq \frac{n}{b^{\log_b(n/n_0)-1}} \\ &= \frac{bn}{n/n_0} \\ &= bn_0, \end{aligned}$$

so the quantity $f\left(n/b^{\lfloor \log_b(n/n_0) \rfloor}\right)$ belongs to the image of the interval $[n_0, bn_0]$ under function f. Let μ be any positive lower bound of the image $f([n_0, bn_0])$; such a lower bound exists, by our assumption on $f(n)$, e.g., $\mu = \inf f([n_0, bn_0])$. Depending on the values of a and c, the ratio a/c may be either less or no less than 1. If $a/c < 1$, we have $(a/c)^{\lfloor \log_b(n/n_0) \rfloor} \geq (a/c)^{\log_b(n/n_0)}$, and thus

$$\begin{aligned} f(n) &\geq (a/c)^{\lfloor \log_b(n/n_0) \rfloor} f\left(n/b^{\lfloor \log_b(n/n_0) \rfloor}\right) \\ &\geq (a/c)^{\log_b(n/n_0)} \mu \\ &= \frac{(n/n_0)^{\log_b a}}{(n/n_0)^{\log_b c}} \mu \qquad \text{(by equations (3.18), (3.20), and (3.21))} \\ &= n^{\log_b a - \log_b c} \cdot n_0^{\log_b c - \log_b a} \cdot \mu. \end{aligned}$$

In case that $a/c \geq 1$, it holds $(a/c)^{\lfloor \log_b(n/n_0) \rfloor} \geq (a/c)^{\log_b(n/n_0)-1}$, so

$$\begin{aligned} f(n) &\geq (a/c)^{\lfloor \log_b(n/n_0) \rfloor} f\left(n/b^{\lfloor \log_b(n/n_0) \rfloor}\right) \\ &\geq (a/c)^{\log_b(n/n_0)-1}\mu \\ &= \frac{c(n/n_0)^{\log_b a}}{a(n/n_0)^{\log_b c}}\,\mu \qquad \text{(by equations (3.18), (3.20), and (3.21))} \\ &= n^{\log_b a - \log_b c} \cdot n_0^{\log_b c - \log_b a} \cdot (c/a) \cdot \mu. \end{aligned}$$

Regardless of the relation between a and c, we can define a constant $d > 0$ to hide all factors independent on n in the above bounds. Also, since $0 < c < 1$ and $b > 1$, $\log_b c < 0$, so let $\epsilon = -\log_b c > 0$. Then, the lower bound simplifies to

$$f(n) \geq n^{\log_b a + \epsilon} \cdot d,$$

which holds for all $n \geq n_0$. Hence $f(n) = \Omega(n^{\log_b a + \epsilon})$.

4.6-3 ★ The statement can be included into Lemma 4.3 as an extension to case 2:

2′. If $f(n) = \Theta(n^{\log_b a}/\lg n)$, then $g(n) = \Theta(n^{\log_b a} \lg\lg n)$.

We'll prove it similarly as the original case 2.

Since $f(n) = \Theta(n^{\log_b a}/\lg n)$, we have that $f(n/b^j) = \Theta((n/b^j)^{\log_b a}/\lg(n/b^j))$. However, to avoid zero from appearing in the denominator, in equation (4.19) we'll substitute the equivalent bound $f(n/b^j) = \Theta((n/b^j)^{\log_b a}/\lg(n/b^{j-1}))$, since $b > 1$

is a constant:

$$
\begin{aligned}
g(n) &= \sum_{j=0}^{\lfloor \log_b n \rfloor} a^j \, \Theta\left(\left(\frac{n}{b^j}\right)^{\log_b a} \Big/ \lg\left(\frac{n}{b^{j-1}}\right) \right) \\
&= \Theta\left(\sum_{j=0}^{\lfloor \log_b n \rfloor} a^j \left(\frac{n}{b^j}\right)^{\log_b a} \Big/ \lg\left(\frac{n}{b^{j-1}}\right) \right) && \text{(by Problem 3-5(c), repeatedly)} \\
&= \Theta\left(n^{\log_b a} \sum_{j=0}^{\lfloor \log_b n \rfloor} \frac{a^j}{b^{j \log_b a} \lg(n/b^{j-1})} \right) \\
&= \Theta\left(n^{\log_b a} \sum_{j=0}^{\lfloor \log_b n \rfloor} \frac{1}{\lg(n/b^{j-1})} \right) && \text{(by equation (3.21))} \\
&= \Theta\left(n^{\log_b a} \sum_{j=0}^{\lfloor \log_b n \rfloor} \frac{\log_b 2}{\log_b(n/b^{j-1})} \right) && \text{(by equation (3.19))} \\
&= \Theta\left(n^{\log_b a} \log_b 2 \sum_{j=0}^{\lfloor \log_b n \rfloor} \frac{1}{\log_b n - (j-1)} \right) && \text{(by equations (3.18) and (3.20))} \\
&= \Theta\left(n^{\log_b a} \sum_{j=0}^{\lfloor \log_b n \rfloor} \frac{1}{\log_b n - j + 1} \right) && (\log_b 2 > 0 \text{ is a constant}).
\end{aligned}
$$

The summation within the Θ-notation can be bounded from above as follows:

$$
\begin{aligned}
\sum_{j=0}^{\lfloor \log_b n \rfloor} \frac{1}{\log_b n - j + 1} &\le \sum_{j=0}^{\lfloor \log_b n \rfloor} \frac{1}{\lfloor \log_b n \rfloor - j + 1} \\
&= \sum_{j=1}^{\lfloor \log_b n \rfloor + 1} \frac{1}{j} && \text{(reindexing)} \\
&= \ln(\lfloor \log_b n \rfloor + 1) + O(1) && \text{(by equation (A.9))} \\
&\le \ln(2 \log_b n) + O(1) && \text{(as long as } n \ge b) \\
&= \frac{\lg(\lg n / \lg b)}{\lg e} + \ln 2 + O(1) && \text{(by equations (3.18) and (3.19) (twice))} \\
&= \frac{\lg \lg n}{\lg e} - \frac{\lg \lg b}{\lg e} + \ln 2 + O(1) && \text{(by equations (3.18) and (3.20))} \\
&= O(\lg \lg n) && \text{(after ignoring constants),}
\end{aligned}
$$

and from below as follows:

$$\begin{aligned}
\sum_{j=0}^{\lfloor \log_b n \rfloor} \frac{1}{\log_b n - j + 1} &\geq \sum_{j=0}^{\lfloor \log_b n \rfloor} \frac{1}{\lfloor \log_b n \rfloor + 1 - j + 1} \\
&= \sum_{j=2}^{\lfloor \log_b n \rfloor + 2} \frac{1}{j} && \text{(reindexing)} \\
&= \left(\sum_{j=1}^{\lfloor \log_b n \rfloor + 2} \frac{1}{j} \right) - 1 \\
&= \ln(\lfloor \log_b n \rfloor + 2) + O(1) - 1 && \text{(by equation (A.9))} \\
&> \ln(\log_b n) + O(1) - 1 \\
&= \frac{\lg(\lg n / \lg b)}{\lg e} + O(1) - 1 && \text{(by equation (3.19) (twice))} \\
&= \frac{\lg \lg n}{\lg e} - \frac{\lg \lg b}{\lg e} + O(1) - 1 && \text{(by equations (3.18) and (3.20))} \\
&= \Omega(\lg \lg n) && \text{(after ignoring constants).}
\end{aligned}$$

Hence, we can conclude that $g(n) = \Theta(n^{\log_b a} \lg \lg n)$, thereby completing the proof of case $2'$ of Lemma 4.3.

In a similar fashion we can define an extension to case 2 in Theorem 4.4:

$2'$. If $f(n) = \Theta(n^{\log_b a} / \lg n)$, then $T(n) = \Theta(n^{\log_b a} \lg \lg n)$.

For the same functions $f'(n)$ and $T'(n)$ defined in the proof of Theorem 4.4, we have

$$\begin{aligned}
f'(n) &= f(n_0 n) \\
&= \Theta((n_0 n)^{\log_b a} / \lg(n_0 n)) \\
&= \Theta(n^{\log_b a} / \lg n),
\end{aligned}$$

since a, b, and n_0 are all constants. Thus, by case $2'$ of Lemma 4.3 shown earlier, we have that the summation in equation (4.18) of Lemma 4.2 is $\Theta(n^{\log_b a} \lg \lg n)$, yielding

$$\begin{aligned}
T(n) &= T'(n / n_0) \\
&= \Theta((n/n_0)^{\log_b a}) + \Theta((n/n_0)^{\log_b a} \lg \lg(n/n_0)) \\
&= \Theta(n^{\log_b a}) + \Theta(n^{\log_b a} \lg \lg n) \\
&= \Theta(n^{\log_b a} \lg \lg n) && \text{(by Problem 3-5(c)).}
\end{aligned}$$

⋆ 4.7 Akra-Bazzi recurrences

4.7-1 ⋆ Let $n_0 > 0$ be the threshold constant for recurrence $T'(n)$, such that $T'(n) = \Theta(1)$ for all $n < n_0$. For $n > 0$, define $T''(n) = T'(n)/c$. Then, for all $n \geq n_0$,

$$\begin{aligned} T''(n) &= \frac{cf(n)}{c} + \sum_{i=1}^{k} \frac{a_i T'(n/b_i)}{c} \\ &= f(n) + \sum_{i=1}^{k} a_i T''(n/b_i). \end{aligned}$$

Now, if we let $T'(n) = cT(n)$ for any $n < n_0$, we have that recurrences $T(n)$ and $T''(n)$ are identical for all $n > 0$. Therefore, whatever the solution $\Theta(g(n))$ to $T(n)$ is, the solution to $T'(n)$ is $c \cdot \Theta(g(n)) = \Theta(g(n))$, and so any constant multiplier of the driving function in any Akra-Bazzi recurrence doesn't affect its asymptotic solution.

4.7-2 Choose any $\phi \geq 1$. Let $1 \leq \psi \leq \phi$, and thus $1^2 \leq \psi^2 \leq \phi^2$. Multiplying all sides by n^2, where $n > 0$, yields

$$n^2 \leq \psi^2 n^2 \leq \phi^2 n^2$$

for all $n > 0$. Since $n^2/(\phi^2 + 1) < n^2$ and $\phi^2 n^2 < (\phi^2 + 1)n^2$, we can let $d = \phi^2 + 1 > 1$ to get

$$f(n)/d < f(\psi n) < df(n).$$

Thus, the function $f(n) = n^2$ satisfies the polynomial-growth condition.

Now for $f(n) = 2^n$ let $\phi = \psi = 2$. We have

$$\begin{aligned} f(\psi n) &= 2^{\psi n} \\ &= 2^{2n} \\ &= 2^n \cdot 2^n \\ &\leq d\,2^n \\ &= df(n), \end{aligned}$$

as long as $d \geq 2^n$, so there is no suitable constant d for the second to last step to hold for all sufficiently large n. We conclude that $f(n) = 2^n$ doesn't satisfy the polynomial-growth condition.

4.7-3 *$f(n)$ is asymptotically nonnegative, not only asymptotically positive.*

Since $f(n)$ satisfies the polynomial-growth condition, there exist constants $\widehat{n} > 0$ and $d > 1$ such that $f(n)/d \le df(n)$ for all $n \ge \widehat{n}$. Then,

$$\begin{aligned} 0 &\le df(n) - f(n)/d \\ &= f(n)(d - 1/d) \\ &= f(n)(d^2 - 1)/d. \end{aligned}$$

Because $d > 1$, $d^2 - 1 > 0$, and $(d^2 - 1)/d > 0$, so for the above inequality to hold, there should be $f(n) \ge 0$, which means that $f(n)$ is asymptotically nonnegative.

4.7-4  *Work in progress.*

4.7-5 In this exercise, when calculating integrals we ignore the constant of integration, since in all cases it is swallowed by the Θ-notation in equation (4.23).

a. This is an Akra-Bazzi recurrence with $a_1 = a_2 = a_3 = 1$, $b_1 = 2$, $b_2 = 3$, $b_3 = 6$, and $f(n) = n \lg n$. Note that $1/2 + 1/3 + 1/6 = 1$, so $p = 1$. We'll also need the following integral:

$$\begin{aligned} I &= \int \frac{\lg x}{x}\, dx \\ &= \int \lg x (\ln x)'\, dx \\ &= \lg x \ln x - \int \frac{\ln x}{x \ln 2}\, dx && \text{(integrating by parts)} \\ &= \frac{\lg^2 x}{\lg e} - \int \frac{\lg x}{x}\, dx && \text{(by equation (3.19))} \\ &= \frac{\lg^2 x}{\lg e} - I, \end{aligned}$$

which gives

$$I = \frac{\lg^2 x}{2 \lg e}.$$

Finally, by identity (4.23) we have

$$\begin{aligned}
T(n) &= \Theta\left(n\left(1+\int_1^n \frac{x\lg x}{x^2}\,dx\right)\right)\\
&= \Theta\left(n\left(1+\int_1^n \frac{\lg x}{x}\,dx\right)\right)\\
&= \Theta\left(n\left(1+\left[\frac{\lg^2 x}{2\lg e}\right]_1^n\right)\right)\\
&= \Theta\left(n\left(1+\frac{\lg^2 n}{2\lg e}\right)\right)\\
&= \Theta(n\lg^2 n).
\end{aligned}$$

b. *Work in progress.*

c. In this recurrence we have $a_1 = 2/3$, $a_2 = 1/3$, $b_1 = 3$, $b_2 = 3/2$, and $f(n) = \lg n$. Observe that $a_1 + a_2 = 1$, in which case we immediately obtain $p = 0$. Then,

$$\begin{aligned}
T(n) &= \Theta\left(n^0\left(1+\int_1^n \frac{\lg x}{x}\,dx\right)\right)\\
&= \Theta\left(1+\left[\frac{\lg^2 x}{2\lg e}\right]_1^n\right) && \text{(we calculated this integral in part (a))}\\
&= \Theta\left(1+\frac{\lg^2 n}{2\lg e}\right)\\
&= \Theta(\lg^2 n).
\end{aligned}$$

d. Here, $a_1 = 1/3$, $b_1 = 3$, and $f(n) = 1/n$. The term $\frac{1/3}{3^p} = 3^{-p-1}$ equals 1, if $p = -1$. We know from calculus that $\int dx/x = \ln x$, and thus, by equation (4.23):

$$\begin{aligned}
T(n) &= \Theta\left(n^{-1}\left(1+\int_1^n \frac{1/x}{x^0}\,dx\right)\right)\\
&= \Theta\left((1/n)\left(1+\int_1^n \frac{dx}{x}\right)\right)\\
&= \Theta\big((1/n)\big(1+[\ln x]_1^n\big)\big)\\
&= \Theta\big((1/n)(1+\ln n)\big)\\
&= \Theta((\lg n)/n).
\end{aligned}$$

e. Here, $a_1 = a_2 = 3$, $b_1 = 3$, $b_2 = 3/2$, and $f(n) = n^2$. We evaluate the sum

$$\frac{3}{3^p} + \frac{3}{(3/2)^p} = \frac{1+2^p}{3^{p-1}}$$

using different values for p, to find that the sum achieves 1 when $p = 3$. Since $\int dx/x^2 = -1/x$, we have

$$\begin{aligned}
T(n) &= \Theta\left(n^3\left(1 + \int_1^n \frac{x^2}{x^4}\,dx\right)\right) \\
&= \Theta\left(n^3\left(1 + \int_1^n \frac{dx}{x^2}\right)\right) \\
&= \Theta\big(n^3\big(1 + [-1/x]_1^n\big)\big) \\
&= \Theta\big(n^3\big(1 + (-1/n + 1)\big)\big) \\
&= \Theta(n^3(2 - 1/n)) \\
&= \Theta(n^3).
\end{aligned}$$

4.7-6  *Work in progress.*

Problems

4-1 ***Recurrence examples***

We use the master method in parts (a)–(f).

a. We have $a = 2$ and $b = 2$, which means that $n^{\log_b a} = n^{\log_2 2} = n$, and $f(n) = n^3 = \Omega(n^{1+\epsilon})$ for any $\epsilon \le 2$. Also, it holds that $2f(n/2) = 2n^3/8 \le cf(n)$ for any $c \ge 1/4$, satisfying the regularity condition. By case 3, the solution to the recurrence is $T(n) = \Theta(n^3)$.

b. We have $a = 1$ and $b = 11/8$, which means that $n^{\log_b a} = n^{\log_{11/8} 1} = 1$, and $f(n) = n = \Omega(n^\epsilon)$ for $\epsilon \le 1$. Also, it holds that $f(8n/11) = (8/11)n \le cf(n)$ for any $c \ge 8/11$, satisfying the regularity condition. By case 3, the solution to the recurrence is $T(n) = \Theta(n)$.

c. We have $a = 16$ and $b = 4$, which means that $n^{\log_b a} = n^{\log_4 16} = n^2$. Since $f(n) = n^2 = \Theta(n^{\log_b a})$, we apply case 2 to get $T(n) = \Theta(n^2 \lg n)$.

d. We have $a = 4$ and $b = 2$, which means that $n^{\log_b a} = n^{\log_2 4} = n^2$. Since $f(n) = n^2 \lg n = \Theta(n^{\log_b a} \lg n)$, we apply case 2 to get $T(n) = \Theta(n^2 \lg^2 n)$.

e. We have $a = 8$ and $b = 3$, which means that $n^{\log_b a} = n^{\log_3 8} = O(n^{1.9})$, and $f(n) = n^2 = \Omega(n^{\log_b a + \epsilon})$, where ϵ can be as large as approximately 0.1. Also, it holds that $8f(n/3) = 8n^2/9 \le cf(n)$ for any $c \ge 8/9$, satisfying the regularity condition. By case 3, the solution to the recurrence is $T(n) = \Theta(n^2)$.

f. We have $a = 7$ and $b = 2$, which means that $n^{\log_b a} = n^{\log_2 7} = \Omega(n^{2.8})$. Since $f(n) = n^2 \lg n = O(n^{\log_b a - \epsilon})$, where ϵ can be as large as approximately 0.8, we apply case 1 to get $T(n) = \Theta(n^{\lg 7})$.

g. We already showed in Exercise 4.5-1(b) that this recurrence has solution $T(n) = \Theta(\sqrt{n} \lg n)$.

h. This is not a master recurrence, so we can't apply the master method to it. Instead, we'll guess the solution and verify it using the substitution method.

The recurrence adds up the square of the problem size to the total cost, and decreases the problem size by 2. The partial costs are therefore n^2, $(n-2)^2$, $(n-4)^2$, ..., so the total cost resolves roughly to

$$\begin{aligned}\sum_{k=0}^{n} (n-k)^2 &= \sum_{k=0}^{n} k^2 \\ &= \frac{n(n+1)(2n+1)}{6} && \text{(by equation (A.4))} \\ &= \Theta(n^3).\end{aligned}$$

Let's then adopt an inductive hypothesis that $T(n) \ge c_1 n^3$ for all $n \ge n_1$, where c_1, $n_1 > 0$ are constants. Assume by induction that the hypothesis holds for all numbers at least as big as n_1 and less than n. Let $n \ge n_1 + 2$, so that $T(n-2) \ge c_1(n-2)^3$. By substitution,

$$\begin{aligned}T(n) &\ge c_1(n-2)^3 + n^2 \\ &= c_1(n^3 - 3 \cdot 2n^2 + 3 \cdot 2^2 n - 2^3) + n^2 \\ &= c_1 n^3 + (1 - 6c_1)n^2 + 12c_1 n - 8c_1.\end{aligned}$$

Let $f_c(n) = (1-6c)n^2 + 12cn - 8c$, where c is a positive constant. We have

$$\begin{aligned} f_c(n) &> (1-6c)n^2 - 8c \\ &\geq (1-6c)n^2 - 8cn^2 && \text{(as long as } n \geq 1) \\ &= (1-14c)n^2 \\ &\geq 0 && \text{(as long as } c \leq 1/14). \end{aligned}$$

Thus, we get that for $c_1 \leq 1/14$, $T(n) \geq c_1 n^3 + f_{c_1}(n) \geq c_1 n^3$ for all $n \geq n_1 + 2$, which proves the inductive hypothesis holds for the inductive case.

Now let $n_1 \leq n < n_1 + 2$ and let's pick $n_1 = 1$. Both $T(1)$ and $T(2)$ are constants, and picking $c_1 = \min\{1/14, T(1), T(2)/8\}$ satisfies both inequalities $T(1) \geq c_1 \cdot 1^3$ and $T(2) \geq c_1 \cdot 2^3$, handling the base cases.

To show the upper bound on $T(n)$ we follow a similar reasoning as above, adopting the inductive hypothesis $T(n) \leq c_2 n^3$ for all $n \geq n_2$, where $c_2, n_2 > 0$ are constants. At some point we'll have that $T(n) \leq c_2 n^3 + f_{c_2}(n)$. Since

$$\begin{aligned} f_c(n) &< (1-6c)n^2 + 12cn \\ &\leq (1-6c)n^2 + 4cn^2 && \text{(as long as } n \geq 3) \\ &= (1-2c)n^2 \\ &\leq 0 && \text{(as long as } c \geq 1/2), \end{aligned}$$

we conclude that when $c_2 \geq 1/2$ and $n_2 \geq 1$, it holds that $T(n) \leq c_2 n^3$ for all $n \geq n_2 + 2$.

Let's choose $n_2 = 1$. We handle both $T(1) \leq c_2 \cdot 1^3$ and $T(2) \leq c_2 \cdot 2^3$ by choosing $c_2 = \max\{1/2, T(1), T(2)/8\}$, which finishes the inductive proof of the upper bound on $T(n)$.

We've proven that $c_1 n^3 \leq T(n) \leq c_2 n^3$ for all $n \geq 1$, and thus $T(n) = \Theta(n^3)$.

4-2 *Parameter-passing costs*

a. *Work in progress.*

b. *Work in progress.*

c. *Work in progress.*

4-3 *Solving recurrences with a change of variables*

a. Let $m = \lg n$. Then, $n = 2^m$, and thus

$$T(2^m) = 2T(2^{m/2}) + \Theta(m).$$

Now, letting $S(m) = T(2^m)$ produces the new recurrence

$$S(m) = 2S(m/2) + \Theta(m).$$

b. Recurrence $S(m)$ is identical to recurrence (2.3) and very much like the recurrence we solved in Exercise 4.3-1(c), so $S(m) = \Theta(m \lg m)$.

c. Changing back from $S(m)$ to $T(n)$, we obtain $T(n) = T(2^m) = S(m) = \Theta(m \lg m) = \Theta(\lg n \lg \lg n)$.

d. Let's assume that the base case is $T(n) = \Theta(1)$ for $n < 2$ and let c represent the upper-bound constant hidden by the $\Theta(\lg n)$ term for $n \geq 2$. Figure 4-3 shows the recursion tree for the recurrence $T(n) = 2T\left(\sqrt{n}\right) + c \lg n$.

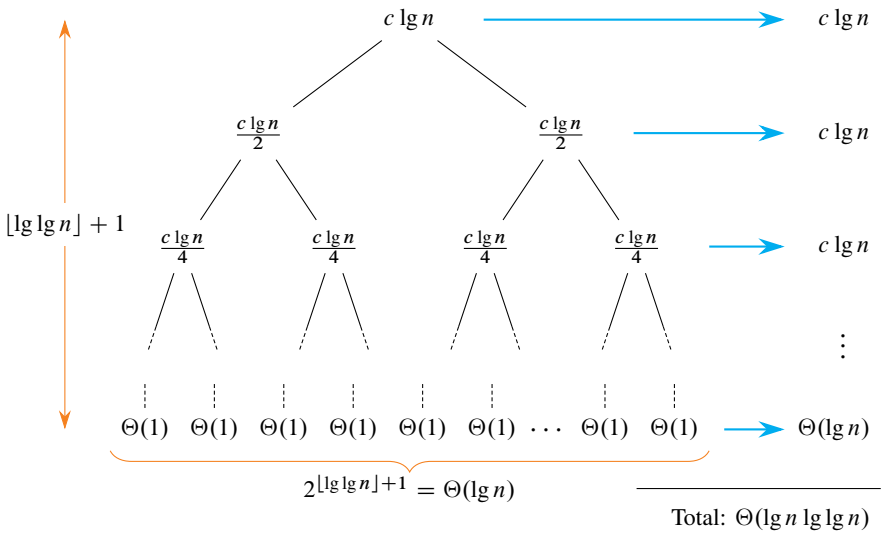


Figure 4-3 A recursion tree for the recurrence $T(n) = 2T\left(\sqrt{n}\right) + c \lg n$.

Let h be the height of the tree. The cost of the root is $c \lg n$. The recurrence breaks down a problem of size n into two subproblems, each of size $\sqrt{n}$, and each incurring the cost of $c \lg \sqrt{n} = c \lg n^{1/2} = (c \lg n)/2$. In general, at depth i, where $i = 0, 1, \ldots, h-1$, there are 2^i subproblems, each of size $n^{1/2^i}$, and the total level cost is $2^i c \lg\left(n^{1/2^i}\right) = (2^i c \lg n)/2^i = c \lg n$. The bottom level contains leaves representing subproblems of size less than 2, so we have $n^{1/2^{h-1}} \geq 2$ and $n^{1/2^h} < 2$. From there we get $\lg \lg n < h \leq \lg \lg n + 1$, or $h = \lfloor \lg \lg n \rfloor + 1$ (by inequality (3.2)), leading to a total leaf cost of $2^h \cdot \Theta(1) = \Theta(\lg n)$.

Adding up the costs of all internal nodes and the costs of all leaves yields

$$\begin{aligned} T(n) &= \sum_{i=0}^{h-1} c \lg n + \Theta(\lg n) \\ &= c \lg n \cdot h + \Theta(\lg n) \\ &= c \lg n (\lfloor \lg \lg n \rfloor + 1) + \Theta(\lg n) \\ &\leq c \lg n (\lg \lg n + 1) + \Theta(\lg n) \\ &= O(\lg n \lg \lg n). \end{aligned}$$

We follow a similar reasoning for c representing the lower-bound constant hidden by the $\Theta(\lg n)$ term in recurrence (4.25), to obtain $T(n) = \Omega(\lg n \lg \lg n)$.

e. For this recurrence we use the same substitution $m = \lg n$ as in part (a), to get

$$T(2^m) = 2T(2^{m/2}) + \Theta(1).$$

Defining $S(m) = T(2^m)$ yields

$$S(m) = 2S(m/2) + \Theta(1),$$

which is identical to recurrence (4.12), and has solution $S(m) = \Theta(m)$. Hence, $T(n) = T(2^m) = S(m) = \Theta(m) = \Theta(\lg n)$.

f. Define $m = \log_3 n$, so that $n = 3^m$. By changing the variable in $T(n)$ we get

$$T(3^m) = 3T(3^{m/3}) + \Theta(3^m),$$

and by substituting $S(m) = T(3^m)$ we get

$$S(m) = 3S(m/3) + \Theta(3^m).$$

$S(m)$ is a master recurrence with $a = b = 3$ and $f(m) = \Theta(3^m)$, that we can solve by the master method. We have $m^{\log_b a} = m$, and $f(m) = \Omega(m^{1+\epsilon})$ for any $\epsilon > 0$. To verify that the regularity condition holds, we need to show that there exists a constant $c < 1$ such that

$$\begin{aligned} af(m/b) &= 3 \cdot 3^{m/3} \\ &= 3^{-2m/3+1} \cdot 3^m \\ &\leq cf(m) \end{aligned}$$

for all sufficiently large m. The quantity $h(m) = 3^{-2m/3+1}$ is an exponential function with the exponent that decreases as m increases, and so $h(m)$ decreases as well. For $m \geq 3$, $h(m) \leq 1/3$, so we can pick $c = 1/3$ to satisfy the above inequality. We then

apply case 3 of the master theorem to get the solution $S(m) = \Theta(3^m)$. Finally, we obtain $T(n) = T(3^m) = S(m) = \Theta(3^m) = \Theta(n)$.

4-4 *More recurrence examples*

a. Using the master method, we have $a = 5$ and $b = 3$, which means that $n^{\log_b a} = n^{\log_3 5} = \Omega(n^{1.46})$. Since $f(n) = n \lg n = O(n^{\log_b a - \epsilon})$, where ϵ can be as large as approximately 0.46, we apply case 1 to get $T(n) = \Theta(n^{\log_3 5})$.

b. It turns out that neither the master method, nor the Akra-Bazzi method apply here. The former fails due to the fact that the driving function does not grow polynomially slower than the watershed function, as explained in the book on page 105 for a similar recurrence. And the latter fails because the integral $\int_1^n \frac{dx}{x \lg x}$ used in equation (4.23) for this recurrence does not converge.

We'll use a different approach for $T(n)$, instead considering a family of recurrences $T_b(n) = bT_b(n/b) + n/\lg n$ for a constant $b > 1$, where $T(n)$ belongs. Let's use the change-of-variables method from Problem 4-3, letting $m = \log_b n$. Then, $n = b^m$, and we obtain

$$T_b(b^m) = bT_b(b^{m-1}) + \frac{b^m}{m \lg b}.$$

Defining $S_b(m) = T_b(b^m)$ produces the new recurrence

$$S_b(m) = bS_b(m-1) + \frac{b^m}{m \lg b}.$$

Assume that $T_b(n) = \Theta(1)$ for $n < b$, which means that $S_b(m) = \Theta(1)$ for $m < 1$. We'll show by induction on $m \geq 1$ that $S_b(m) = b^m(H_m + \Theta(1))/\lg b$, where H_m is the mth harmonic number. For $m = 1$,

$$\begin{aligned} S_b(1) &= bS_b(0) + \frac{b}{\lg b} \\ &= b \cdot \Theta(1) + \frac{bH_1}{\lg b} \\ &= \frac{b^1(H_1 + \Theta(1))}{\lg b} \qquad \text{(because } \lg b \cdot \Theta(1) = \Theta(1)\text{)}, \end{aligned}$$

so the base case holds. For the inductive step, let $m \geq 2$, and adopt the inductive

hypothesis $S_b(m-1) = b^{m-1}(H_{m-1} + \Theta(1))/\lg b$ to get

$$\begin{aligned}
S_b(m) &= b \cdot \frac{b^{m-1}(H_{m-1} + \Theta(1))}{\lg b} + \frac{b^m}{m \lg b} \\
&= \frac{b^m(H_{m-1} + \Theta(1)) + b^m/m}{\lg b} \\
&= \frac{b^m(H_m + \Theta(1))}{\lg b} && \text{(because } \Theta(1) + 1/m = \Theta(1)\text{)} \\
&= \frac{b^m \ln m + \Theta(1)}{\lg b} && \text{(by equation (A.9), and because } b^m \cdot \Theta(1) = \Theta(1)\text{)} \\
&= \Theta(b^m \lg m).
\end{aligned}$$

Changing back from $S_b(m)$ to $T_b(n)$, we obtain $T_b(n) = T_b(b^m) = S_b(m) = \Theta(b^m \lg m) = \Theta(n \lg \log_b n) = \Theta(n \lg \lg n)$.

As we can see, the asymptotic solution to $T_b(n)$ doesn't depend on b, so it also applies to recurrence $T(n) \equiv T_3(n)$ that we study in this part.

c. Using the master method, we have $a = 8$ and $b = 2$, which means that $n^{\log_b a} = n^{\log_2 8} = n^3$, and $f(n) = n^3\sqrt{n} = n^{7/2} = \Omega(n^{\log_b a + \epsilon})$ for $\epsilon \le 1/2$. Also, it holds that $8f(n/2) = 8(n/2)^{7/2} = 8n^{7/2}/(2^3\sqrt{2}) \le cf(n)$ for any $c \ge 1/\sqrt{2}$, satisfying the regularity condition. By case 3, the solution to the recurrence is $T(n) = \Theta(n^{7/2})$.

d. Observe that the function $f(n) = n/2$ satisfies the polynomial-growth condition, because for any ϕ and ψ, where $1 \le \psi \le \phi$, it's true that $1/\phi \le \psi$, so we can let $d = \phi + 1 > 1$ to get $n/2d < n/2\phi \le \psi n/2 \le \phi n/2 < dn/2$ for any n.

Now let $h(n) = -2$. Then $|h(n)| = O(n/\lg^{1+\epsilon} n)$ for any $\epsilon > 0$, so by the fact stated in the book on page 117, we can replace $T(n/2 + h(n))$ by $T(n/2)$ in the definition of $T(n)$ to obtain a recurrence with the same asymptotic solution.

Consider $T'(n) = 2T'(n/2) + n/2$. Using the master method, we have $a = b = 2$, and $f(n) = n/2 = \Theta(n) = \Theta(n^{\log_b a})$, so applying case 2 of the master theorem gives solution $T'(n) = \Theta(n \lg n)$. Therefore, $T(n) = \Theta(n \lg n)$ as well.

e. We are dealing with recurrence $T_2(n)$ from the family of recurrences $T_b(n)$ we studied in part (b). Based on that analysis, we conclude that $T(n) = \Theta(n \lg \lg n)$.

f. This is an Akra-Bazzi recurrence with $a_1 = a_2 = a_3 = 1, b_1 = 2, b_2 = 4, b_3 = 8$, and $f(n) = n$. Let p be a unique real number such that

$$\left(\frac{1}{2}\right)^p + \left(\frac{1}{4}\right)^p + \left(\frac{1}{8}\right)^p = 1.$$

Since $(1/2)^0 + (1/4)^0 + (1/8)^0 > 1$ and $(1/2)^1 + (1/4)^1 + (1/8)^1 < 1$, p lies in the range $0 < p < 1$, as for recurrence (4.24). Also, both recurrences share the same driving function $f(n) = n$. This means that the Akra-Bazzi method applied to the studied recurrence gives the same solution as for recurrence (4.24): $T(n) = \Theta(n)$.

g. Let $n_0 > 0$ be the implicit threshold constant, and let $d > 0$ be another constant such that $T(n) \le d$ for all $n < n_0$. We'll prove by induction that $H_n - H_{n_0} < T(n) < H_n + d$ for all $n > 0$.

First, let $0 < n < n_0$. Since the harmonic series strictly increases and consists of positive terms, $H_n - H_{n_0} < 0 \le T(n) \le d < H_n + d$.

For the inductive case let $n \ge n_0$ and suppose that

$$H_{n-1} - H_{n_0} < T(n-1) < H_{n-1} + d.$$

We have

$$\begin{aligned} T(n) &= T(n-1) + 1/n \\ &> H_{n-1} - H_{n_0} + 1/n \\ &= H_n - H_{n_0}, \end{aligned}$$

and

$$\begin{aligned} T(n) &= T(n-1) + 1/n \\ &< H_{n-1} + d + 1/n \\ &= H_n + d, \end{aligned}$$

thereby completing the inductive proof.

By the bounds shown above we obtain

$$\begin{aligned} T(n) &= \Theta(H_n) \\ &= \Theta(\ln n + O(1)) && \text{(by equation (A.9))} \\ &= \Theta(\lg n). \end{aligned}$$

h. Let $n_0 > 0$ be the implicit threshold constant, and let $d > 0$ be another constant such that $T(n) \le d$ for all $n < n_0$. We'll prove by induction that

$$\lg(n!) - \lg(n_0!) < T(n) \le \lg(n!) + d$$

for all $n > 0$.

Let $0 < n < n_0$. It follows that $0 \le \lg(n!) < \lg(n_0!)$, and thus $\lg(n!) - \lg(n_0!) < 0 \le T(n) \le d \le \lg(n!) + d$.

For the inductive case let $n \ge n_0$ and suppose that

$$\lg((n-1)!) - \lg(n_0!) < T(n-1) \le \lg((n-1)!) + d.$$

We have

$$\begin{aligned} T(n) &= T(n-1) + \lg n \\ &> \lg((n-1)!) - \lg(n_0!) + \lg n \\ &= \lg((n-1)! \cdot n) - \lg(n_0!) \\ &= \lg(n!) - \lg(n_0!) \end{aligned}$$

and

$$\begin{aligned} T(n) &= T(n-1) + \lg n \\ &\le \lg((n-1)!) + d + \lg n \\ &= \lg((n-1)! \cdot n) + d \\ &= \lg(n!) + d, \end{aligned}$$

thereby completing the inductive proof.

By the bounds shown above and by equation (3.28), we obtain $T(n) = \Theta(\lg(n!)) = \Theta(n \lg n)$.

i. *Work in progress.*

j. Dividing $T(n)$ by n yields

$$\frac{T(n)}{n} = \frac{T\left(\sqrt{n}\right)}{\sqrt{n}} + 1,$$

and substituting $S(n) = T(n)/n$ yields

$$S(n) = S\left(\sqrt{n}\right) + 1.$$

If we now define $m = \lg n$ and substitute $R(m) = S(2^m)$, we get

$$R(m) = R(m/2) + 1.$$

Recurrence $R(m)$ is a special case of the recurrence describing the running time of binary search, and we've already solved it in Exercise 4.5-3, so $R(m) = \Theta(\lg m)$. Going back to the original recurrence, we finally obtain

$$T(n) = nS(n) = nR(\lg n) = n \cdot \Theta(\lg \lg n) = \Theta(n \lg \lg n).$$

4-5 *Fibonacci numbers*

a.

$$\begin{aligned}
\mathcal{F}(z) &= \sum_{i=0}^{\infty} F_i z^i \\
&= F_0 + zF_1 + \sum_{i=2}^{\infty} (F_{i-1} + F_{i-2}) z^i && \text{(by recurrence (3.31))} \\
&= z + \sum_{i=2}^{\infty} F_{i-1} z^i + \sum_{i=2}^{\infty} F_{i-2} z^i \\
&= z + \sum_{i=1}^{\infty} F_i z^{i+1} + \sum_{i=0}^{\infty} F_i z^{i+2} && \text{(reindexing)} \\
&= z + z\mathcal{F}(z) + z^2\mathcal{F}(z)
\end{aligned}$$

b. The first equality follows by the identity from part (a):

$$\begin{aligned}
\mathcal{F}(z) &= z + z\mathcal{F}(z) + z^2\mathcal{F}(z), \\
(1 - z - z^2)\mathcal{F}(z) &= z, \\
\mathcal{F}(z) &= \frac{z}{1 - z - z^2}.
\end{aligned}$$

The denominator of the last fraction is a quadratic function that zeroes if $z = -\phi$ or $z = -\widehat{\phi}$, so we can express it in the factored form:

$$\begin{aligned}
1 - z - z^2 &= -(z + \phi)(z + \widehat{\phi}) \\
&= -\phi\widehat{\phi} - \phi z - \widehat{\phi} z - z^2 \\
&= 1 - \phi z - \widehat{\phi} z + \phi\widehat{\phi} z^2 && \text{(since } \phi\widehat{\phi} = -1\text{)} \\
&= (1 - \phi z)(1 - \widehat{\phi} z),
\end{aligned}$$

thereby proving the second equality. We get the last equality, by observing that

$$\begin{aligned}
\frac{1}{1 - \phi z} - \frac{1}{1 - \widehat{\phi} z} &= \frac{1 - \widehat{\phi} z - 1 + \phi z}{(1 - \phi z)(1 - \widehat{\phi} z)} \\
&= \frac{z(\phi - \widehat{\phi})}{(1 - \phi z)(1 - \widehat{\phi} z)} \\
&= \frac{z\sqrt{5}}{(1 - \phi z)(1 - \widehat{\phi} z)} && \text{(since } \phi - \widehat{\phi} = \sqrt{5}\text{)},
\end{aligned}$$

and therefore,

$$\begin{aligned}\frac{z}{(1-\phi z)(1-\widehat{\phi} z)} &= \frac{1}{\sqrt{5}} \cdot \frac{z\sqrt{5}}{(1-\phi z)(1-\widehat{\phi} z)} \\ &= \frac{1}{\sqrt{5}}\left(\frac{1}{1-\phi z} - \frac{1}{1-\widehat{\phi} z}\right).\end{aligned}$$

c. We'll prove the formula in two ways. In one approach we follow the hint to get

$$\begin{aligned}\mathcal{F}(z) &= \frac{1}{\sqrt{5}}\left(\frac{1}{1-\phi z} - \frac{1}{1-\widehat{\phi} z}\right) \\ &= \frac{1}{\sqrt{5}}\left(\sum_{i=0}^{\infty}(\phi z)^i - \sum_{i=0}^{\infty}(\widehat{\phi} z)^i\right) \quad \begin{array}{l}\text{(by the generating-function version} \\ \text{of equation (A.7) for } x = \phi z \text{ and } x = \widehat{\phi} z)\end{array} \\ &= \sum_{i=0}^{\infty}\frac{1}{\sqrt{5}}(\phi^i - \widehat{\phi}^i) z^i .\end{aligned}$$

In another approach we can obtain the equality immediately, if we substitute $F_i = \left(\phi^i - \widehat{\phi}^i\right)/\sqrt{5}$ (as shown in Exercise 3.3-8) in the definition of $\mathcal{F}(z)$.

d. Since $\left|\widehat{\phi}\right| < 1$, it holds that $\left|\widehat{\phi}^i\right| < 1$ for $i > 0$, and $\left|\widehat{\phi}^i\right|/\sqrt{5} < 1/\sqrt{5} < 1/2$. Then,

$$\begin{aligned}F_i &= \frac{1}{\sqrt{5}}(\phi^i - \widehat{\phi}^i) && \text{(by part (c))} \\ &= \frac{\phi^i}{\sqrt{5}} - \frac{\widehat{\phi}^i}{\sqrt{5}},\end{aligned}$$

from where

$$\frac{\phi^i}{\sqrt{5}} - \frac{1}{2} < F_i < \frac{\phi^i}{\sqrt{5}} + \frac{1}{2}.$$

Those bounds on F_i differ exactly by 1, so there is a unique integer between them, close to $\phi^i/\sqrt{5}$. By the above inequalities, that integer has to be F_i.

e. Let's evaluate the difference

$$\begin{aligned} F_{i+2} - \phi^i &= \frac{\phi^{i+2} - \widehat{\phi}^{i+2}}{\sqrt{5}} - \phi^i && \text{(by Exercise 3.3-8)} \\ &= \frac{\phi^i(\phi^2 - \sqrt{5}) - \widehat{\phi}^{i+2}}{\sqrt{5}} \\ &= \frac{\phi^i \cdot \frac{3-\sqrt{5}}{2} - \widehat{\phi}^i \cdot \frac{3-\sqrt{5}}{2}}{\sqrt{5}} \\ &= \frac{3-\sqrt{5}}{2\sqrt{5}} \left(\phi^i - \widehat{\phi}^i\right). \end{aligned}$$

Since $\phi > |\widehat{\phi}|$, the last expression is nonnegative for all $i \geq 0$, and so $F_{i+2} \geq \phi^i$ for $i \geq 0$.

4-6 Chip testing

a. Let G be the set of all good chips and let B be the set of all bad chips. Good chips report that any chip in G is good, and that any chip in B is bad. On the other hand, bad chips can follow the strategy to mirror the good chips behavior — they can consistently report that any chip in B is good, and that any chip in G is bad. This means that relying on pairwise tests only — with no additional assumptions — the professor wouldn't be able to distinguish G from B.

b. We can view the result of each pairwise test as an ordered pair from the Cartesian product {"good", "bad"} × {"good", "bad"}. The first element of such a pair is the verdict of the first chip about the second chip, and the second element is the verdict of the second chip about the first chip.

Let's pair up the n chips and test each of the $\lfloor n/2 \rfloor$ pairs. Note that if a test outcome is other than ("good", "good"), we can discard both chips in that pair, because at least one of these chips is bad and among the remaining chips there will still be more good chips than bad chips. The pairs that report ("good", "good") consist of either both good or both bad chips, so we can safely discard one chip from each such pair.

Now we need to figure out, whether we should keep or eliminate the unpaired chip u that wasn't tested in this round, in case that n is odd. As it turns out, this depends on the number m of the ("good", "good") outcomes we've seen. If m is odd, among those m pairs that reported this outcome there were more consisting of good chips than of bad chips, so the set of the m kept chips already has the required property, and we discard u. If m is even, the number of pairs of good chips could be the same as the number of pairs of bad chips, in which case if we eliminated u, we would end up with a

set of exactly half of good chips. By the assumption about the amount of good chips in the initial set, if among the m pairs that reported ("good", "good") exactly half of them consisted of good chips, u has to be good. Therefore, we must keep u.

As m can be at most $\lfloor n/2 \rfloor$, we've shown how to reduce the set of size n to a set of size at most $m = \lfloor n/2 \rfloor = \lceil n/2 \rceil$ for even n, and at most $m + 1 = \lfloor n/2 \rfloor + 1 = \lceil n/2 \rceil$ for odd n.

c. By recursively carrying out the process described in part (b), we maintain the invariant about the set of remaining chips, while reducing the size of that set. We can finish that process once we obtain a set of size at most 2, which — by the invariant — contains only good chips.

The number of pairwise tests needed to find one good chip in the worst case (i.e., when every test results in ("good", "good")) is given by the recurrence

$$T(n) = T(\lceil n/2 \rceil) + \lfloor n/2 \rfloor .$$

We can solve it using the master method, after ignoring the ceilings in the argument to T on the right-hand side. We have $a = 1$, $b = 2$, and $f(n) = \lfloor n/2 \rfloor$. Since $f(n) = \Omega(n^{\epsilon}) = \Omega(n^{\log_b a + \epsilon})$ for any $\epsilon \le 1$, and

$$\begin{aligned} a f(n/b) &= \lfloor n/4 \rfloor \\ &= \lfloor \lfloor n/2 \rfloor / 2 \rfloor && \text{(by equation (3.6))} \\ &\le \lfloor n/2 \rfloor / 2 \\ &\le c f(n) \end{aligned}$$

for any $c \ge 1/2$, case 3 of the master theorem applies, and the solution is $T(n) = \Theta(n)$.

d. Now that we've identified a good chip c_0, we'll use it to identify the remaining ones. Let's run $n - 1$ pairwise tests with c_0 and each of the other chips. If we get ("good", "good") then both tested chips are good, and so we've identified a new good chip. Obtaining any other outcome means that at least one of the tested chips is bad, so it has to be the other one, since c_0 is good. It follows that after performing all those additional $\Theta(n)$ tests, we'll identify all the remaining good chips.

4-7 *Monge arrays*

a. The elements of a Monge array A satisfy the inequality

$$A[i, j] + A[k, l] \le A[i, l] + A[k, j],$$

where $1 \le i < k \le m$ and $1 \le j < l \le n$. In particular, it can be $k = i + 1$ and $l = j + 1$, and thus the "only if" part holds.

Now suppose that

$$A[i, j] + A[i + 1, j + 1] \leq A[i, j + 1] + A[i + 1, j] \tag{1}$$

for all $i = 1, 2, \ldots, m - 1$ and all $j = 1, 2, \ldots, n - 1$. Fix i and j, and let $r \geq 0$. We'll show by induction on r that

$$A[i, j] + A[i + r, j + 1] \leq A[i, j + 1] + A[i + r, j]. \tag{2}$$

When $r = 0$, the inequality holds trivially, establishing the base case. Now let $r \geq 1$ and suppose that (2) holds for $r - 1$. By the inductive hypothesis and by (1), we have

$$\begin{aligned} A[i, j] + A[i + r, j + 1] &\leq A[i, j + 1] + A[i + r - 1, j] - A[i + r - 1, j + 1] \\ &\quad + A[i + r - 1, j + 1] + A[i + r, j] - A[i + r - 1, j] \\ &= A[i, j + 1] + A[i + r, j]. \end{aligned}$$

Next we induct on l to show that

$$A[i, j] + A[k, l] \leq A[i, l] + A[k, j] \tag{3}$$

holds for all i, j, k, and l such that $1 \leq i < k \leq m$ and $1 \leq j < l \leq n$. Let's fix i, j, and k. When $l = j + 1$ the inequality reduces to (2) for $r = k - i$, so let $l > j + 1$. Suppose that (3) holds for $l - 1$. Then we have

$$\begin{aligned} A[i, j] + A[k, l] &\leq (A[i, l - 1] + A[k, j] - A[k, l - 1]) + A[k, l] \\ &\leq (A[i, l - 1] + A[k, j] - A[k, l - 1]) \\ &\quad + (A[k - 1, l] + A[k, l - 1] - A[k - 1, l - 1]) && \text{(by (1))} \\ &= (A[i, l - 1] + A[k - 1, l]) + A[k, j] - A[k - 1, l - 1] \\ &\leq A[i, l] + A[k - 1, l - 1] + A[k, j] - A[k - 1, l - 1] && \text{(by (2))} \\ &= A[i, l] + A[k, j]. \end{aligned}$$

b. From part (a) it follows that the Monge array condition isn't satisfied, because $A[1, 2] + A[2, 3] \leq A[1, 3] + A[2, 2]$ doesn't hold. To restore the condition, we can set $A[1, 3]$ to any value between 24 and 29.

c. The proof is by contradiction. Suppose that in a Monge array A there exists $1 \leq i < m$ such that $f(i) > f(i + 1)$. By the definition of f,

$$A[i, f(i + 1)] > A[i, f(i)]$$

and

$$A[i + 1, f(i)] \geq A[i + 1, f(i + 1)],$$

and thus

$$A[i, f(i+1)] + A[i+1, f(i)] > A[i, f(i)] + A[i+1, f(i+1)].$$

However, the above inequality contradicts the condition from the Monge array definition, for $j = f(i+1)$, $k = i+1$, and $l = f(i)$.

d. To determine the leftmost minimum of row i, for an odd i, we need the locations of the leftmost minimums of the adjacent rows, $i-1$ and $i+1$. The algorithm has already computed these values. From part (c), $f(i-1) \le f(i) \le f(i+1)$, so we only need to look through the subarray $A[i, f(i-1) : f(i+1)]$ to find its minimum. We can define $f(0) = 1$ and $f(m+1) = n$ to simplify the procedure when processing the first row, or the last row in case m is odd.

When looking for the leftmost minimum element of row i, we examine exactly $f(i+1) - f(i-1) + 1$ cells, so the total number of examined cells in the worst case is

$$\begin{aligned}
\sum_{\substack{1 \le i \le m \\ i \text{ is odd}}} (f(i+1) - f(i-1) + 1) &= \sum_{k=0}^{\lceil m/2 \rceil - 1} (f(2k+2) - f(2k) + 1) \\
&= \lceil m/2 \rceil + \sum_{k=0}^{\lceil m/2 \rceil - 1} (f(2k+2) - f(2k)) \\
&= \lceil m/2 \rceil + f(2\lceil m/2 \rceil) - f(0) \qquad \text{(because the sum telescopes)} \\
&\le \lceil m/2 \rceil + n - 1 \\
&= O(m+n).
\end{aligned}$$

e. Let $T(m, n)$ be the worst-case running time of the algorithm for an $m \times n$ Monge array. Computing the leftmost minimum in a Monge array A with just one row requires checking $O(n)$ cells, so $T(1, n) = O(n)$.

If A has more than one row, we construct a submatrix A' consisting of the even-numbered rows of A. Instead of copying the contents of these rows to a newly allocated storage, we can follow a similar approach as when partitioning matrices in MATRIX-MULTIPLY-RECURSIVE — index calculations. This way we can keep the same elements for both matrices and only involve arithmetic when accessing the even-numbered rows of A. In fact, any method of constructing A' that takes time at most proportional to the number of rows or the number of columns of A won't affect the asymptotic upper bound of $T(m, n)$.

The conquer step involves running the algorithm recursively on a $\lfloor m/2 \rfloor \times n$ Monge array A', taking $T(\lfloor m/2 \rfloor, n)$ time. Finally, by part (d), the combine step requires

checking $O(m+n)$ cells. Therefore, we obtain the following recurrence:

$$T(m,n) = \begin{cases} O(n), & \text{if } m = 1, \\ T(\lfloor m/2 \rfloor, n) + O(m+n), & \text{if } m > 1. \end{cases}$$

Let $c > 0$ represent larger of the upper-bound constants, and let n_0 be the larger of the threshold constants hidden by the O-notations above. Let $n \geq n_0$. We'll use the substitution method on m to show that $T(m,n) \leq c(2m + n \lg(2m))$. For $m = 1$,

$$\begin{aligned} T(1,n) &\leq cn \\ &< c(2 + n \lg 2), \end{aligned}$$

so the base step holds. Let $m \geq 2$ and let's adopt the inductive hypothesis that $T(\lfloor m/2 \rfloor, n) \leq c(2\lfloor m/2 \rfloor + n \lg(2\lfloor m/2 \rfloor))$. We have

$$\begin{aligned} T(m,n) &\leq c(2\lfloor m/2 \rfloor + n \lg(2\lfloor m/2 \rfloor)) + c(m+n) \\ &\leq c(m + n \lg m) + c(m+n) \\ &= c(2m + n(\lg m + 1)) \\ &= c(2m + n \lg(2m)), \end{aligned}$$

which handles the inductive step. Hence, $T(n) = O(m + n \lg m)$.

5 Probabilistic Analysis and Randomized Algorithms

5.1 The hiring problem

5.1-1 Let $\prec$ be the relation on the set of the ranks of the candidates, which we use in line 4 of procedure HIRE-ASSISTANT to determine which candidate is best. In other words, if $rank(i) \prec rank(j)$, then candidate j is better qualified than candidate i. Let $\preceq$ be the ***weak variant*** of relation $\prec$: $rank(i) \preceq rank(j)$ whenever $rank(i) \prec rank(j)$ or $rank(i) = rank(j)$. We'll show that $\preceq$ is a total order on the ranks of the candidates.

By definition, it immediately follows that relation $\preceq$ is reflexive.

As any permutation of the candidates may appear as input to HIRE-ASSISTANT, we should be able to compare any pair of different candidates using $\prec$, which proves that $\preceq$ is a total relation.

Suppose that $\prec$ is not antisymmetric, and let $i \neq j$ be two candidates for which $rank(i) \prec rank(j)$ and $rank(j) \prec rank(i)$. Then, depending on which of the two candidates i and j will appear right after the other, the latter will be hired, which contradicts the assumption that we are always able to determine which candidate is best. Therefore, $\prec$ is antisymmetric, and so is $\preceq$.

Finally, suppose that $\prec$ is not transitive, and let i, j, and k be candidates for which $rank(i) \prec rank(j)$, $rank(j) \prec rank(k)$, and $rank(k) \prec rank(i)$. If the algorithm received candidate i, then j, then k, it will hire k, but if it received candidate i, then k, then j, it will hire j — a contradiction. Therefore, $\prec$ is transitive, and so is $\preceq$.

5.1-2 ★

```
RANDOM(a, b)
1  while a < b
2      mid = ⌊(a + b)/2⌋
3      if RANDOM(0, 1) == 0
4          a = mid + 1
5      else b = mid
6  return a
```

The procedure generates a random integer from the range $[a, b]$. Each iteration of the **while** loop splits this range approximately in half and randomly discards one part of the split, until the range becomes a single point. Observe that the problem is reduced similarly as in binary search. If we assume that each call to RANDOM$(0, 1)$ in line 3 takes constant time, we have that the expected running time of RANDOM(a, b) and the worst-case running time $T(n)$ of BINARY-SEARCH ran on an array of size $n = b - a + 1$ match asymptotically. According to Exercise 4.5-3, $T(n) = \Theta(\lg n)$, so the expected running time of RANDOM(a, b) is $\Theta(\lg(b - a))$.

5.1-3 ★ Note that the event of getting first 0 and then 1, and the event of getting first 1 and then 0 in two subsequent calls to BIASED-RANDOM have the same probability of $p(1 - p)$. We'll call BIASED-RANDOM twice until obtaining different results, and once that happens we'll return one of those results.

The following algorithm implements the described process:

```
UNBIASED-RANDOM
1  repeat
2      x = BIASED-RANDOM
3      y = BIASED-RANDOM
4  until x ≠ y
5  return x
```

Let's assume that each call to BIASED-RANDOM takes constant time, which means that each iteration of the **repeat** loop also takes constant time. Subsequent iterations create a sequence of Bernoulli trials, where success means obtaining different results from both calls to BIASED-RANDOM (i.e., the condition from line 4), that occurs with probability $2p(1 - p)$. The expected number of trials before obtaining a success is given by (C.36) and equals $1/(2p(1 - p))$. Hence, we conclude that the expected running time of the algorithm is $\Theta(1/(p(1 - p)))$.

5.2 Indicator random variables

5.2-1 Note that the first candidate is hired in any call to HIRE-ASSISTANT, so hiring exactly one time happens when no other candidate is hired. In this case, candidate 1 is the most qualified in the set of all n candidates (i.e., $rank(1) = n$). The candidate ranked highest can appear at any of the n positions in the input sequence, so the probability that they will occupy the first position is $1/n$.

To hire all n candidates, we must interview them in ascending rank order. There is only one such input permutation, so the probability of this event is $1/n!$.

5.2-2 Note that we always hire the first candidate that appears at input, and the one with the highest rank. If the HIRE-ASSISTANT procedure is to hire exactly twice, then candidate 1 should have rank $i \leq n-1$, and candidates with ranks $i+1, i+2, \ldots, n-1$ should appear after the candidate with rank n.

Denote by E_i the event that $rank(1) = i$. Of course, $\Pr\{E_i\} = 1/n$ for every $i = 1, 2, \ldots, n$. Let b be the position of the highest qualified candidate in the input sequence, and let F be the event that $rank(1) > rank(j)$ for all $j = 2, 3, \ldots, b-1$. If E_i occurs, then F occurs only if $i \neq n$, and among the $n-i$ candidates whose ranks are greater than i, the one with the rank of n is interviewed earliest. Hence we have $\Pr\{F \mid E_i\} = 1/(n-i)$, if $i \neq n$. Finally, let A denote the event that exactly two applicants are hired in HIRE-ASSISTANT. We have

$$\begin{aligned} A &= F \cap (E_1 \cup E_2 \cup \cdots \cup E_{n-1}) \\ &= (F \cap E_1) \cup (F \cap E_2) \cup \cdots \cup (F \cap E_{n-1}), \end{aligned}$$

and since the events $E_1, E_2, \ldots, E_n$ are mutually exclusive,

$$\Pr\{A\} = \sum_{i=1}^{n-1} \Pr\{F \cap E_i\}.$$

By the definition (C.16) of conditional probability,

$$\begin{aligned} \Pr\{F \cap E_i\} &= \Pr\{F \mid E_i\} \Pr\{E_i\} \\ &= \frac{1}{n-i} \cdot \frac{1}{n}, \end{aligned}$$

and therefore

$$\begin{aligned}\Pr\{A\} &= \sum_{i=1}^{n-1} \frac{1}{n-i} \cdot \frac{1}{n} \\ &= \frac{1}{n} \sum_{i=1}^{n-1} \frac{1}{n-i} \\ &= \frac{1}{n} \sum_{i=1}^{n-1} \frac{1}{i} \\ &= \frac{H_{n-1}}{n}.\end{aligned}$$

5.2-3 First we'll calculate the expected value of one die. For $j = 1, 2, \ldots, 6$, let X_j be the indicator random variable of a die coming up j. Assuming that each die is fair, we have $\Pr\{X_j = 1\} = 1/6$ for each $j = 1, 2, \ldots, 6$. Let X be the random variable of the value of a single die roll. Then,

$$\begin{aligned}\mathrm{E}[X] &= \sum_{j=1}^{6} j \Pr\{X = j\} \\ &= \sum_{j=1}^{6} j \Pr\{X_j = 1\} \\ &= (1/6) \cdot (1 + 2 + 3 + 4 + 5 + 6) \\ &= 21/6 \\ &= 3.5.\end{aligned}$$

The expected sum of the values of n dice is therefore

$$\begin{aligned}\mathrm{E}[nX] &= n\,\mathrm{E}[X] && \text{(by equation (C.25))} \\ &= 3.5n.\end{aligned}$$

5.2-4 Let X_1 and X_2 be the random variables representing the values on each die. Both variables have the same probability distribution, so their expectations are identical. We

have

$$\begin{aligned}
\mathrm{E}[X_1] &= \sum_{x=1}^{6} x \Pr\{X_1 = x\} \\
&= (1/6) \cdot (1 + 2 + 3 + 4 + 5 + 6) \\
&= 21/6 \\
&= 3.5.
\end{aligned}$$

Therefore, the expected value of the sum of two 6-sided dice is

$$\begin{aligned}
\mathrm{E}[X_1 + X_2] &= \mathrm{E}[X_1] + \mathrm{E}[X_2] \qquad \text{(by equation (C.24), linearity of expectation)} \\
&= 3.5 + 3.5 \\
&= 7.
\end{aligned}$$

For the second scenario, consider the random variables Y_1 and Y_2 representing the values of the first die, and the second die, respectively. Note that Y_1 has the same probability distribution as X_1 (and X_2) from the first scenario, so $\mathrm{E}[Y_1] = \mathrm{E}[X_1] = 3.5$. On the other hand, Y_2 is dependent on Y_1, taking the identical values as Y_1, so $Y_2 = Y_1$. Therefore,

$$\begin{aligned}
\mathrm{E}[Y_1 + Y_2] &= \mathrm{E}[2Y_1] \\
&= 2\,\mathrm{E}[Y_1] \qquad \text{(by equation (C.25))} \\
&= 2 \cdot 3.5 \\
&= 7.
\end{aligned}$$

Let Z_1 and Z_2 be the random variables indicating the values of both dice in the third scenario. Note that $Z_2 = 7 - Z_1$, so

$$\begin{aligned}
\mathrm{E}[Z_1 + Z_2] &= \mathrm{E}[Z_1 + 7 - Z_1] \\
&= \mathrm{E}[7] \\
&= 7.
\end{aligned}$$

5.2-5 Let X_i, for $i = 1, 2, \ldots, n$, be the indicator random variable associated with the event in which the ith customer received his or her hat. Also, let $X = X_1 + X_2 + \cdots + X_n$, so X represents the number of customers who got back their own hats. The hat-check person distributes the n hats to all n customers at random, so the chances that a particular

customer will get their own hat are $1/n$. We have

$$\begin{aligned}
\mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n} X_i\right] \\
&= \sum_{i=1}^{n} \mathrm{E}[X_i] && \text{(by equation (C.24))} \\
&= \sum_{i=1}^{n} \Pr\{\text{customer } i \text{ got back their own hat}\} && \text{(by Lemma 5.1)} \\
&= \sum_{i=1}^{n} \frac{1}{n} \\
&= 1,
\end{aligned}$$

so on average only one customer will get their hat back.

5.2-6 Let X_{ij}, for $1 \le i < j \le n$, be an indicator random variable associated with the event that the array $A[1:n]$ has inversion (i, j). For any pair (i, j), where $1 \le i < j \le n$, there are equal chances for $A[i] < A[j]$ and for $A[i] > A[j]$, so

$$\Pr\{\text{pair } (i, j) \text{ is an inversion of } A\} = 1/2. \tag{4}$$

Consider the random variable $X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$ representing the total number of inversions of A. We have

$$\begin{aligned}
\mathrm{E}[X] &= \mathrm{E}\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathrm{E}\left[X_{ij}\right] && \text{(by equation (C.24))} \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{\text{pair } (i, j) \text{ is an inversion of } A\} && \text{(by Lemma 5.1)} \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} && \text{(by equation (4))} \\
&= \frac{1}{2} \sum_{i=1}^{n-1} (n - i) \\
&= \frac{1}{2} \sum_{i=1}^{n-1} i \\
&= \frac{n(n-1)}{4} \\
&= \frac{\binom{n}{2}}{2},
\end{aligned}$$

which means that on average half of all pairs (i, j), where $1 \le i < j \le n$, are inversions of A.

5.3 Randomized algorithms

5.3-1 Below is the modified procedure RANDOMLY-PERMUTE:

```
RANDOMLY-PERMUTE′(A, n)
1  swap A[1] with A[RANDOM(1, n)]
2  for i = 2 to n
3      swap A[i] with A[RANDOM(i, n)]
```

In Lemma 5.4, the loop invariant remains unchanged (except for the numbers of lines of code containing the **for** loop), and the only change is required in the proof of the invariant initial state.

Initialization: Consider the situation just before the first loop iteration, so that $i = 2$. The loop invariant says that for each possible 1-permutation, the subarray $A[1:1]$ contains this 1-permutation with probability $(n-1)!/n! = 1/n$. The subarray $A[1:1]$ consists of just one element $A[1]$, which has already been exchanged with a randomly chosen element of A in line 1. Thus, $A[1:1]$ contains any 1-permutation with probability $1/n$, and the loop invariant holds prior to the first iteration.

5.3-2 This algorithm does not work as intended. If $n > 1$, it generates permutations in which each position contains an element different from the one that initially occupied that position. For instance, for the input array $A = \langle 1, 2, 3\rangle$, the algorithm is capable of producing one of two permutations: $\langle 2, 1, 3\rangle$ and $\langle 3, 1, 2\rangle$.

5.3-3 Note that the sequence of n calls to the random number generator in PERMUTE-WITH-ALL can generate n^n possible sequences of positions of A, while the procedure can produce one of the $n!$ possible permutations of A. Suppose that $n > 2$ and that the procedure generates each permutation with equal probability. Therefore, each output permutation corresponds to the same number c of index sequences, i.e., $n^n = cn!$. In this equation, $n-1$ divides the right-hand side, so it should also divide the left-hand side. By equation (A.6) for $x = n$ we have

$$\sum_{k=0}^{n-1} n^k = \frac{n^n - 1}{n-1},$$

which gives

$$n^n = (n-1)\sum_{k=0}^{n-1} n^k + 1.$$

This means that the remainder of n^n when divided by $n-1$ is 1 — a contradiction.

Therefore, we conclude that the procedure PERMUTE-WITH-ALL can't produce uniform random permutations.

5.3-4 Let $1 \le i, j \le n$ be two indices. The element $A[i]$ winds up in $B[j]$ when $dest = j$. If $i + offset \le n$, then $dest = i + offset$ can take any value between $i+1$ and n. If, on the other hand, $i + offset > n$, then $dest = i + offset - n$ can take any value between $(n+1) - n = 1$ and $(i+n) - n = i$. Therefore, j can be any index between 1 and n, so the probability that $A[i] = B[j]$, for a particular j, is $1/n$.

Note that the algorithm cyclically shifts the whole permutation, without changing the relative positions of the elements, and picking *offset* solely determines the resulting permutation. Since *offset* can take one of the n possible values, for a given input array of size n the algorithm is capable of producing only n different outputs, which for $n > 2$ is fewer than $n!$, and so the resulting permutation is not uniformly random.

5.3-5  *Work in progress.*

⋆ 5.4 Probabilistic analysis and further uses of indicator random variables

5.4-1  *Work in progress.*

5.4-2  *Work in progress.*

5.4-3  *Work in progress.*

5.4-4  *Work in progress.*

5.4-5  *Work in progress.*

5.4-6  *Work in progress.*

5.4-7  *Work in progress.*

5.4-8  *Work in progress.*

Problems

5-1 ***Probabilistic counting***

a. Let X_j, for $j = 1, 2, \ldots, n$, denote the increase of the value represented by the counter after the jth call to the INCREMENT operation. Furthermore, let $X = \sum_{j=1}^{n} X_j$ denote the value represented by the counter after n INCREMENT operations have been performed. Now suppose that right before the jth INCREMENT operation is performed, the counter is set to i, and as such represents a count of n_i. Upon success of the

jth call — occurring with the probability of $1/(n_{i+1} - n_i)$ — the represented count increases by $n_{i+1} - n_i$. For every $j = 1, 2, \ldots, n$ we have

$$\begin{aligned}
\mathrm{E}\left[X_j\right] &= 0 \cdot \left(1 - \frac{1}{n_{i+1} - n_i}\right) + (n_{i+1} - n_i) \cdot \frac{1}{n_{i+1} - n_i} \\
&= 1,
\end{aligned}$$

and therefore

$$\begin{aligned}
\mathrm{E}\left[X\right] &= \mathrm{E}\left[\sum_{j=1}^{n} X_j\right] \\
&= \sum_{j=1}^{n} \mathrm{E}\left[X_j\right] \\
&= \sum_{j=1}^{n} 1 \\
&= n.
\end{aligned}$$

b. Let X_j for $j = 1, 2, \ldots, n$, and X be the random variables defined in the previous part. For $n_i = 100i$ the value represented by the counter will increase by $n_{i+1} - n_i = 100$ after a call to INCREMENT, with the probability of $1/(n_{i+1} - n_i) = 1/100$. For all $j = 1, 2, \ldots, n$ we get

$$\begin{aligned}
\mathrm{Var}\left[X_j\right] &= \mathrm{E}\left[X_j^2\right] - \mathrm{E}^2\left[X_j\right] && \text{(by equation (C.31))} \\
&= 0^2 \cdot \left(1 - \frac{1}{100}\right) + 100^2 \cdot \frac{1}{100} - 1^2 \\
&= 99,
\end{aligned}$$

and since $X_1, X_2, \ldots, X_n$ are pairwise independent, we obtain

$$\begin{aligned}
\mathrm{Var}\left[X\right] &= \mathrm{Var}\left[\sum_{j=1}^{n} X_j\right] \\
&= \sum_{j=1}^{n} \mathrm{Var}\left[X_j\right] && \text{(by equation (C.33))} \\
&= \sum_{j=1}^{n} 99 \\
&= 99n.
\end{aligned}$$

5-2 Searching an unsorted array

a.

```
RANDOM-SEARCH(A, n, x)
 1  let picked[1 : n] be a new array
 2  for i = 1 to n
 3      picked[i] = FALSE
 4  k = 0
 5  while k < n
 6      i = RANDOM(1, n)
 7      if A[i] == x
 8          return i
 9      if not picked[i]
10          picked[i] = TRUE
11          k = k + 1
12  return NIL
```

The algorithm uses the auxiliary array $picked[1:n]$ of boolean values for keeping track of the information whether it has already picked a given index, and the variable k for the number of picked indices so far. In each iteration of the **while** loop in lines 5–11 the algorithm picks a random index into A and compares $A[i]$ with x. If x is found at position i, the algorithm immediately returns that position. Otherwise, if $A[i]$ hasn't been examined in previous iterations, the algorithm increments k. If array A doesn't contain the element x, then after checking all n indices at least once, the algorithm terminates by returning the special value NIL.

b. Let X be the random variable denoting the number of picked indices into A before x is found. The search for x performed by the procedure RANDOM-SEARCH is a sequence of Bernoulli trials, where a success (the condition in line 7) occurs with probability $p = 1/n$. Applying equation (C.36), we get that the expected number of picked indices is $\mathrm{E}[X] = 1/p = n$.

c. Let's again consider X and a similar sequence of Bernoulli trials to the one from part (b). Here a success occurs with probability $p = k/n$, so the expected number of indices picked in RANDOM-SEARCH before x is found, is $\mathrm{E}[X] = 1/p = n/k$.

d. This situation can be viewed as an example of the coupon collector's problem (see Section 5.4.2), where the coupons we are trying to collect are represented by the indices of array A, and obtaining a coupon is represented by picking an index in line 6 of

RANDOM-SEARCH. According to the solution to the problem presented in the book, in order to pick each of n different indices, we should expect to check approximately $n \ln n$ randomly chosen indices.

Since the running time of DETERMINISTIC-SEARCH is proportional to the number of picked indices, in parts (e)–(g) we'll focus on finding these numbers in the average and the worst cases.

e. DETERMINISTIC-SEARCH is identical to the linear search algorithm we studied in Exercise 2.1-4. The case when there is only one copy of x in the array of length n was considered in Exercise 2.2-3, where it was found that the algorithm will check $(n+1)/2$ indices in the average case, and all n indices in the worst case.

f. Let X denote the number of indices of array A picked before x is found. For $i = 1, 2, \ldots, n-k+1$, let B_i be the event in which index i is picked in DETERMINISTIC-SEARCH, and let X_i be an indicator random variable associated with B_i. Event B_i occurs as long as all instances of x contained in the array A occupy the subarray $A[i:n]$, so we have

$$\Pr\{B_i\} = \frac{\binom{n-i+1}{k}}{\binom{n}{k}}.$$

Of course $X = \sum_{i=1}^{n-k+1} X_i$, so taking expectations of both sides and applying linearity of expectation, we obtain

$$\begin{aligned}
\mathrm{E}[X] &= \sum_{i=1}^{n-k+1} \mathrm{E}[X_i] \\
&= \sum_{i=1}^{n-k+1} \Pr\{B_i\} && \text{(by Lemma 5.1)} \\
&= \sum_{i=1}^{n-k+1} \frac{\binom{n-i+1}{k}}{\binom{n}{k}} \\
&= \frac{\sum_{i=k}^{n} \binom{i}{k}}{\binom{n}{k}} && \text{(reindexing).}
\end{aligned}$$

To simplify the last expression, we'll show that

$$\sum_{i=k}^{n} \binom{i}{k} = \binom{n+1}{k+1} \tag{5}$$

for all $k = 1, 2, \ldots, n$. If $n = 1$, then $k = 1$ and both sides of (5) reduce to 1. Now suppose that $n > 1$ and assume by induction that for all $k = 1, 2, \ldots, n-1$ it holds

$$\sum_{i=k}^{n-1} \binom{i}{k} = \binom{n}{k+1}.$$

We have

$$\begin{aligned}
\sum_{i=k}^{n} \binom{i}{k} &= \sum_{i=k}^{n-1} \binom{i}{k} + \binom{n}{k} \\
&= \binom{n}{k+1} + \binom{n}{k} && \text{(by the inductive hypothesis)} \\
&= \binom{n+1}{k+1} && \text{(by Exercise C.1-7).}
\end{aligned}$$

If $k = n$, then (5) reduces to $\binom{n}{n} = \binom{n+1}{n+1}$, and so trivially holds.

Finally, we obtain the number of indices picked by DETERMINISTIC-SEARCH in the average case to be

$$\begin{aligned}
\mathrm{E}[X] &= \frac{\sum_{i=k}^{n} \binom{i}{k}}{\binom{n}{k}} \\
&= \frac{\binom{n+1}{k+1}}{\binom{n}{k}} && \text{(by equation (5))} \\
&= \frac{n+1}{k+1} && \text{(by equation (C.9)).}
\end{aligned}$$

The worst case for DETERMINISTIC-SEARCH is when all instances of x occupy the k rightmost positions of the array. The algorithm will then check exactly $n - k + 1$ indices before x is found and the algorithm terminates.

g. When x is absent in A the algorithm has to look through the entire array, so in both the average case and the worst case the algorithm checks all n array indices.

h. Suppose that we use procedure RANDOMLY-PERMUTE (see Section 5.3) to permute the input array, performing n exchanges of array elements. The running time of SCRAMBLE-SEARCH is then proportional to the sum of n and the number of comparisons performed by deterministic linear search. These numbers for different values of k were determined in parts (e), (f) and (g).

i. When there are $k \geq 1$ copies of the searched element in the input array the running times of all three algorithms are of the same order. However, if the input array does not contain the searched element (i.e., when $k = 0$), the running times of DETERMINISTIC-SEARCH and SCRAMBLE-SEARCH are asymptotically smaller than the running time of RANDOM-SEARCH. Moreover, DETERMINISTIC-SEARCH does not introduce the overhead of permuting the array, thus running faster by a constant factor than SCRAMBLE-SEARCH. Therefore, DETERMINISTIC-SEARCH seems to be most practical one to use.

Part VIII Appendix: Mathematical Background

A Summations

A.1 Summation formulas and properties

A.1-1 Let $g_1, g_2, \ldots, g_n$ be functions, so that $g_k(i) = O(f_k(i))$ for each $k = 1, 2, \ldots, n$. From the definition of O-notation, there exist positive constants $c_1, c_2, \ldots, c_n$ and i_0, such that $g_k(i) \le c_k f_k(i)$ for all $i \ge i_0$. Let $c = \max\{c_k : 1 \le k \le n\}$. For $i \ge i_0$ we have

$$\begin{aligned}\sum_{k=1}^{n} g_k(i) &\le \sum_{k=1}^{n} c_k f_k(i) \\ &\le c \sum_{k=1}^{n} f_k(i) \\ &= O\left(\sum_{k=1}^{n} f_k(i)\right).\end{aligned}$$

A.1-2

$$\begin{aligned}\sum_{k=1}^{n}(2k-1) &= 2\sum_{k=1}^{n} k - \sum_{k=1}^{n} 1 && \text{(by the linearity property)} \\ &= \frac{2n(n+1)}{2} - n && \text{(by equation (A.1))} \\ &= n^2\end{aligned}$$

A.1-3 If we let $x = 10$ in equation (A.6), we obtain:

$$\begin{aligned} 111{,}111{,}111 &= \sum_{k=0}^{8} 10^k \\ &= \frac{10^9 - 1}{10 - 1}. \end{aligned}$$

A.1-4 Each subsequent term is the previous term multiplied by $-1/2$, so letting $x = -1/2$ in equation (A.7) gives us the value of the series:

$$\begin{aligned} \sum_{k=0}^{\infty} (-1/2)^k &= \frac{1}{1 - (-1/2)} \\ &= 2/3. \end{aligned}$$

A.1-5 For the upper bound, we bound k by n:

$$\begin{aligned} \sum_{k=1}^{n} k^c &\le \sum_{k=1}^{n} n^c \\ &= n \cdot n^c \\ &= O(n^{c+1}). \end{aligned}$$

To get the lower bound, we drop around half of the smallest terms and bound the remaining ones:

$$\begin{aligned} \sum_{k=1}^{n} k^c &\ge \sum_{k=\lceil n/2 \rceil}^{n} k^c \\ &\ge \sum_{k=\lceil n/2 \rceil}^{n} \lceil n/2 \rceil^c \\ &= (n - \lceil n/2 \rceil + 1)\lceil n/2 \rceil^c \\ &= (\lfloor n/2 \rfloor + 1)\lceil n/2 \rceil^c \\ &\ge \lceil n/2 \rceil^{c+1} \\ &\ge n^{c+1}/2^{c+1} \\ &= \Omega(n^{c+1}) \qquad \text{(since } 1/2^{c+1} \text{ is a constant).} \end{aligned}$$

By applying Theorem 3.1, we obtain the tight bound.

A.1-6 Let's differentiate both sides of equation (A.11):

$$\begin{aligned}\sum_{k=0}^{\infty} k \cdot k x^{k-1} &= \frac{(1-x)^2 + 2x(1-x)}{(1-x)^4} \\ &= \frac{(1-x) + 2x}{(1-x)^3} \\ &= \frac{1+x}{(1-x)^3}.\end{aligned}$$

To get the desired result, we multiply both sides of the equation by x.

A.1-7 The asymptotic upper bound:

$$\begin{aligned}\sum_{k=1}^{n} \sqrt{k \lg k} &\le \sum_{k=1}^{n} \sqrt{n \lg n} \\ &= n \cdot \sqrt{n \lg n} \\ &= O(n^{3/2} \lg^{1/2} n).\end{aligned}$$

For the asymptotic lower bound, assume for convenience that n is even and at least 4. Then:

$$\begin{aligned}\sum_{k=1}^{n} \sqrt{k \lg k} &> \sum_{k=n/2+1}^{n} \sqrt{k \lg k} \\ &> \sum_{k=n/2+1}^{n} \sqrt{(n/2) \lg(n/2)} \\ &= (n/2) \sqrt{(n/2)(\lg n - 1)} \\ &\ge (n/2) \sqrt{(n/2)(\lg n)/2} \\ &= (n^{3/2} \lg^{1/2} n)/4 \\ &= \Omega(n^{3/2} \lg^{1/2} n).\end{aligned}$$

The asymptotically tight bound follows from Theorem 3.1.

A.1-8 ★

$$\begin{aligned}\sum_{k=1}^{n}\frac{1}{2k-1} &< 1+\sum_{k=2}^{n+1}\frac{1}{2k-2}\\ &= 1+\sum_{k=1}^{n}\frac{1}{2k}\\ &= 1+\frac{H_n}{2}\\ &= 1+\frac{\ln n+O(1)}{2} && \text{(by equation (A.9))}\\ &= \ln\sqrt{n}+O(1)\end{aligned}$$

A.1-9 ★

$$\begin{aligned}\sum_{k=0}^{\infty}\frac{k-1}{2^k} &= \sum_{k=0}^{\infty}\frac{k}{2^k}-\sum_{k=0}^{\infty}\frac{1}{2^k}\\ &= \sum_{k=0}^{\infty}k(1/2)^k-\sum_{k=0}^{\infty}(1/2)^k\\ &= \frac{1/2}{(1-1/2)^2}-\frac{1}{1-1/2} && \text{(by equations (A.11) and (A.7))}\\ &= 0\end{aligned}$$

A.1-10 ★

$$\begin{aligned}\sum_{k=1}^{\infty}(2k+1)x^{2k} &= 2\sum_{k=0}^{\infty}k(x^2)^k+\sum_{k=0}^{\infty}(x^2)^k-1\\ &= \frac{2x^2}{(1-x^2)^2}+\frac{1}{1-x^2}-1 && \text{(by equations (A.11) and (A.7))}\\ &= \frac{x^2(3-x^2)}{(1-x^2)^2}\end{aligned}$$

A.1-11 ★

$$\begin{aligned}\prod_{k=2}^{n}\left(1-\frac{1}{k^2}\right) &= \prod_{k=2}^{n}\frac{k^2-1}{k^2}\\ &= \frac{\prod_{k=2}^{n}(k^2-1)}{\prod_{k=2}^{n}k^2}\\ &= \frac{\prod_{k=2}^{n}(k-1)\cdot\prod_{k=2}^{n}(k+1)}{\left(\prod_{k=1}^{n}k\right)^2}\\ &= \frac{\prod_{k=1}^{n-1}k\cdot\prod_{k=3}^{n+1}k}{(n!)^2}\\ &= \frac{(n-1)!\cdot\frac{(n+1)!}{2}}{(n!)^2}\\ &= \frac{n+1}{2n}\end{aligned}$$

A.2 Bounding summations

A.2-1 Using the properties of telescoping series, we get

$$\begin{aligned}\sum_{k=1}^{n}\frac{1}{k^2} &\le 1+\sum_{k=2}^{n}\frac{1}{k(k-1)}\\ &= 1+\sum_{k=2}^{n}\left(\frac{1}{k-1}-\frac{1}{k}\right)\\ &= 1+1-\frac{1}{n}\\ &< 2.\end{aligned}$$

A.2-2

$$\begin{aligned}
\sum_{k=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^k} \right\rceil &< \sum_{k=0}^{\lfloor \lg n \rfloor} \left(\frac{n}{2^k} + 1 \right) && \text{(by inequality (3.2))} \\
&\le \lg n + 1 + \sum_{k=0}^{\lfloor \lg n \rfloor} \frac{n}{2^k} \\
&< \lg n + 1 + \sum_{k=0}^{\infty} \frac{n}{2^k} \\
&= \lg n + 1 + \frac{n}{1 - 1/2} && \text{(by equation (A.7))} \\
&= \lg n + 1 + 2n \\
&= O(n)
\end{aligned}$$

A.2-3

Proceeding similarly as when searching for an upper bound on the nth harmonic number, we split the range 1 to n into $\lfloor \lg n \rfloor$ pieces and lower-bound the contribution of each piece by $1/2$:

$$\begin{aligned}
\sum_{k=1}^{n} \frac{1}{k} &\ge \sum_{i=0}^{\lfloor \lg n \rfloor - 1} \sum_{j=0}^{2^i - 1} \frac{1}{2^i + j} \\
&\ge \sum_{i=0}^{\lfloor \lg n \rfloor - 1} \sum_{j=0}^{2^i - 1} \frac{1}{2^{i+1}} \\
&= \sum_{i=0}^{\lfloor \lg n \rfloor - 1} (1/2) \\
&= \lfloor \lg n \rfloor / 2 \\
&= \Omega(\lg n).
\end{aligned}$$

A.2-4

The function $f(k) = k^3$ is monotonically increasing, so we approximate the summation using inequality (A.18):

$$\int_0^n x^3 \, dx \le \sum_{k=1}^{n} k^3 \le \int_1^{n+1} x^3 \, dx.$$

The integral approximation gives the upper bound

$$\int_0^n x^3 \, dx = \frac{n^4}{4},$$

and the lower bound

$$\int_1^{n+1} x^3\,dx = \frac{(n+1)^4 - 1}{4}.$$

A.2-5 Applying inequality (A.19) to $\sum_{k=1}^{n} 1/k$ leads to an improper integral:

$$\begin{aligned}\sum_{k=1}^{n} \frac{1}{k} &\le \int_0^n \frac{dx}{x} \\ &= \lim_{a \to 0^+} \int_a^n \frac{dx}{x} \\ &= \infty.\end{aligned}$$

As a result, we don't get any useful information about the upper bound on the summation. By rewriting it as $1 + \sum_{k=2}^{n} 1/k$, we can apply formula (A.19) to the second term and upper-bound the initial summation by $\ln n + 1$.

Problems

A-1 ***Bounding summations***

In order to determine the asymptotic tight bounds on each summation, we will find their asymptotic upper bound and asymptotic lower bound by substituting each term in the summations by appropriate values. Since the bounds on the summations don't depend on the parity of n, for simplicity we will assume that n is even.

a. An asymptotic upper bound:

$$\begin{aligned}\sum_{k=1}^{n} k^r &\le \sum_{k=1}^{n} n^r \\ &= n \cdot n^r \\ &= O(n^{r+1}).\end{aligned}$$

An asymptotic lower bound:

$$\begin{aligned}\sum_{k=1}^{n} k^r &\geq \sum_{k=n/2+1}^{n} k^r \\ &\geq \sum_{k=n/2+1}^{n} (n/2)^r \\ &= (n/2) \cdot (n/2)^r \\ &= \Omega(n^{r+1}).\end{aligned}$$

Based on the results, we conclude that the asymptotic tight bound on the summation is

$$\sum_{k=1}^{n} k^r = \Theta(n^{r+1}).$$

b. An asymptotic upper bound:

$$\begin{aligned}\sum_{k=1}^{n} \lg^s k &\leq \sum_{k=1}^{n} \lg^s n \\ &= n \cdot \lg^s n \\ &= O(n \lg^s n).\end{aligned}$$

An asymptotic lower bound:

$$\begin{aligned}\sum_{k=1}^{n} \lg^s k &\geq \sum_{k=n/2+1}^{n} \lg^s k \\ &\geq \sum_{k=n/2+1}^{n} \lg^s (n/2) \\ &= (n/2) \cdot \lg^s (n/2) \\ &= \Omega(n \lg^s n).\end{aligned}$$

The asymptotic tight bound:

$$\sum_{k=1}^{n} \lg^s k = \Theta(n \lg^s n).$$

c. An asymptotic upper bound:

$$\begin{aligned}\sum_{k=1}^{n} k^r \lg^s k &\le \sum_{k=1}^{n} n^r \lg^s n \\ &= n \cdot n^r \lg^s n \\ &= O(n^{r+1} \lg^s n).\end{aligned}$$

An asymptotic lower bound:

$$\begin{aligned}\sum_{k=1}^{n} k^r \lg^s k &\ge \sum_{k=n/2+1}^{n} k^r \lg^s k \\ &\ge \sum_{k=n/2+1}^{n} (n/2)^r \lg^s (n/2) \\ &= (n/2) \cdot (n/2)^r \lg^s (n/2) \\ &= \Omega(n^{r+1} \lg^s n).\end{aligned}$$

The asymptotic tight bound:

$$\sum_{k=1}^{n} k^r \lg^s k = \Theta(n^{r+1} \lg^s n).$$

Assuming $s = 0$ and $r = 0$ in the bound above, respectively, we get the summations and their asymptotic tight bounds from parts (a) and (b).

B Sets, Etc.

B.1 Sets

B.1-1 See Figure B.1-1.

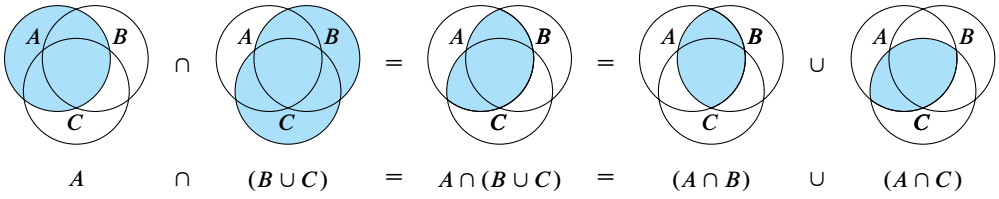


Figure B.1-1 A Venn diagram illustrating the first of the distributive laws (B.1).

B.1-2 We prove the first formula by induction on n. For $n = 1$ the proof is trivial, and for $n = 2$ the formula is the first of DeMorgan's laws. Let's assume that $n > 2$, and that the formula holds for a family of $n - 1$ sets. We have:

$$\begin{aligned}\overline{A_1 \cap A_2 \cap \cdots \cap A_{n-1} \cap A_n} &= \overline{(A_1 \cap A_2 \cap \cdots \cap A_{n-1}) \cap A_n} \\ &= \overline{A_1 \cap A_2 \cap \cdots \cap A_{n-1}} \cup \overline{A_n} \\ &= \overline{A_1} \cup \overline{A_2} \cup \cdots \cup \overline{A_{n-1}} \cup \overline{A_n}.\end{aligned}$$

The second equality follows from the first of DeMorgan's laws, and the third equality follows from the inductive hypothesis.

The proof of the second formula is similar. If $n = 2$, then the formula is DeMorgan's second law, and if $n > 2$, then it is enough to apply the above reasoning with the symbols $\cup$ and $\cap$ swapped, and use DeMorgan's second law.

B.1-3 ★ We prove the principle of inclusion and exclusion by induction on n. For $n = 1$ the proof is trivial, and for $n = 2$ the formula is identical to (B.3). If $n > 2$, then by (B.3)

and the generalized first distributive law, we have:

$$\begin{aligned}|A_1 \cup A_2 \cup \cdots \cup A_n| &= |(A_1 \cup A_2 \cup \cdots \cup A_{n-1}) \cup A_n| \\ &= |A_1 \cup A_2 \cup \cdots \cup A_{n-1}| + |A_n| \\ &\quad - |(A_1 \cup A_2 \cup \cdots \cup A_{n-1}) \cap A_n| \\ &= |A_1 \cup A_2 \cup \cdots \cup A_{n-1}| + |A_n| \\ &\quad - |(A_1 \cap A_n) \cup \cdots \cup (A_{n-1} \cap A_n)|.\end{aligned}$$

We now apply the induction hypothesis to the first and the last summand:

$$\begin{aligned}|A_1 \cup A_2 \cup \cdots \cup A_{n-1}| = &\sum_{1 \le i_1 < n} |A_{i_1}| \\ &- \sum_{1 \le i_1 < i_2 < n} |A_{i_1} \cap A_{i_2}| \\ &+ \sum_{1 \le i_1 < i_2 < i_3 < n} |A_{i_1} \cap A_{i_2} \cap A_{i_3}| \\ &\qquad \vdots \\ &+ (-1)^{n-2} |A_1 \cap A_2 \cap \cdots \cap A_{n-1}|,\end{aligned}$$

$$\begin{aligned}|(A_1 \cap A_n) \cup \cdots \cup (A_{n-1} \cap A_n)| = &\sum_{1 \le i_1 < n} |A_{i_1} \cap A_n| \\ &- \sum_{1 \le i_1 < i_2 < n} |A_{i_1} \cap A_{i_2} \cap A_n| \\ &\qquad \vdots \\ &+ (-1)^{n-2} |A_1 \cap A_2 \cap \cdots \cap A_{n-1} \cap A_n|.\end{aligned}$$

We finish the proof by inserting the resulting expressions into the initial formula:

$$\begin{aligned}|A_1 \cup A_2 \cup \cdots \cup A_n| = &\sum_{1 \le i_1 \le n} |A_{i_1}| \\ &- \sum_{1 \le i_1 < i_2 \le n} |A_{i_1} \cap A_{i_2}| \\ &+ \sum_{1 \le i_1 < i_2 < i_3 \le n} |A_{i_1} \cap A_{i_2} \cap A_{i_3}| \\ &\qquad \vdots \\ &+ (-1)^{n-1} |A_1 \cap A_2 \cap \cdots \cap A_n|.\end{aligned}$$

B.1-4 An example of a one-to-one correspondence between $\mathbb{N}$ and the set of odd natural

numbers is: $n \to 2n + 1$. Thus, the set of odd natural numbers is countable.

B.1-5 We prove by induction on the cardinality of the set S.

When $|S| = 0$, it holds that $|2^S| = 2^{|S|} = 1$, because S has exactly one subset — the empty set. Now let $|S| > 0$ and assume that $|2^S| = 2^{|S|}$. Choose $p \notin S$ and consider the set $S' = S \cup \{p\}$. We can divide the subsets of S' into those that contain p and those that do not contain p. By the inductive hypothesis there are $|2^S| = \left|2^{S'-\{p\}}\right| = 2^{|S'-\{p\}|}$ of the latter. In fact there are as many subsets containing p, because each is formed as a union of the singleton $\{p\}$ with some subset not containing p. Thus we have:

$$\begin{aligned}\left|2^{S'}\right| &= 2 \cdot 2^{|S'-\{p\}|} \\ &= 2^{|S'-\{p\}|+1} \\ &= 2^{|S'|}.\end{aligned}$$

B.1-6 An n-tuple of elements $a_1, a_2, \ldots, a_n$ is defined as:

$$(a_1, a_2, \ldots, a_n) = \begin{cases} \emptyset & \text{if } n = 0, \\ \{a_1\} & \text{if } n = 1, \\ \{a_1, \{a_1, a_2\}\} & \text{if } n = 2, \\ (a_1, (a_2, \ldots, a_n)) & \text{if } n \geq 3. \end{cases}$$

B.2 Relations

B.2-1 To show that the relation "$\subseteq$" on the power set $2^{\mathbb{Z}}$ is a partial order, we will show that "$\subseteq$" is reflexive, antisymmetric and transitive. Let $A, B, C \in 2^{\mathbb{Z}}$. Of course, $x \in A$ implies $x \in A$, so reflexivity trivially holds. If $A \subseteq B$ and $B \subseteq A$, then $x \in A$ implies $x \in B$, and $x \in B$ implies $x \in A$. By the law of transitivity of implication, $x \in A$ if and only if $x \in B$, and therefore $A = B$. The combination of $A \subseteq B$ and $B \subseteq C$ means that $x \in A$ implies $x \in B$, and $x \in B$ implies $x \in C$. Referring again to the law of transitivity of implication, we have that $x \in A$ implies $x \in C$, that is, $A \subseteq C$.

The relation "$\subseteq$" on $2^{\mathbb{Z}}$ is not a total order, because, for example, $\{0, 1\} \not\subseteq \{1, 2\}$ and $\{1, 2\} \not\subseteq \{0, 1\}$.

B.2-2 Let us denote the relation "equivalent modulo n", for $n \in \mathbb{N} - \{0\}$, by

$$R_n = \{(a, b) \in \mathbb{Z} \times \mathbb{Z} : a = b \pmod{n}\}.$$

For any $a \in \mathbb{Z}$ we have $a = a \pmod{n}$, because $a - a = 0 \cdot n$, so the relation R_n is reflexive. For any $a, b \in \mathbb{Z}$, if there exists $q \in \mathbb{Z}$ that $a - b = qn$, then $b - a = -qn$, and therefore the fact that $a = b \pmod{n}$ implies $b = a \pmod{n}$. Therefore, R_n is symmetric. For transitivity let's choose any $a, b, c \in \mathbb{Z}$ such that $a = b \pmod{n}$ and $b = c \pmod{n}$. This means that there exist $q, r \in \mathbb{Z}$ such that $a - b = qn$ and $b - c = rn$. Hence $a - c = a - b + b - c = qn + rn = (q + r)n$, and therefore $a = c \pmod{n}$.

Based on the proof above R_n is an equivalence relation that partitions the set $\mathbb{Z}$ into n abstraction classes; the ith class, where $i = 1, 2, \ldots, n$, is the set of integers that leave remainder $i - 1$ when divided by n.

B.2-3 ***a.*** The relation $\{(a, a), (b, b), (c, c), (a, b), (b, a), (a, c), (c, a)\}$ on the set $\{a, b, c\}$.

b. The relation $\{(a, a), (a, b), (b, b)\}$ on the set $\{a, b\}$.

c. The empty relation on any nonempty set.

B.2-4 If R is an equivalence relation, then $s \in [s]$ for all $s \in S$. By antisymmetry of R, if $s' \, R \, s$ and $s \, R \, s'$, then $s = s'$, so there are no such s' that $s' \in [s] - \{s\}$. This means that for all $s \in S$, $[s] = \{s\}$.

B.2-5 Symmetry and transitivity are defined using implications. For an implication "p implies q" to hold true, it isn't required for p to be true. A relation on a set remains symmetric and transitive, if there is an element in the set, that is not related to any element in that set. Reflexivity, on the other hand, requires that each element is related to itself. Thus, there exist relations that are symmetric and transitive relations but not reflexive (an example of one is given in Exercise B.2-3(c)).

B.3 Functions

B.3-1 ***a.*** For any function $f : A \to B$, $f(A) \subseteq B$, which results directly from the definition of the range of f. Additionally, if f is injective, then each argument $a \in A$ has a distinct image $f(a) \in f(A)$, so $|A| = |f(A)|$. Hence, $|A| \leq |B|$.

b. For any function $f : A \to B$, where A is finite, it holds $|A| \geq |f(A)|$, because there may be arguments $a_1, a_2 \in A$ such that $f(a_1) = f(a_2)$. Additionally, if f is surjective, then $f(A) = B$, and therefore $|A| \geq |B|$.

B.3-2 The function $f(x) = x + 1$, with $\mathbb{N}$ as both the domain and the codomain, is not bijective because there is no $x \in \mathbb{N}$ such that $f(x) = 0$. If we consider $\mathbb{Z}$ as the domain and the codomain, then f becomes bijective, since every integer y in the codomain is an image under f of a uniquely determined integer in the domain: $y = f(y - 1)$.

B.3-3 The definition comes as a natural generalization of the functional inverse. Let R be a binary relation on sets A an B. The binary relation R^{-1} on sets B and A, such that

$b\ R^{-1}\ a$ if and only if $a\ R\ b$,

is called the ***inverse*** of R.

B.3-4 ★ Let $h : \mathbb{Z} \to \mathbb{Z} \times \mathbb{Z}$ be the function we are looking for. Since the inverse of a bijection is also a bijection, we will focus on finding the inverse $h^{-1} : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$, which is equivalent to finding a way to sequentially number each ordered pair with integer elements, so that each integer is used as the number of some pair. We will now describe the construction of one of such mappings.

Let us make some simplification — instead of numbering pairs by integers, we will restrict ourselves to natural numbers. Let $g : \mathbb{Z} \times \mathbb{Z} \to \mathbb{N}$ and $f : \mathbb{N} \to \mathbb{Z}$ be such bijections that $h^{-1} = f \circ g$. It was already noted on page 1162 of the text that $f(n) = (-1)^n \lceil n/2 \rceil$ is bijective, so all that remains is to find the function g.

Consider the numbering of ordered pairs with integer elements illustrated in Figure B.3-4 in the form of a spiral. Since each such pair (x, y) is assigned a unique natural number, we can treat this spiral as a description of the function g. Let $d = \max\{|x|, |y|\}$ and let $D = (2d - 1)^2 - 1$. Informally, d and D denote, respectively, the number of the "lap" around the origin covered by the spiral at the time of passing through (x, y), and the largest number on the spiral during the previous "lap". Then we can define the function g as follows:

$$g(x, y) = \begin{cases} 0 & \text{if } d = |x| = |y| = 0, \\ D + \ \ d + y & \text{if } d = x \neq |y|, \\ D + 3d - x & \text{if } d = y \neq 0, \\ D + 5d - y & \text{if } d = -x \neq |y|, \\ D + 7d + x & \text{if } d = -y \neq 0. \end{cases}$$

B.4 Graphs

B.4-1 We can represent the relation of "shaking hands" on the faculty party as an undirected graph $G = (V, E)$, where V is the set of professors attending the party and E is the

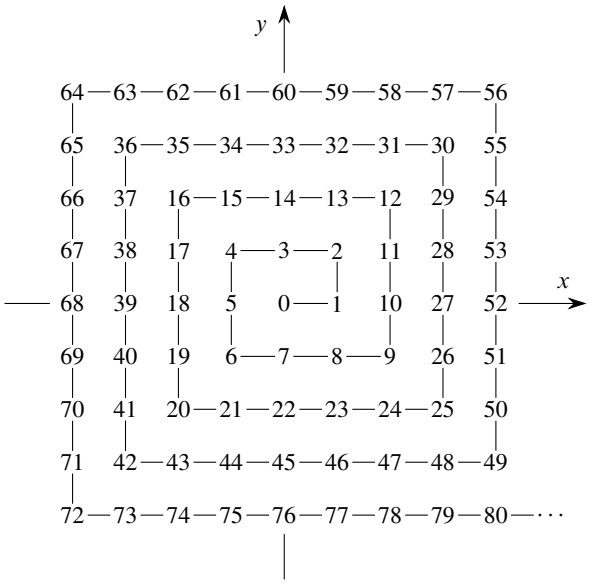


Figure B.3-4 A bijection from $\mathbb{Z} \times \mathbb{Z}$ to $\mathbb{N}$. Individual natural numbers denote the values of this bijection for points with integer coordinates in the Cartesian coordinate system.

set of unordered pairs of professors who shook hands. Summing up the degrees of all vertices of G, we get twice the number of its edges, because each edge is incident on exactly two vertices.

B.4-2 Let $\langle v_0, v_1, \ldots, v_n \rangle$ be a path from vertex u to vertex v in a directed or undirected graph. If the path is simple, the vertices in the path are distinct. Otherwise, the path contains a cycle $\langle v_i, v_{i+1}, \ldots, v_{i+k}, v_i \rangle$ for some i and k, and by eliminating the subpath $\langle v_i, v_{i+1}, \ldots, v_{i+k} \rangle$ from the path, we discard an extra repetition of v_i. By applying this modification until we remove all such repetitions, we obtain a simple path from u to v.

The proof for cycles is analogous with $v_0 = v_n$ and we have to remember not to remove the very last repetition of v_0, required for the path to form a cycle.

B.4-3 We proceed by induction on the number of vertices in the graph $G = (V, E)$. When $|V| = 1$, the graph has no edges and the inequality trivially holds. For the inductive step, pick a vertex $v \in V$. G is connected so there must be at least one edge incident on v and another vertex from V. Let $G' = (V', E')$ be the subgraph of G induced by $V' = V - \{v\}$ (i.e., the graph G with vertex v and all edges incident on v removed).

We have $|V'| = |V| - 1$ and $|E'| \leq |E| - 1$, so:

$$\begin{aligned} |E| &\geq |E'| + 1 \\ &\geq (|V'| - 1) + 1 \qquad \text{(by the inductive hypothesis applied for } G') \\ &= |V| - 1. \end{aligned}$$

B.4-4 Every vertex of a directed or undirected graph is reachable from itself because there is a path of length 1 containing only that vertex, therefore the "is reachable from" relation is reflexive.

Let u, v, w be vertices of a directed or undirected graph such that $u \overset{p}{\rightsquigarrow} v$ and $v \overset{q}{\rightsquigarrow} w$. Then there is a path from u to w which is a concatenation of the sequences p and q (with the extra v between them eliminated), which proves transitivity.

The relation is symmetric only in an undirected graph, because for its any vertices u, v, if $u \overset{p}{\rightsquigarrow} v$ then we can mirror the sequence p to obtain p' such that $v \overset{p'}{\rightsquigarrow} u$. The sequence p' is a valid path, because if (x, y) is an edge, so is (y, x). In a directed graph the edges are ordered pairs, so if (x, y) is an edge, (y, x) may not be. Thus mirroring may not produce a valid path and so symmetry does not hold in general for directed graphs.

B.4-5 See Figure B.4-5.

Figure B.4-5 **(a)** The undirected version of the directed graph in Figure B.2(a). **(b)** The directed version of the undirected graph in Figure B.2(b).

B.4-6 A hypergraph $H = (V_H, E_H)$ can be represented as a bipartite graph $G = (V_1 \cup V_2, E)$, where $V_1 = V_H$ and $V_2 = E_H$. A pair $\{v_1, v_2\}$, such that $v_1 \in V_1$ and $v_2 \in V_2$, is an edge of the graph G, if and only if the hyperedge v_2 is incident on vertex v_1 in the hypergraph H. In the graph G there are no edges between elements of V_1 or between elements of V_2, so G is indeed bipartite.

B.5 Trees

B.5-1 See Figure B.5-1.

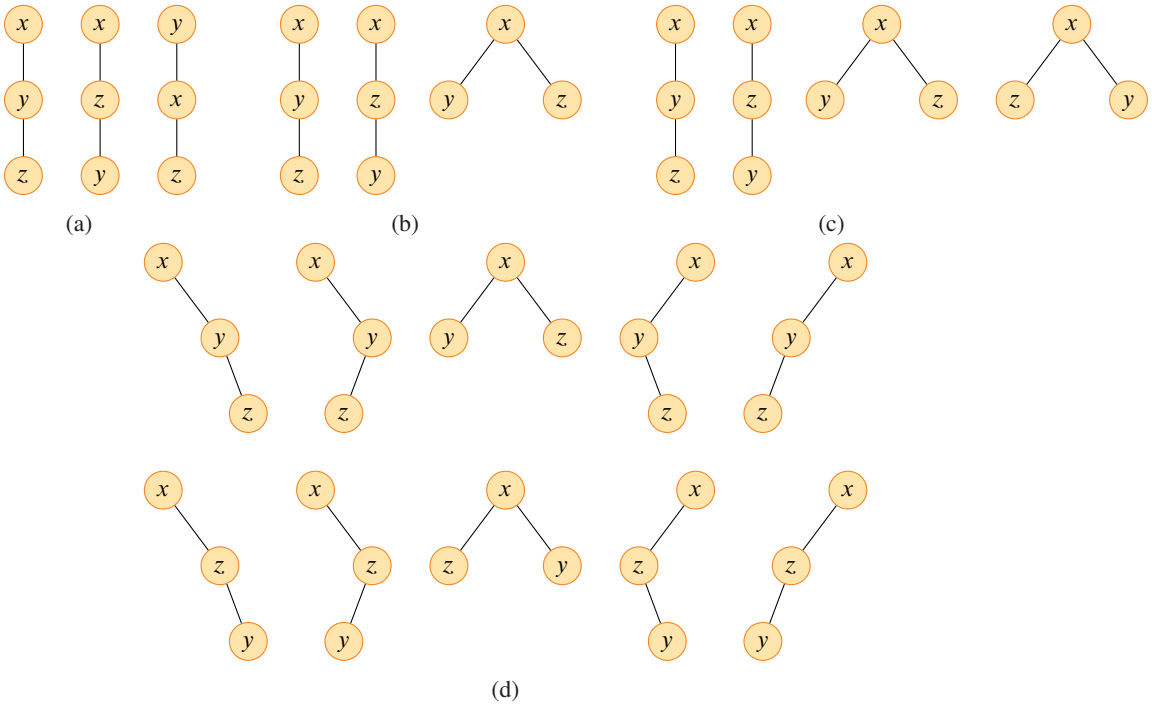


Figure B.5-1 **(a)** All the free trees with vertices x, y, and z. **(b)** All the rooted trees with nodes x, y, and z with x as the root. **(c)** All the ordered trees with nodes x, y, and z with x as the root. **(d)** All the binary trees with nodes x, y, and z with x as the root.

B.5-2 Let $G' = (V, E')$ be the undirected version of the graph G. Note that G' is connected, since every vertex $v \in V$ is reachable from vertex v_0 in G'.

Now suppose that G' does not form a tree. This means it has a cycle of length $k \geq 3$, in particular a simple cycle of such a length (after Exercise B.4-2), say $\langle v_1, v_2, \ldots, v_{k+1} \rangle$, where $v_{k+1} = v_1$. For simplicity, let $v_{k+2} = v_2$. The graph G is acyclic, so for some integer $1 \leq l \leq k$ there must be edges $(v_l, v_{l+1}), (v_{l+2}, v_{l+1}) \in E$. By assumption, both v_l and v_{l+2} are reachable from v_0, so there are two different paths from v_0 to v_{l+1}: $v_0 \rightsquigarrow v_l \rightarrow v_{l+1}$ and $v_0 \rightsquigarrow v_{l+2} \rightarrow v_{l+1}$. The obtained contradiction leads to the conclusion that G' is acyclic and so it is a tree.

B.5-3 The proof is by induction on the number n of nodes in the tree. The base case is when $n = 1$. A binary tree with only one node has one leaf and no degree-2 nodes, thus the statement holds.

For the inductive step, suppose that $n \geq 1$ and assume that the statement is true for all binary trees with n nodes. Let T be a binary tree with $n + 1$ nodes. We can remove the root node and obtain two subtrees, each of which has less than $n + 1$ nodes. If any of the subtrees is empty, the statement follows immediately from the inductive hypothesis applied to the other (nonempty) subtree, since the root of T is neither a leaf nor a degree-2 node.

Now suppose that both the left subtree and the right subtree are nonempty. Let L and D be, respectively, the number of leaves and the number of degree-2 nodes in T. Similarly, let L_1 and D_1 be the analogous numbers for the left subtree, and let L_2 and D_2 be the analogous numbers for the right subtree. It holds that $L = L_1 + L_2$ and $D = D_1 + D_2 + 1$. Then:

$$\begin{aligned} D &= D_1 + D_2 + 1 \\ &= L_1 - 1 + L_2 - 1 + 1 && \text{(by the inductive hypothesis)} \\ &= L_1 + L_2 - 1 \\ &= L - 1. \end{aligned}$$

In a full binary tree each internal node is a degree-2 node, so based on the above result, such a tree has one fewer internal nodes than leaves.

B.5-4 We construct a sequence of full binary trees $T_1, T_2, \ldots$, as follows. Let T_1 be a binary tree containing a single node, so it has one leaf. For any $k \geq 2$, let T_k be a binary tree whose left subtree is T_{k-1} and whose right subtree is a one-node binary tree. Given that T_{k-1} has $k - 1$ leaves, T_k has $(k - 1) + 1 = k$ leaves.

B.5-5 The proof is by induction on the height h of the binary tree.

The only nonempty binary tree with $h = 0$ is the one with a single node, so $n = 1$ and the statement $h \geq \lfloor \lg n \rfloor$ holds. Now let $h > 0$ and suppose that the statement holds for all binary trees with heights at most $h - 1$. Consider a binary tree T with height h. One of its subtrees must have height $h' = h - 1$, and the other subtree must have height $h'' \leq h - 1$. Let n' and n'' be the numbers of nodes in these subtrees, respectively. By the inductive hypothesis, $h' \geq \lfloor \lg n' \rfloor$ and $h'' \geq \lfloor \lg n'' \rfloor$. Observe that for any integer $k \geq 0$ it is true that $\lfloor \lg 2^{k+1} \rfloor = k + 1$ but $\lfloor \lg(2^{k+1} - 1) \rfloor = k$. Therefore we have

$$n' \leq 2^{h'+1} - 1$$

and

$$n'' \leq 2^{h''+1} - 1.$$

The tree T has $n = n' + n'' + 1$ nodes, so

$$\begin{aligned} n &= n' + n'' + 1 \\ &\le 2^{h'+1} - 1 + 2^{h''+1} - 1 + 1 \\ &\le 2^h + 2^h - 1 \\ &= 2^{h+1} - 1. \end{aligned}$$

Using the above observation again, we conclude that $\lfloor \lg n \rfloor \le h$.

B.5-6 ★ We proceed by induction on n. The base case is when $n = 0$. The only full binary tree that has no internal nodes is a one-node tree, for which we have $e = i = 0$ and the statement holds.

Now let $n \ge 1$ and suppose that the statement holds for all full binary trees with less than n internal nodes. We can construct an arbitrary full binary tree T with n internal nodes by choosing any pair of full binary trees, whose numbers of internal nodes sum up to $n - 1$, for its left subtree T_L and its right subtree T_R. For $k \ge 0$, if T_L has k internal nodes, T_R has $n - k - 1$ internal nodes. By the conclusion made in Exercise B.5-3, T_L and T_R have $k + 1$ and $n - k$ leaves, respectively.

Let i_L and i_R be the internal path lengths of T_L and T_R, respectively, and similarly, let e_L and e_R be their external path lengths. The depth of each node in T — other than the root, for which the depth is 0 — is 1 more than the depth of this node in either T_L or T_R. Hence,

$$\begin{aligned} i &= 0 + (i_L + k) + (i_R + n - k - 1) \\ &= i_L + i_R + n - 1 \end{aligned}$$

and

$$\begin{aligned} e &= (e_L + k + 1) + (e_R + n - k) \\ &= e_L + e_R + n + 1. \end{aligned}$$

By the inductive hypothesis, $e_L = i_L + 2k$ and $e_R = i_R + 2(n - k - 1)$, so

$$\begin{aligned} e - i &= e_L + e_R + n + 1 - i_L - i_R - n + 1 \\ &= i_L + 2k + i_R + 2(n - k - 1) - i_L - i_R + 2 \\ &= 2n. \end{aligned}$$

Therefore, the statement holds for all full binary trees.

B.5-7 ★ The proof is by induction on the number n of nodes of the tree. In the empty tree (where $n = 0$) the sum of the leaf weights is 0, which obviously satisfies the inequality. When

$n = 1$, then the tree consists of a single leaf with a depth of 0, so the sum of the leaf weights is $2^{-0} = 1$.

Now suppose that $n > 1$ and that the inequality holds for trees with less than n nodes. Let T be a binary tree with n nodes, and let T_L, T_R be its left and right subtrees, respectively. By the inductive hypothesis, we have that the inequality is satisfied separately for trees T_L and T_R. The depth of a leaf in tree T is 1 greater than the depth of the same leaf in either T_L or T_R, hence the weights of the leaves of T are half as compared to the weights of the same leaves in its subtrees. Thus, in tree T both the sum of the leaf weights from T_L, as well as the sum of the leaf weights from T_R do not exceed $1/2$, and so the total sum of the leaf weights in T does not exceed 1.

B.5-8 ★ Let T be a binary tree with $L \geq 2$ leaves. Suppose that T does not contain a subtree having between $L/3$ and $2L/3$ leaves, inclusive. Since $L \geq 2$, the root of T is not a leaf. We will follow a path p from the root to a leaf, at each level traversing from a node to its child whose subtree has more leaves. The numbers of leaves in subtrees rooted at the visited nodes decrease from L to 1 as we move on the path p. By assumption, there is a node x on path p such that the subtree rooted at x has more than $2L/3$ leaves, and that the subtree T_y rooted at its child y has less than $L/3$ leaves. The subtree rooted at the other child of x has fewer leaves than T_y (0 if x has only one child). This leads to a contradiction because the total number of leaves in both subtrees is less than $2L/3$.

Problems

B-1 ***Graph coloring***

a. For a given tree let us distinguish any of its nodes and view the tree as a rooted tree in that node. Edges in such a tree are incident on nodes from adjacent levels, so nodes can be colored according to the parity of their depth in the tree (e.g., those on odd levels get color 1, and those on even levels get color 2).

b. (1) $\Rightarrow$ (2): Since $G = (V, E)$ is a bipartite graph, the vertex set V can be partitioned into two subsets V_1 and V_2, such that vertices from one subset aren't adjacent to vertices from the other subset. Thanks to this property it is possible to assign one color to each vertex from V_1 and a different color to each vertex from V_2, obtaining a 2-coloring of graph G.

(2) $\Rightarrow$ (3): Suppose that graph G has a cycle $p = \langle v_0, v_1, \ldots, v_{2k+1} \rangle$ for some $k \geq 1$. Let c be a 2-coloring of G and without loss of generality suppose that $c(v_0) = 1$. Then it must be $c(v_{2i}) = 1$ and $c(v_{2i+1}) = 2$ for all $i = 0, 1, \ldots, k$. But since p

is a cycle, $v_{2k+1} = v_0$, which yields the contradiction. Therefore, graph G must not contain a cycle of odd length.

(3) $\Rightarrow$ (1): Suppose that graph G is connected. Otherwise, apply the following proof separately to each connected component of G.

Choose any $v_0 \in V$. Let V_1 be the set of all vertices $v \in V$ for which there exists a path from v_0 to v of even length, and let $V_2 = V - V_1$. If there were vertices u, $w \in V_1$, such that there is a path from u to w of odd length, then $v_0 \rightsquigarrow u \rightsquigarrow w \rightsquigarrow v_0$ would be a cycle of an odd length. Therefore, no two vertices from V_1 are adjacent, and we can prove a similar fact for vertices in V_2. This means that $G = (V_1 \cup V_2, E)$ is bipartite.

c. We carry out the proof by induction on the number of vertices in graph $G = (V, E)$. If the graph has only one vertex (with degree 0), then of course one color is enough.

Now suppose that $|V| > 1$. Let us choose an arbitrary vertex $v \in V$ and denote by G' the subgraph of G induced by the set $V - \{v\}$. By the inductive hypothesis, G' can be colored with $d' + 1$ colors, where d' is the maximum degree of any vertex in G'. It is true that $d' \leq d$, so G' can also be colored with $d + 1$ colors. Since vertex v has at most d neighbors in G, among the colors used in a $(d + 1)$-coloring of G' there is one that is not assigned to any neighbor of v in graph G. By assigning that color to v, we obtain a $(d + 1)$-coloring of G.

d. For any $k \geq 2$, if graph G is k-colorable, but is not $(k-1)$-colorable, then for every two different colors used in a k-coloring of G there are two adjacent vertices, one with the first color and the other with the second color. If that weren't true, there would be two different colors that we could not distinguish, and so we could reduce the number of colors required, obtaining a $(k-1)$-coloring of G.

Based on the above reasoning, we have $\binom{k}{2} \leq |E|$. We use the fact that for $k \geq 2$ it holds that $k/2 \leq k - 1$, and so

$$\begin{aligned} k^2 &= 4(k/2)(k/2) \\ &\leq 4(k-1)k/2 \\ &= 4\binom{k}{2} \\ &\leq 4|E| \\ &= O(|V|). \end{aligned}$$

This fact implies that there exist positive constants c and k_0 such that $0 \leq k^2 \leq c|V|$ for all $k \geq k_0$. By applying square roots, we obtain that $0 \leq k \leq \sqrt{c}\sqrt{|V|}$ for all

$k \geq k_0$, and since $\sqrt{c}$ is a positive constant, we get $k = O(\sqrt{|V|})$.

B-2 Friendly graphs

a.

Theorem
In any undirected graph $G = (V, E)$, where $|V| \geq 2$, there are two vertices with the same degree.

Proof The degrees of the vertices in graph G are numbers from 0 to $|V| - 1$, inclusive. Observe, however, that G has an isolated vertex if and only if it doesn't have a vertex adjacent to any other vertex of G. Therefore, the degrees of the vertices of G can in fact take $|V| - 1$ possible values, so some two vertices must have equal degrees. ■

b.

Theorem
An undirected graph $G = (V, E)$ of 6 vertices, or its complement[1] $\overline{G}$, contain a clique[2] of size 3.

Proof Let's pick any $v \in V$. Since $|V| = 6$, there is a set $U \subseteq V - \{v\}$ of three vertices, for which exactly one of the following cases holds:

1. each vertex in U is adjacent to v,
2. no vertex in U is adjacent to v.

Since case 1 in G is equivalent to case 2 in $\overline{G}$, and vice versa, let us focus on case 1 only. If there is a pair of adjacent vertices among those in U, then the vertices of this pair together with v form a clique in G. Now suppose there is no such pair. Then, however, the vertices in U form a clique in $\overline{G}$. ■

c.

Theorem
For any undirected graph $G = (V, E)$ the set V can be partitioned into two subsets such that for any vertex $v \in V$ at least half of its neighbors do not belong to the subset that v belongs to.

---

[1] See page 1085 of the text.
[2] See page 1081 of the text.

Proof Let us divide the vertices of G arbitrarily into two disjoint subsets V_1 and V_2. Let

$$S = \{(v_1, v_2) \in E : v_1 \in V_1,\ v_2 \in V_2\}.$$

If more than half of the neighbors of some vertex $v \in V_1$ are also in V_1, then moving v to V_2 increases the size of set S. Similarly, for vertices $v \in V_2$ with this property — by moving them to set V_1, we increase set S. This process must terminate after a finite number of steps, because S cannot grow indefinitely. The obtained partition $V = V_1 \cup V_2$ satisfies the desired property. ■

d. Before we formulate the main theorem, we will need the following lemma.

Lemma
Let $G = (V, E)$ be an undirected graph, where $|V| \geq 3$ and

$$\text{degree}(u) + \text{degree}(v) \geq |V| \tag{6}$$

for every pair of non-adjacent vertices $u, v \in V$. Then G is hamiltonian[3].

Proof Suppose the lemma does not hold, that is there is a graph $G = (V, E)$ with at least 3 vertices, that satisfies the property (6), but is not hamiltonian. Among all such graphs let us consider that of the greatest number of edges $|E|$. Let $n = |V|$. Then G must have a hamiltonian path[4] $\langle v_1, v_2, \ldots, v_n \rangle$ — otherwise we could add the missing edges without violating (6) and get a graph with more than $|E|$ edges. Since G does not have a hamiltonian cycle, $(v_1, v_n) \notin E$. By assumption, we have that $\text{degree}(v_1) + \text{degree}(v_n) \geq n$.

Now let us define the following sets:

$$S_1 = \{i : 2 \leq i \leq n \text{ and } (v_1, v_i) \in E\},$$
$$S_n = \{i : 2 \leq i \leq n \text{ and } (v_{i-1}, v_n) \in E\}.$$

Then $|S_1| = \text{degree}(v_1)$ and $|S_n| = \text{degree}(v_n)$. Since $|S_1| + |S_n| \geq n$ and the set $S_1 \cup S_n$ has at most $n - 1$ elements, the set $S_1 \cap S_n$ must be non-empty. So there exists i for which (v_1, v_i), $(v_{i-1}, v_n) \in E$. Hence, the path

$$v_1 \rightsquigarrow v_{i-1} \to v_n \to v_{n-1} \rightsquigarrow v_i \to v_1$$

is a hamiltonian cycle in graph G, which is a contradiction. ■

Theorem
Let $G = (V, E)$ be an undirected graph, where $|V| \geq 3$ and

$$\text{degree}(v) \geq |V|/2$$

[3]See page 1056 of the text.
[4]See Exercise 34.2-6.

for every vertex $v \in V$. Then G is hamiltonian.

Proof If the property holds for every $v \in V$, then

$$\text{degree}(u) + \text{degree}(v) \geq |V|$$

for every $u, v \in V$ whether they are adjacent or not. So G satisfies the assumptions of the lemma, and thus it is hamiltonian. ■

B-3 *Bisecting trees*

a. The fact trivially holds for binary trees with at most three nodes, so in the rest of the proof we will assume that $n \geq 4$. We define a ***branch*** of a binary tree as a path $\langle x_0, x_1, \ldots, x_k \rangle$, where x_0 is the root, x_k is a leaf, and for $i = 1, 2, \ldots, k$, x_i is the root of the subtree of x_{i-1} with more nodes. Note that a tree may have more than one branch. Let's denote by $s(x)$ the number of nodes in the subtree rooted at x. For every $i = 0, 1, \ldots, k-1$,

$$s(x_i) \leq 2s(x_{i+1}) + 1. \tag{7}$$

Of course, the sequence $\langle s(x_0), s(x_1), \ldots, s(x_k) \rangle$ decreases from n to 1, so there must be j, such that $s(x_j) > n/4$ and $s(x_{j+1}) \leq n/4$. Thus, by removing edge (x_{j-1}, x_j), we partition the nodes of the tree into two sets of sizes

$$\begin{aligned} s(x_j) &\leq 2s(x_{j+1}) + 1 && \text{(by inequality (7))} \\ &= 2n/4 + 1 \\ &\leq 3n/4 \end{aligned}$$

and

$$\begin{aligned} n - s(x_j) &< n - n/4 \\ &= 3n/4. \end{aligned}$$

b. The constant $3/4$ is sufficient to make balanced partitions, as we have shown in part (a). The example of the binary tree in Figure B-3 shows that the constant cannot be lower.

c. Assume that $n \geq 6$. We'll remove one edge from the original tree to cut off a subtree of at most $\lfloor n/2 \rfloor$ vertices. Consider any of the tree's branches $\langle x_0, x_1, \ldots, x_k \rangle$, as well as the function s, both defined in the solution to part (a). There is an edge (x_j, x_{j+1}),

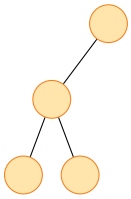


Figure B-3 The binary tree whose most evenly balanced partition upon removal of a single edge has a subset with 3 vertices.

such that $s(x_j) > \lfloor n/2 \rfloor$ and $s(x_{j+1}) \leq \lfloor n/2 \rfloor$. From inequality (7) we have

$$\begin{aligned} s(x_{j+1}) &\geq (s(x_j) - 1)/2 \\ &> (\lfloor n/2 \rfloor - 1)/2 \\ &\geq \lfloor n/2 \rfloor / 3. \end{aligned}$$

We cut off the subtree rooted at x_{j+1} and continue partitioning the remaining tree, with at most

$$\lfloor n/2 \rfloor - s(x_{j+1}) < (2/3)\lfloor n/2 \rfloor$$

vertices still to remove from it. In each subsequent step the number of nodes remaining to be cut is reduced by a factor of $2/3$. Therefore, we'll need to remove $O(\log_{3/2} n) = O(\lg n)$ edges.

C Counting and Probability

C.1 Counting

C.1-1 We will only count non-empty k-substrings, so $k \geq 1$. The first k-substring occupies positions 1 through k in the n-string, the second one occupies positions 2 through $k+1$, and so on. The last k-substring ends at position n, so it must start at position $n-k+1$. So an n-string has $n-k+1$ k-substrings in total.

By the rule of sum, the total number of all substrings of an n-string is

$$\begin{aligned}\sum_{k=1}^{n}(n-k+1) &= \sum_{k=1}^{n} k \\ &= \frac{n(n+1)}{2} && \text{(by equation (A.1))} \\ &= \binom{n+1}{2}.\end{aligned}$$

C.1-2 Consider the set of n-bit integers $N = \{0, 1, \ldots, 2^n - 1\}$ and the set of m-bit integers $M = \{0, 1, \ldots, 2^m - 1\}$. We are counting functions $f : N \to M$ or, equivalently, sequences $\langle a_1, \ldots, a_{2^n} \rangle$ with terms from M. We can choose each term in such a sequence in 2^m ways, so following the rule of product gives us

$$(2^m)^{2^n} = 2^{m2^n}$$

possibilities in total for the number of sequences or the number of n-input, m-output boolean functions. Specifically, there is

$$2^{2^n}$$

n-input, 1-output boolean functions.

C.1-3 Let S_n be the number of ways to arrange n professors around a circular table. A seating is indistinguishable from $n-1$ others that are formed from it by rotation. There are $n!$ permutations of the n professors, so $nS_n = n!$. Hence

$$S_n = (n-1)!.$$

C.1-4 Three integers add up to an even number only if all of them are even, or if exactly one of them is even. There are 49 even and 50 odd numbers in the set $\{1, 2, \ldots, 99\}$, so in the first option we can make a choice in $\binom{49}{3}$ ways, and in the second — in $\binom{50}{2}\binom{49}{1}$ ways. Therefore, the total number of ways is

$$\binom{49}{3} + \binom{50}{2}\binom{49}{1} = 78{,}449.$$

C.1-5

$$\begin{aligned}\binom{n}{k} &= \frac{n!}{k!\,(n-k)!} \\ &= \frac{n}{k} \cdot \frac{(n-1)!}{(k-1)!\,(n-k)!} \\ &= \frac{n}{k}\binom{n-1}{k-1}\end{aligned}$$

C.1-6

$$\begin{aligned}\binom{n}{k} &= \frac{n!}{k!\,(n-k)!} \\ &= \frac{n}{n-k} \cdot \frac{(n-1)!}{k!\,(n-k-1)!} \\ &= \frac{n}{n-k}\binom{n-1}{k}\end{aligned}$$

C.1-7 Let S be an n-set and let $s \in S$. The set S has $\binom{n}{k}$ k-subsets, for $0 \le k \le n$. We can divide these k-subsets into those that do not contain s and those that do contain s. There are $\binom{n-1}{k}$ of the former and $\binom{n-1}{k-1}$ of the latter. Hence, we have

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}.$$

C.1-8 The first rows of Pascal's triangle:

$$
\begin{array}{ccccccc}
1 & & & & & & \\
1 & 1 & & & & & \\
1 & 2 & 1 & & & & \\
1 & 3 & 3 & 1 & & & \\
1 & 4 & 6 & 4 & 1 & & \\
1 & 5 & 10 & 10 & 5 & 1 & \\
1 & 6 & 15 & 20 & 15 & 6 & 1
\end{array}
$$

In the nth row the leftmost element is $\binom{n}{0} = 1$, and the rightmost element is $\binom{n}{n} = 1$. The remaining elements are determined based on the identity from Exercise C.1-7.

C.1-9

$$
\begin{aligned}
\binom{n+1}{2} &= \frac{(n+1)!}{2!\,(n-1)!} \\
&= \frac{n(n+1)}{2} \\
&= \sum_{i=1}^{n} i \qquad \text{(by equation (A.1))}
\end{aligned}
$$

C.1-10 Let's fix n. We'll view the expression $\binom{n}{k}$ as a function $b_n(k)$ for $k = 0, 1, \ldots, n$ and will find k for which the value $b_n(k)$ is the greatest. Consider the ratio:

$$
\begin{aligned}
\frac{b_n(k+1)}{b_n(k)} &= \frac{\binom{n}{k+1}}{\binom{n}{k}} \\
&= \frac{n!}{(k+1)!\,(n-k-1)!} \cdot \frac{k!\,(n-k)!}{n!} \\
&= \frac{n-k}{k+1}.
\end{aligned}
$$

If $n - k \geq k + 1$, or $k \leq (n-1)/2$, then function b_n is monotonically increasing. Conversely, when $k \geq (n-1)/2$, then function b_n is monotonically decreasing. So

when n is odd, b_n reaches its maximum for $k = (n-1)/2 = \lfloor n/2 \rfloor$. Moreover:

$$\begin{aligned} b_n((n-1)/2) &= \binom{n}{(n-1)/2} \\ &= \binom{n}{n-(n-1)/2} && \text{(by equation (C.3))} \\ &= \binom{n}{(n+1)/2} \\ &= b_n((n+1)/2), \end{aligned}$$

so the maximum value is also reached when $k = (n+1)/2 = \lceil n/2 \rceil$.

In case n is an even number, b_n achieves the maximum value for either $k = n/2$ or $k = n/2 - 1$. Let's see which value is bigger:

$$\begin{aligned} \frac{b_n(n/2)}{b_n(n/2-1)} &= \frac{\binom{n}{n/2}}{\binom{n}{n/2-1}} \\ &= \frac{n!}{(n/2)!\,(n/2)!} \cdot \frac{(n/2-1)!\,(n/2+1)!}{n!} \\ &= \frac{n/2+1}{n/2} \\ &> 1. \end{aligned}$$

So we have that b_n reaches its maximum for $k = n/2 = \lfloor n/2 \rfloor = \lceil n/2 \rceil$.

C.1-11 ★ First, let's observe that

$$\begin{aligned} (j+k)! &= j!\,(j+1)(j+2)\cdots k \\ &\geq j!\,1 \cdot 2 \cdots k \\ &= j!\,k!, \end{aligned}$$

where equality holds only if $j = 0$ or $k = 0$. Then

$$\begin{aligned}\binom{n}{j+k} &= \frac{n!}{(j+k)!\,(n-j-k)!} \\ &\le \frac{n!}{j!\,k!\,(n-j-k)!} \\ &= \frac{n!}{j!\,(n-j)!} \cdot \frac{(n-j)!}{k!\,(n-j-k)!} \\ &= \binom{n}{j}\binom{n-j}{k}.\end{aligned}$$

We can interpret the expression $\binom{n}{j+k}$ as the number of possible ways to choose $j+k$ items out of n, and the expression $\binom{n}{j}\binom{n-j}{k}$ as the number of ways to choose j items out of n, and then k items out of $n-j$ left after the first choice. In each strategy we come up with a set of the $j+k$ selected items. There is exactly one way to construct a given set of $j+k$ items using the first strategy, and at least one if we follow the second strategy. It is so, because we can arbitrarily partition the set into the set of j items that will be selected in the first step and the set of k items that will be selected in the second step.

C.1-12 ★ It's easy to check that the inequality holds for $k = 0$. Now let $k \ge 1$ and let's assume that

$$\binom{n}{k-1} \le \frac{n^n}{(k-1)^{k-1}(n-k+1)^{n-k+1}}.$$

Then

$$\begin{aligned}\binom{n}{k} &= \frac{n}{k}\binom{n-1}{k-1} && \text{(by Exercise C.1-5)} \\ &= \frac{n}{k} \cdot \frac{n-k+1}{n}\binom{n}{k-1} && \text{(by Exercise C.1-6)} \\ &\le \frac{n^n}{k(k-1)^{k-1}(n-k+1)^{n-k}} && \text{(by the inductive hypothesis).}\end{aligned}$$

Now it suffices to show that

$$\frac{n^n}{k(k-1)^{k-1}(n-k+1)^{n-k}} \le \frac{n^n}{k^k(n-k)^{n-k}}.$$

To prove this, we examine the ratio of both expressions:

$$\begin{aligned}\frac{\frac{n^n}{k(k-1)^{k-1}(n-k+1)^{n-k}}}{\frac{n^n}{k^k(n-k)^{n-k}}} &= \frac{k^{k-1}(n-k)^{n-k}}{(k-1)^{k-1}(n-k+1)^{n-k}} \\ &= \frac{\left(\frac{k}{k-1}\right)^{k-1}}{\left(\frac{n-k+1}{n-k}\right)^{n-k}} \\ &= \frac{\left(1+\frac{1}{k-1}\right)^{k-1}}{\left(1+\frac{1}{n-k}\right)^{n-k}}.\end{aligned}$$

The sequence $e_n = (1+1/n)^n$ is increasing — therefore the above ratio does not exceed 1, as long as $k-1 \le n-k$ or, equivalently, $k \le (n+1)/2$, and in particular when $k \le n/2$. Thus, we have shown that inequality (C.7) holds for $k \le n/2$.

When $n/2 \le k \le n$, then $0 \le n-k \le n/2$, so

$$\begin{aligned}\binom{n}{k} &= \binom{n}{n-k} && \text{(by equation (C.3))} \\ &\le \frac{n^n}{(n-k)^{n-k}k^k},\end{aligned}$$

where the last inequality holds after substituting k by $n-k$ and applying the result from the first part of the exercise.

C.1-13 ★ Using Stirling's approximation (3.25), we obtain

$$\begin{aligned}\binom{2n}{n} &= \frac{(2n)!}{(n!)^2} \\ &= \frac{\sqrt{4\pi n}\,(2n/e)^{2n}(1+\Theta(1/n))}{2\pi n\,(n/e)^{2n}(1+\Theta(1/n))^2} \\ &= \frac{2^{2n}}{\sqrt{\pi n}} \cdot \frac{1+\Theta(1/n)}{(1+\Theta(1/n))^2}.\end{aligned}$$

Now let's examine the last fraction, which we intentionally didn't reduce because the denominator may not be equal to the square of the numerator. Let c, d be constants such that $c \ge d > 0$, the expression $1+c/n$ bounds the numerator from above, and

the expression $1 + d/n$ bounds the denominator from below. Then

$$\begin{aligned}\frac{1+\Theta(1/n)}{(1+\Theta(1/n))^2} &\le \frac{1+c/n}{(1+d/n)^2}\\ &< \frac{1+c/n}{1+d/n}\\ &= \frac{n+c}{n+d}\\ &= 1+\frac{c-d}{n+d}\\ &< 1+\frac{c-d}{n}\\ &= 1+O(1/n).\end{aligned}$$

Therefore,

$$\binom{2n}{n} = \frac{2^{2n}}{\sqrt{\pi n}}\,(1+O(1/n)).$$

C.1-14 ★ Let's differentiate the binary entropy function H:

$$\begin{aligned}H'(\lambda) &= -\left(\lg\lambda + \lambda\cdot\frac{1}{\lambda\ln 2}\right) - \left(-\lg(1-\lambda) + (1-\lambda)\cdot\frac{-1}{(1-\lambda)\ln 2}\right)\\ &= -\lg\lambda - \lg e + \lg(1-\lambda) + \lg e\\ &= \lg\frac{1-\lambda}{\lambda},\\ H''(\lambda) &= \frac{\lambda}{(1-\lambda)\ln 2}\cdot\frac{-\lambda-(1-\lambda)}{\lambda^2}\\ &= -\frac{\lg e}{\lambda(1-\lambda)}.\end{aligned}$$

The first derivative reaches zero when $\lambda = 1/2$. For $\lambda = 1/2$ the second derivative is negative, so the function H achieves its maximum of $H(1/2) = 1$.

C.1-15 ★ For $n = 0$ the equality holds trivially, so let $n \geq 1$. Then

$$\begin{aligned}\sum_{k=0}^{n} \binom{n}{k} k &= 0 + \sum_{k=1}^{n} \binom{n-1}{k-1} n && \text{(by identity (C.9))} \\ &= n \sum_{k=0}^{n-1} \binom{n-1}{k} \\ &= n 2^{n-1} && \text{(by indentity (C.4) where } x = y = 1).\end{aligned}$$

C.1-16 ★ We can enumerate the cases with $n = 0, 1, 2, 3$ and $k = 0, 1$, and confirm that the identity holds for all of them.

Now let $n \geq 4$. We have

$$\begin{aligned}\binom{n}{k} &= \frac{n(n-1)\cdots(n-(k-1))}{k!} \\ &= \frac{n^k}{k!}\left(1 - \frac{1}{n}\right)\cdots\left(1 - \frac{k-1}{n}\right) \\ &\geq \frac{n^k}{k!}\left(1 - \frac{k-1}{n}\right)^{k-1}.\end{aligned}$$

Now consider the function

$$f(x, y) = \left(1 - \frac{x}{n}\right)^y,$$

where $0 \leq x \leq n$ and $y \geq 0$. Observe that the function f monotonically decreases, when x increases for a fixed y, or when y increases for a fixed x. Therefore, since $k - 1 < \sqrt{n}$,

$$\begin{aligned}\left(1 - \frac{k-1}{n}\right)^{k-1} &\geq \left(1 - \frac{\sqrt{n}}{n}\right)^{k-1} \\ &\geq \left(1 - \frac{1}{\sqrt{n}}\right)^{\sqrt{n}}.\end{aligned}$$

It is true that

$$\left(1 - \frac{1}{t}\right)^t \geq \frac{1}{4}$$

for all $t \geq 2$. Therefore, when $n \geq 4$,

$$\begin{aligned}\binom{n}{k} &\geq \frac{n^k}{k!}\left(1 - \frac{1}{\sqrt{n}}\right)^{\sqrt{n}} \\ &\geq \frac{n^k}{4k!}.\end{aligned}$$

C.2 Probability

C.2-1 Let's represent the outcome of the experiment as a 3-string over $\{\text{H}, \text{T}\}$, with the first two elements denoting the results of Professor Rosencrantz's both tosses, and the last element denoting the result of Professor Guildenstern's toss. The sample space consists of $2^3 = 8$ elementary events. The probability that Professor Rosencrantz obtains strictly more heads than Professor Guildenstern is

$$\begin{aligned}\Pr\{\text{HHH}, \text{HHT}, \text{HTT}, \text{THT}\} &= \Pr\{\text{HHH}\} + \Pr\{\text{HHT}\} + \Pr\{\text{HTT}\} + \Pr\{\text{THT}\} \\ &= 1/2.\end{aligned}$$

C.2-2 Let $C_1, C_2, \ldots$ be a finite or countably infinite sequence of events, where

$$C_i = A_i - \bigcup_{j=1}^{i-1} A_j$$

for every $i = 1, 2, \ldots$. Because $\bigcup_i A_i = \bigcup_i C_i$ and any two events C_j, C_k, where $j \neq k$, are mutually exclusive, we get

$$\begin{aligned}\Pr\left\{\bigcup_i A_i\right\} &= \Pr\left\{\bigcup_i C_i\right\} \\ &= \sum_i \Pr\{C_i\} \\ &\leq \sum_i \Pr\{A_i\}.\end{aligned}$$

The last inequality holds, since $\Pr\{C_i\} \leq \Pr\{A_i\}$ for every $i = 1, 2, \ldots$.

C.2-3 Let n_1, n_2, n_3 be the numbers on the selected cards, in order of removing the cards from the deck. Consider the events $A = \{n_1 < n_2\}$ and $B = \{n_2 < n_3\}$. If A occurs, then n_1 could have been chosen in $n_2 - 1$ ways. Similarly, if B occurs, then n_3 can be

chosen in $10 - n_2$ ways. We need to calculate $\Pr\{A \cap B\}$. The number of outcomes in $A \cap B$ is

$$\sum_{n_2=1}^{10} (n_2 - 1)(10 - n_2) = 120.$$

The sample space consists of all possible ordered triples of cards (3-permutations) selected from the deck. By (C.1), the sample space has the size of $10!/7! = 720$. Thus,

$$\begin{aligned}\Pr\{A \cap B\} &= 120/720 \\ &= 1/6.\end{aligned}$$

C.2-4 Since $(A \cap B) \cup (\overline{A} \cap B) = B$, and since events $A \cap B$ and $\overline{A} \cap B$ are mutually exclusive, we get

$$\begin{aligned}\Pr\{A \mid B\} + \Pr\left\{\overline{A} \mid B\right\} &= \frac{\Pr\{A \cap B\}}{\Pr\{B\}} + \frac{\Pr\left\{\overline{A} \cap B\right\}}{\Pr\{B\}} \\ &= \frac{\Pr\{B\}}{\Pr\{B\}} \\ &= 1.\end{aligned}$$

C.2-5 We prove by induction on the number of events n. For $n = 1$ the equality trivially holds, so let's assume $n \geq 2$. We get

$$\begin{aligned}\Pr\left\{\bigcap_{i=1}^{n} A_i\right\} &= \Pr\left\{A_n \cap \bigcap_{i=1}^{n-1} A_i\right\} \\ &= \Pr\left\{\bigcap_{i=1}^{n-1} A_i\right\} \Pr\left\{A_n \;\middle|\; \bigcap_{i=1}^{n-1} A_i\right\} \qquad \text{(by (C.16))} \\ &= \Pr\{A_1\} \Pr\{A_2 \mid A_1\} \Pr\{A_3 \mid A_1 \cap A_2\} \cdots \Pr\left\{A_n \;\middle|\; \bigcap_{i=1}^{n-1} A_i\right\},\end{aligned}$$

where the last step follows from the inductive hypothesis.

C.2-6 ★ Let $n \geq 3$ and let $S = \{s_{pq} : 1 \leq p, q \leq n\}$ be a sample space with the uniform probability distribution (i.e., $\Pr\{s_{pq}\} = 1/n^2$ for every $p = 1, 2, \ldots, n$ and $q = 1, 2, \ldots, n$). For each $i = 1, 2, \ldots, n$ consider the event

$$A_i = \{s_{pi} \in S : 1 \leq p \leq i\} \cup \{s_{iq} \in S : i < q \leq n\}.$$

Then $|A_i| = n$, so $\Pr\{A_i\} = 1/n$. Also,

$$|A_{i_1} \cap A_{i_2}| = 1$$

and

$$|A_{i_1} \cap A_{i_2} \cap A_{i_3}| = 0$$

for every distinct indices $1 \le i_1, i_2, i_3 \le n$. Therefore,

$$\begin{aligned}\Pr\{A_{i_1} \cap A_{i_2}\} &= 1/n^2 \\ &= \Pr\{A_{i_1}\} \Pr\{A_{i_2}\},\end{aligned}$$

and so the events $A_1, A_2, \ldots, A_n$ are pairwise independent. On the other hand, for each $k > 2$ and distinct indices $1 \le i_1, \ldots, i_k \le n$,

$$\Pr\left\{\bigcap_{j=1}^{k} A_{i_j}\right\} = 0,$$

while

$$\prod_{j=1}^{k} \Pr\{A_{i_j}\} = 1/n^k,$$

so no k-subset of the events $A_1, A_2, \ldots, A_n$ is mutually independent.

C.2-7 ★ Suppose we have two coins — a fair one and a biased one that always comes up heads. We will randomly pick one of the coins and flip it twice. Let A be the event, that the first flip resulted in heads, let B be the event that the second flip resulted in heads, and let C be the event that the fair coin was picked. Then,

$$\begin{aligned}\Pr\{A\} &= \Pr\{B\} \\ &= (1/2) \cdot (1/2) + (1/2) \cdot 1 \\ &= 3/4,\end{aligned}$$

$$\Pr\{C\} = 1/2.$$

The events A and B are not independent, since

$$\begin{aligned}\Pr\{A \cap B\} &= (1/2) \cdot (1/4) + (1/2) \cdot 1 \\ &= 5/8,\end{aligned}$$

while $\Pr\{A\}\Pr\{B\} = 9/16$. But A and B are conditionally independent, given C:

$$\begin{aligned}\Pr\{A \cap B \mid C\} &= 1/4 \\ &= (1/2) \cdot (1/2) \\ &= \Pr\{A \mid C\}\Pr\{B \mid C\}.\end{aligned}$$

C.2-8 ★ Let J, T, and C be the events that Jeff, Tim and Carmine will pass the course, respectively. Furthermore, let G be the event that Professor Gore pointed out that Jeff is the one who will fail. Since the professor can't reveal Carmine's outcome, there must be $\Pr\{G \mid J\} = 0$, $\Pr\{G \mid T\} = 1$, and $\Pr\{G \mid C\} = 1/2$. Before talking to the professor, Carmine knows that his probability of passing is $\Pr\{C\} = 1/3$. His situation after the conversation is described by the event $C \mid G$.

Since $G = (G \cap J) \cup (G \cap T) \cup (G \cap C)$ and since $G \cap J$, $G \cap T$ and $G \cap C$ are mutually exclusive events,

$$\begin{aligned}\Pr\{G\} &= \Pr\{G \cap J\} + \Pr\{G \cap T\} + \Pr\{G \cap C\} \\ &= \Pr\{J\}\Pr\{G \mid J\} + \Pr\{T\}\Pr\{G \mid T\} + \Pr\{C\}\Pr\{G \mid C\}.\end{aligned}$$

Using Bayes's theorem, we obtain

$$\begin{aligned}\Pr\{C \mid G\} &= \frac{\Pr\{C\}\Pr\{G \mid C\}}{\Pr\{G\}} \\ &= \frac{\Pr\{C\}\Pr\{G \mid C\}}{\Pr\{J\}\Pr\{G \mid J\} + \Pr\{T\}\Pr\{G \mid T\} + \Pr\{C\}\Pr\{G \mid C\}} \\ &= \frac{(1/3) \cdot (1/2)}{(1/3) \cdot 0 + (1/3) \cdot 1 + (1/3) \cdot (1/2)} \\ &= 1/3.\end{aligned}$$

Thus, the information that Carmine has obtained does not change his chance of passing.

C.3 Discrete random variables

C.3-1 The expected sum of the two values showing on two 6-sided dice has already been calculated in Exercise 5.2-4 and equals 7.

Define the random variable Y to be the maximum of the values showing. For each $y = 1, 2, \ldots, 6$, the value y is the maximum of the values on both dice, when one die has value y, and the other die has value at most y. This happens for exactly $2y - 1$

outcomes. Thus,

$$\begin{aligned}
\mathrm{E}[Y] &= \sum_{y=1}^{6} y \Pr\{Y = y\} \\
&= (1/36) \cdot (1 \cdot 1 + 2 \cdot 3 + 3 \cdot 5 + 4 \cdot 7 + 5 \cdot 9 + 6 \cdot 11) \\
&= 161/36 \\
&\approx 4.47.
\end{aligned}$$

C.3-2 Let X be the random variable denoting the index of the maximum element in the array $A[1:n]$. For every $i = 1, 2, \ldots, n$, $\Pr\{X = i\} = 1/n$, and therefore

$$\begin{aligned}
\mathrm{E}[X] &= \sum_{i=1}^{n} i \Pr\{X = i\} \\
&= \frac{1}{n} \sum_{i=1}^{n} i \\
&= \frac{1}{n} \cdot \frac{n(n+1)}{2} \\
&= \frac{n+1}{2}.
\end{aligned}$$

In fact, this result is identical for each element of the array, in particular for the maximum element and for the minimum element.

C.3-3 Let X be the random variable representing the player's gain (in dollars) from playing the carnival game once. For $k = 0, 1, 2, 3$, let A_k be the event that the player's number appeared on exactly k dice. Then

$$\begin{aligned}
\Pr\{A_k\} &= \binom{3}{k} \left(\frac{1}{6}\right)^k \left(\frac{5}{6}\right)^{3-k} \\
&= \binom{3}{k} \frac{5^{3-k}}{6^3}
\end{aligned}$$

We have that

$$\begin{aligned}
\mathrm{E}[X] &= (-1) \cdot \Pr\{A_0\} + 1 \cdot \Pr\{A_1\} + 2 \cdot \Pr\{A_2\} + 3 \cdot \Pr\{A_3\} \\
&= (1/6^3) \cdot ((-1) \cdot 1 \cdot 125 + 1 \cdot 3 \cdot 25 + 2 \cdot 3 \cdot 5 + 3 \cdot 1 \cdot 1) \\
&= -17/216 \\
&\approx -0.08,
\end{aligned}$$

so the player will lose about 8 cents on average in this game.

C.3-4 Consider the subset of the sample space, where $X \geq Y$ holds. Because $Y \geq 0$, then $\mathrm{E}[Y] \geq 0$, and so

$$\begin{aligned}\mathrm{E}[\max\{X, Y\}] &= \mathrm{E}[X] \\ &\leq \mathrm{E}[X] + \mathrm{E}[Y].\end{aligned}$$

The case in which $Y \geq X$ is similar.

C.3-5 ★ According to the definition of independent random variables X and Y, for all x and y, we have

$$\Pr\{X = x \text{ and } Y = y\} = \Pr\{X = x\}\Pr\{Y = y\}.$$

Let f and g be any functions over the reals. If $X = x$, then $f(X) = f(x)$, and similarly, if $Y = y$, then $g(Y) = g(y)$. Therefore, we can write the above formula as

$$\Pr\{f(X) = f(x) \text{ and } g(Y) = g(y)\} = \Pr\{f(X) = f(x)\}\Pr\{g(Y) = g(y)\},$$

from which we conclude that $f(X)$ and $g(Y)$ are independent random variables.

C.3-6 ★ For any $t > 0$, we have

$$\begin{aligned}\mathrm{E}[X] &= \sum_{x \geq 0} x \Pr\{X = x\} \\ &= \sum_{0 \leq x < t} x \Pr\{X = x\} + \sum_{x \geq t} x \Pr\{X = x\} \\ &\geq \sum_{x \geq t} x \Pr\{X = x\} \\ &\geq t \sum_{x \geq t} \Pr\{X = x\} \\ &= t \Pr\{X \geq t\}.\end{aligned}$$

C.3-7 ★ Let's pick any real t. Let $A = \{s \in S : X(s) \geq t\}$ and $A' = \{s \in S : X'(s) \geq t\}$. It follows from the relationship between random variables X and X', that if $X'(s) \geq t$,

then $X(s) \geq t$, so $A' \subseteq A$. From the definition of a random variable, we have

$$\begin{aligned}\Pr\{X \geq t\} &= \sum_{s \in A} \Pr\{s\} \\ &= \sum_{s \in A'} \Pr\{s\} + \sum_{s \in A - A'} \Pr\{s\} \\ &\geq \sum_{s \in A'} \Pr\{s\} \\ &= \Pr\left\{X' \geq t\right\}.\end{aligned}$$

C.3-8 Let $f(x) = x^2$ and let $0 \leq \lambda \leq 1$. Then

$$\begin{aligned}f(\lambda x + (1-\lambda)y) - (\lambda f(x) + (1-\lambda) f(y)) \\ &= \lambda^2 x^2 + (1-\lambda)^2 y^2 + 2\lambda(1-\lambda)xy - \lambda x^2 - (1-\lambda)y^2 \\ &= -\lambda(1-\lambda)x^2 - \lambda(1-\lambda)y^2 + 2\lambda(1-\lambda)xy \\ &= -\lambda(1-\lambda)(x-y)^2 \\ &\leq 0,\end{aligned}$$

which means that the function $f(x)$ is convex. By applying it to Jensen's inequality, we obtain

$$\mathrm{E}\left[X^2\right] \geq \mathrm{E}^2[X].$$

C.3-9 Since X takes on only the values 0 and 1, we have $X^2 = X$. Then,

$$\begin{aligned}\mathrm{Var}[X] &= \mathrm{E}\left[X^2\right] - \mathrm{E}^2[X] \\ &= \mathrm{E}[X] - \mathrm{E}^2[X] \\ &= \mathrm{E}[X](1 - \mathrm{E}[X]) \\ &= \mathrm{E}[X]\,\mathrm{E}[1 - X] \qquad \text{(by linearity of expectation).}\end{aligned}$$

C.3-10

$$\begin{aligned}\mathrm{Var}[aX] &= \mathrm{E}\left[a^2X^2\right] - \mathrm{E}^2[aX] \\ &= a^2\,\mathrm{E}\left[X^2\right] - a^2\,\mathrm{E}^2[X] \qquad \text{(by equation (C.25))} \\ &= a^2\left(\mathrm{E}\left[X^2\right] - \mathrm{E}^2[X]\right) \\ &= a^2\,\mathrm{Var}[X]\end{aligned}$$

C.4 The geometric and binomial distributions

C.4-1 Let X be a random variable representing the number of trials before obtaining a success in a sequence of Bernoulli trials, where a success occurs with probability $p > 0$ and a failure with probability $q = 1 - p$. Then,

$$\begin{aligned}\sum_{k=1}^{\infty} \Pr\{X = k\} &= \sum_{k=1}^{\infty} q^{k-1} p \\ &= p \sum_{k=0}^{\infty} q^k \\ &= \frac{p}{1-q} && \text{(by equation (A.7))} \\ &= 1,\end{aligned}$$

so axiom 2 of the probability axioms holds.

C.4-2 We have a success when of all six coins, any three came up heads and the other three came up tails — it is therefore $\binom{6}{3} = 20$ ways to obtain a success. There are $2^6 = 64$ possible outcomes, so the probability of success is $p = 20/64 = 5/16$. By (C.36), before obtaining one we need to make $1/p \approx 3.2$ flips on average.

C.4-3 Let X be a random variable having a geometric distribution with the probability $p > 0$ of a success and the probability $q = 1 - p$ of a failure. First, let's find the expectation of X^2:

$$\begin{aligned}\mathrm{E}\left[X^2\right] &= \sum_{k=1}^{\infty} k^2 \Pr\{X = k\} \\ &= \sum_{k=1}^{\infty} k^2 q^{k-1} p \\ &= \frac{p}{q} \sum_{k=0}^{\infty} k^2 q^k \\ &= \frac{p}{q} \cdot \frac{q(1+q)}{(1-q)^3} && \text{(by Exercise A.1-6)} \\ &= (1+q)/p^2.\end{aligned}$$

From the definition (C.31) of variance and equation (C.36), we have

$$\begin{aligned}\mathrm{Var}[X] &= \mathrm{E}\left[X^2\right] - \mathrm{E}^2[X] \\ &= (1+q)/p^2 - 1/p^2 \\ &= q/p^2.\end{aligned}$$

C.4-4

$$\begin{aligned}b(k;n,p) &= \binom{n}{k} p^k (1-p)^{n-k} \\ &= \binom{n}{n-k} q^{n-k}(1-q)^k && \text{(by equation (C.3))} \\ &= b(n-k;n,q)\end{aligned}$$

C.4-5

The binomial distribution achieves a maximum at an integer k, such that $np - q \le k \le (n+1)p$, so a good approximation of the maximum is the value for $k = np$. Since np may not be integer, we'll sacrifice mathematical rigor in order to simplify our calculations:

$$\begin{aligned}b(np;n,p) &= \binom{n}{np} p^{np}(1-p)^{n-np} \\ &= \frac{n!}{(np)!\,(n-np)!}\, p^{np}(1-p)^{n-np} \\ &= \frac{n!}{(np)!\,(nq)!}\, p^{np} q^{nq}.\end{aligned}$$

Using Stirling's approximation (3.25), we can simplify the first factor of the last expression above:

$$\begin{aligned}\frac{n!}{(np)!\,(nq)!} &\approx \frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n}{\sqrt{2\pi np}\left(\frac{np}{e}\right)^{np}\sqrt{2\pi nq}\left(\frac{nq}{e}\right)^{nq}} \\ &= \frac{\left(\frac{n}{e}\right)^n \left(\frac{e}{np}\right)^{np}\left(\frac{e}{nq}\right)^{nq}}{\sqrt{2\pi npq}} \\ &= \frac{1}{p^{np}q^{nq}\sqrt{2\pi npq}}.\end{aligned}$$

Hence, the value of the maximum of $b(k;n,p)$ is roughly equal to $1/\sqrt{2\pi npq}$.

C.4-6 ★ We are interested in the value of $b(0; n, 1/n)$ and we can approximate it by examining its limit:

$$\begin{aligned}\lim_{n\to\infty} b(0; n, 1/n) &= \lim_{n\to\infty} (1 - 1/n)^n \\ &= 1/e \qquad \text{(by equation (3.16)).}\end{aligned}$$

In the second part we similarly approximate $b(1; n, 1/n)$:

$$\begin{aligned}\lim_{n\to\infty} b(1; n, 1/n) &= \lim_{n\to\infty} \frac{(1 - 1/n)^n}{1 - 1/n} \\ &= \frac{1/e}{1} \qquad \text{(by equation (3.16))} \\ &= 1/e.\end{aligned}$$

C.4-7 ★ We'll calculate the probability that the professors get the same number of heads in two ways. In the first one, we'll treat the result of each experiment as a $2n$-element sequence of heads and tails, such that its initial n terms are the results obtained by Professor Rosencrantz, and the final n terms are the results obtained by Professor Guildenstern. There are $2^{2n} = 4^n$ of all such sequences. Following the hint, we call a head a success for Professor Rosencrantz and we call a tail a success for Professor Guildenstern. Note that both professors obtain the same number of heads if and only if they achieve exactly n successes in total. The number of ways this can be done is the number of choices of n items responsible for successes among all $2n$ items in the sequence. This number is equal to $\binom{2n}{n}$, so the probability we are looking for is $\binom{2n}{n}/4^n$.

Another approach to determine the desired probability, is to define random variables R and G denoting the number of heads obtained by Professor Rosencrantz and by Professor Guildenstern, respectively. Again, defining successes for each professor according to the hint, we get that R has the binomial distribution $b(k; n, 1/2)$ and G has the binomial distribution $b(n - k; n, 1/2)$. The events $R = k$ and $G = k$ are

independent, so the probability we are looking for, is

$$\begin{aligned}\sum_{k=0}^{n} \Pr\{R = k \text{ and } G = k\} &= \sum_{k=0}^{n} \Pr\{R = k\} \Pr\{G = k\} \\ &= \sum_{k=0}^{n} b(k; n, 1/2)\, b(n-k; n, 1/2) \\ &= \sum_{k=0}^{n} \binom{n}{k} \left(\frac{1}{2}\right)^{k} \left(\frac{1}{2}\right)^{n-k} \binom{n}{n-k} \left(\frac{1}{2}\right)^{n-k} \left(\frac{1}{2}\right)^{k} \\ &= \frac{\sum_{k=0}^{n} \binom{n}{k}^2}{4^n}.\end{aligned}$$

Comparing the results obtained in both ways, we get the identity

$$\sum_{k=0}^{n} \binom{n}{k}^2 = \binom{2n}{n}.$$

C.4-8 ★ Let $0 \le \lambda \le 1$. Using the inequality

$$\binom{n}{\lambda n} \le 2^{nH(\lambda)},$$

that follows from (C.7), we obtain

$$\begin{aligned}b(k; n, 1/2) &= \binom{n}{k} \left(\frac{1}{2}\right)^{k} \left(\frac{1}{2}\right)^{n-k} \\ &= \binom{n}{(k/n)\, n} \left(\frac{1}{2}\right)^{n} \\ &\le \frac{2^{nH(k/n)}}{2^n} \\ &= 2^{nH(k/n)-n}.\end{aligned}$$

C.4-9 ★ Let X' be the random variable denoting the number of successes in a series of Bernoulli trials, each with a probability p of success. Using the result from Exercise C.4-10, where $p'_i = p$ for every $i = 1, 2, \ldots, n$, gives

$$\Pr\{X \ge k\} \le \Pr\left\{X' \ge k\right\}.$$

The random variable X' has a binomial distribution, so the above inequality can be reformulated as

$$\Pr\{X \geq k\} \leq \sum_{i=k}^{n} b(i; n, p).$$

Then,

$$\begin{aligned}\Pr\{X < k\} &= 1 - \Pr\{X \geq k\} \\ &\geq \sum_{i=0}^{n} b(i; n, p) - \sum_{i=k}^{n} b(i; n, p) \qquad \text{(by equation (C.40))} \\ &= \sum_{i=0}^{k-1} b(i; n, p).\end{aligned}$$

C.4-10 ★ Let S be the sample space containing all possible sequences of outcomes during n Bernoulli trials. A set A corresponds to a sequence $s \in S$, and $X(s)$ is the number of successes in s.

We'll show how to obtain a set A' of Bernoulli trials by performing experiments on the trials of A. Let A_i denote the event that the ith trial of A is a success, and let A'_i be the event that the ith trial of A' is a success. If A_i occurs, we'll make A'_i to also occur. Otherwise, a success in the ith trial of A' will be generated with some probability r_i. Then,

$$\begin{aligned}p'_i &= \Pr\{A'_i\} \\ &= \Pr\{A'_i \cap A_i\} + \Pr\{A'_i \cap \overline{A_i}\} \\ &= p_i + (1 - p_i) \cdot r_i,\end{aligned}$$

and so,

$$r_i = \frac{p'_i - p_i}{1 - p_i}.$$

Reusing the sample space S, we'll denote by $X'(s)$ the number of successes in a series of the n trials obtained by following the above procedure, based on the outcomes in s. For any $s \in S$ and its corresponding set A, it's clear that after constructing the set A' this way, it's impossible that there are fewer successes in A' than in the original set A. In other words, $X'(s) \geq X(s)$ holds. Using Exercise C.3-7, we get the desired result.

⋆ C.5 The tails of the binomial distribution

C.5-1 ⋆ Intuitively, the situation when each flip in a series results in a head is less likely when the series consists of twice more flips, while we expect the same number of heads as before. Let's prove our intuition. Let X_m for $m \geq 0$ be the random variable denoting the number of heads obtained in m flips of a fair coin. When $n \leq m$, $\Pr\{X_m = n\} = b(n; m, 1/2)$. Thus, the ratio of both studied probabilities is

$$\begin{aligned}
\frac{\Pr\{X_{2n} = n\}}{\Pr\{X_n = n\}} &= \frac{\binom{2n}{n}\left(\frac{1}{2}\right)^{2n}}{\binom{n}{n}\left(\frac{1}{2}\right)^{n}} \\
&= \frac{\binom{2n}{n}}{2^n} \\
&= \frac{2^n}{\sqrt{\pi n}}\,(1 + O(1/n)) && \text{(by equation (C.11))} \\
&> 1,
\end{aligned}$$

which confirms our guess.

C.5-2 ⋆ ***Proof of Corollary C.6*** Let Y be the random variable denoting the total number of failures in this experiment. Then, $\Pr\{X > k\} = \Pr\{Y < n - k\}$. Since $np < k < n$, or $0 < n - k < nq$, we can apply Theorem C.4 for Y, after inverting the roles of successes and failures, to obtain the desired inequality. ■

Proof of Corollary C.7 Similarly as in the previous proof, let's treat the probability of getting more than k successes as the probability of getting fewer than $n - k$ failures. Since $(np + n)/2 < k < n$, we have $0 < n - k < nq/2$, and after swapping the roles of successes and failures, the result follows from Corollary C.5. ■

C.5-3 ⋆ Let $p = a/(a + 1)$ and $q = 1 - p = 1/(a + 1)$, so that $a = p/q$. Then,

$$\begin{aligned}
\sum_{i=0}^{k-1} \binom{n}{i} a^i &= \sum_{i=0}^{k-1} \binom{n}{i} \left(\frac{p}{q}\right)^i \\
&= \frac{\sum_{i=0}^{k-1} \binom{n}{i} p^i q^{n-i}}{q^n} \\
&= \frac{\sum_{i=0}^{k-1} b(i; n, p)}{q^n},
\end{aligned}$$

so by applying Theorem C.4 to the last summation, we obtain

$$\begin{aligned}\sum_{i=0}^{k-1}\binom{n}{i}a^i &< \frac{kq}{q^n(np-k)}\,b(k;n,p)\\ &= \frac{k/(a+1)}{(1/(a+1))^n(na/(a+1)-k)}\,b(k;n,a/(a+1))\\ &= (a+1)^n\,\frac{k}{na-k(a+1)}\,b(k;n,a/(a+1)).\end{aligned}$$

C.5-4 ★ Using the fact that $\binom{n}{i} \geq 1$ for all $i = 0, 1, \ldots, n$, we have

$$\begin{aligned}\sum_{i=0}^{k-1} p^i q^{n-i} &\leq \sum_{i=0}^{k-1}\binom{n}{i}p^i q^{n-i}\\ &= \sum_{i=0}^{k-1} b(i;n,p)\\ &< \frac{kq}{np-k}\,b(k;n,p) && \text{(by Theorem C.4)}\\ &\leq \frac{kq}{np-k}\left(\frac{np}{k}\right)^k\left(\frac{nq}{n-k}\right)^{n-k} && \text{(by Lemma C.1).}\end{aligned}$$

C.5-5 ★ Let $Y = n - X$ and let $\nu = \mathrm{E}[Y]$. By linearity of expectation, $\nu = n - \mathrm{E}[X] = n - \mu$. Since $r > \nu$, we can use Theorem C.8 to obtain

$$\begin{aligned}\Pr\{\mu - X \geq r\} &= \Pr\{n - \nu - X \geq r\}\\ &= \Pr\{Y - \nu \geq r\}\\ &\leq \left(\frac{\nu e}{r}\right)^r\\ &= \left(\frac{(n-\mu)e}{r}\right)^r.\end{aligned}$$

Similarly, in the second part, if we let $Y = n - X$, then $\mathrm{E}[Y] = n - \mathrm{E}[X] = n - np = nq$. Since $r > nq$, we use Corollary C.9 to get

$$\begin{aligned}\Pr\{np - X \geq r\} &= \Pr\{n - nq - X \geq r\}\\ &= \Pr\{Y - nq \geq r\}\\ &\leq \left(\frac{nqe}{r}\right)^r.\end{aligned}$$

C.5-6 ★ *Lemma 4*

Let α, p, and q be nonnegative reals, such that $p + q = 1$. Then,

$$pe^{\alpha q} + qe^{-\alpha p} \leq e^{\alpha^2/2}.$$

Proof Consider the function $f(\alpha) = e^{\alpha^2/2} - (pe^{\alpha q} + qe^{-\alpha p})$ for $\alpha \geq 0$, and examine its derivatives:

$$f'(\alpha) = \alpha e^{\alpha^2/2} - pq(e^{\alpha q} - e^{-\alpha p}),$$
$$f''(\alpha) = \alpha^2 e^{\alpha^2/2} + e^{\alpha^2/2} - pq(qe^{\alpha q} + pe^{-\alpha p}).$$

It's true that $f''(0) = 1 - pq \geq 0$, and we'll show that $f''(\alpha) > 0$ for all $\alpha > 0$. Since

$$\begin{aligned} pq - 1/4 &= p(1-p) - 1/4 \\ &= -p^2 + p - 1/4 \\ &= -(p - 1/2)^2 \\ &\leq 0 \end{aligned}$$

or, equivalently, since $pq \leq 1/4$,

$$\begin{aligned} pq(qe^{\alpha q} + pe^{-\alpha p}) &\leq (qe^{\alpha q} + pe^{-\alpha p})/4 \\ &\leq (e^{\alpha q} + e^{-\alpha p})/4 \\ &= e^{-\alpha p}(e^{\alpha q + \alpha p} + 1)/4 \\ &= e^{-\alpha p}(e^{\alpha} + 1)/4 \\ &\leq (e^{\alpha} + 1)/4. \end{aligned}$$

Combining this inequality with $\alpha^2 e^{\alpha^2/2} > 0$, that holds for all $\alpha > 0$, we get

$$f''(\alpha) > e^{\alpha^2/2} - (e^{\alpha} + 1)/4.$$

Now it suffices to show that $4e^{\alpha^2/2} \geq e^{\alpha} + 1$ or, since $e^{\alpha^2/2} > 1$, that

$$3e^{\alpha^2/2} \geq e^{\alpha}.$$

This is equivalent to $3e^{\alpha^2/2}/e^{\alpha} = 3e^{\alpha^2/2 - \alpha} \geq 1$, which holds when

$$\alpha^2/2 - \alpha + \ln 3 \geq 0.$$

The left-hand side expression attains the minimum at $\alpha = 1$, equal to $\ln 3 - 1/2 > 0$, which concludes the proof that $f''(\alpha) \geq 0$ for $\alpha \geq 0$.

The fact we've just shown implies that the function $f'(\alpha)$ is monotonically increasing for all $\alpha \geq 0$. Since $f'(0) = 0$, it must be $f'(\alpha) \geq 0$ for all $\alpha \geq 0$, which in turn implies that the function $f(\alpha)$ is monotonically increasing. And since $f(0) = 0$, it holds that $f(\alpha) \geq 0$ for all $\alpha \geq 0$, which completes the proof. ■

Using the same notations as in the proof of Theorem C.8, we have

$$\begin{aligned}
\mathrm{E}\left[e^{\alpha(X-\mu)}\right] &= \prod_{i=1}^{n} \mathrm{E}\left[e^{\alpha(X_i-p_i)}\right] && \text{(from the proof on page 1207)} \\
&= \prod_{i=1}^{n} \left(e^{\alpha(1-p_i)}p_i + e^{\alpha(0-p_i)}q_i\right) \\
&= \prod_{i=1}^{n} (p_i e^{\alpha q_i} + q_i e^{-\alpha p_i}) \\
&\le \prod_{i=1}^{n} e^{\alpha^2/2} && \text{(by Lemma 4 for } p = p_i \text{ and } q = q_i) \\
&= \exp(\alpha^2 n/2).
\end{aligned}$$

Then,

$$\begin{aligned}
\Pr\{X - \mu \ge r\} &= \Pr\left\{e^{\alpha(X-\mu)} \ge e^{\alpha r}\right\} && \text{(by equation (C.47))} \\
&\le \mathrm{E}\left[e^{\alpha(X-\mu)}\right] e^{-\alpha r} && \text{(by inequality (C.48))} \\
&\le \exp(\alpha^2 n/2 - \alpha r).
\end{aligned}$$

Now we need to choose the value of α that minimizes the last expression. The argument of the exponential function in that expression is a quadratic function with respect to α, so it's easy to show that it attains a minimum at $\alpha = r/n$. We finally get

$$\begin{aligned}
\Pr\{X - \mu \ge r\} &\le \exp\left((r/n)^2 n/2 - (r/n)r\right) \\
&= e^{-r^2/2n}.
\end{aligned}$$

C.5-7 ★ The right-hand side of inequality (C.51) can be expressed as the function $f(\alpha) = \exp(\mu e^{\alpha} - \alpha r)$, where $\alpha > 0$ and $r > \mu > 0$. We examine its derivatives:

$$\begin{aligned}
f'(\alpha) &= (\mu e^{\alpha} - r)\exp(\mu e^{\alpha} - \alpha r), \\
f''(\alpha) &= \left(\mu e^{\alpha} + (\mu e^{\alpha} - r)^2\right)\exp(\mu e^{\alpha} - \alpha r).
\end{aligned}$$

$f'(\alpha) = 0$ for $\alpha = \ln(r/\mu)$, and $f''(\alpha) > 0$ for all $\alpha > 0$. Therefore, $f(\alpha)$ is minimized at $\alpha = \ln(r/\mu)$.

Problems

C-1 *The Monty Hall problem*

a. Let A be the event that the door we chose at first is the one with the automobile, and let W be the event of winning the automobile. Then $\Pr\{A\} = 1/3$ and

$$\begin{aligned}\Pr\{W\} &= \Pr\{W \cap A\} + \Pr\left\{W \cap \overline{A}\right\} \\ &= \Pr\{A\}\Pr\{W \mid A\} + \Pr\left\{\overline{A}\right\}\Pr\left\{W \mid \overline{A}\right\} \qquad \text{(by equation (C.16))} \\ &= (1/3)\cdot\Pr\{W \mid A\} + (2/3)\cdot\Pr\left\{W \mid \overline{A}\right\}.\end{aligned}$$

Let's calculate the value of the above probability depending on the decision we made after Carol has revealed the first goat. If we decided to stick to our current choice, $\Pr\{W \mid A\} = 1$ and $\Pr\left\{W \mid \overline{A}\right\} = 0$, so we win with probability $\Pr\{W\} = 1/3$. Otherwise, we switched, and then $\Pr\{W \mid A\} = 0$ and $\Pr\left\{W \mid \overline{A}\right\} = 1$. In this case the probability of winning is $\Pr\{W\} = 2/3$. Thus, by choosing to switch, we double the chances of winning the prize.

b. In this game we make two decisions — first we choose one of the three doors, and then, after one of the doors have been opened, whether to stick or switch. There are therefore six outcomes, of which three correspond to winning the game. The results are detailed in Figure C-1.

	stick	switch
the right door	$(1 - p_{\text{right}}\,p_{\text{switch}})/3$	$p_{\text{right}}\,p_{\text{switch}}/3$
the first wrong door	$(1 - p_{\text{wrong}}\,p_{\text{switch}})/3$	$p_{\text{wrong}}\,p_{\text{switch}}/3$
the second wrong door	$(1 - p_{\text{wrong}}\,p_{\text{switch}})/3$	$p_{\text{wrong}}\,p_{\text{switch}}/3$

Figure C-1 The probability distribution in the Monty Hall's game. Rows represent our first choice and columns represent our second choice, and each cell contains the probability of the corresponding outcome. The cells shaded blue correspond to the winning outcomes.

c. By summing up the probabilities of the winning outcomes from the table in Figure C-1, we get

$$\begin{aligned}p_{\text{win}} &= (1 - p_{\text{right}}\,p_{\text{switch}})/3 + p_{\text{wrong}}\,p_{\text{switch}}/3 + p_{\text{wrong}}\,p_{\text{switch}}/3 \\ &= \frac{1}{3}(2\,p_{\text{wrong}}\,p_{\text{switch}} - p_{\text{right}}\,p_{\text{switch}} + 1).\end{aligned}$$

d. In order to minimize our chances p_{win} of winning, given $p_{\text{switch}} > 0$, Monty needs to minimize the term $2p_{\text{wrong}}p_{\text{switch}}$, as well as maximize the term $p_{\text{right}}p_{\text{switch}}$. Therefore, his best strategy is $p_{\text{right}} = 1$ and $p_{\text{wrong}} = 0$. With such values, our chances of winning are

$$\begin{aligned} p_{\text{win}} &= (1 - p_{\text{switch}})/3 \\ &< 1/3. \end{aligned}$$

e. If $p_{\text{switch}} = 0$, the winning probability reduces to $p_{\text{win}} = 1/3$, independently of any choice for p_{right} and p_{wrong}.

f. In order to maximize p_{win} for fixed p_{right} and p_{wrong}, we need to maximize the expression

$$2p_{\text{wrong}}p_{\text{switch}} - p_{\text{right}}p_{\text{switch}} = p_{\text{switch}}(2p_{\text{wrong}} - p_{\text{right}}). \tag{8}$$

If $p_{\text{right}} \le 2p_{\text{wrong}}$, we just set p_{switch} to the maximum possible value of 1. Otherwise, the value of (8) is nonpositive, so by letting $p_{\text{switch}} = 0$ we make it achieve 0.

g. As a function of p_{right} and p_{wrong}, the expression (8) is minimized, when $p_{\text{right}} = 1$ and $p_{\text{wrong}} = 0$. Then, $p_{\text{win}} = (1 - p_{\text{switch}})/3$, so our best strategy to maximize p_{win} is to choose $p_{\text{switch}} = 0$.

h. Let M be the event that Monty gives us an opportunity to switch, and let A and W be the events of the same meaning as in the solution to part (a). Then, $\Pr\{M \mid A\} = p_{\text{right}}$ and $\Pr\{M \mid \overline{A}\} = p_{\text{wrong}}$, and so

$$\begin{aligned} \Pr\{M\} &= \Pr\{A\}\Pr\{M \mid A\} + \Pr\{\overline{A}\}\Pr\{M \mid \overline{A}\} \\ &= (1/3)p_{\text{right}} + (2/3)p_{\text{wrong}}. \end{aligned}$$

Now let S be the event that we switched. Then,

$$\begin{aligned} W \cap M &= \big((A \cap \overline{S}) \cup (\overline{A} \cap S)\big) \cap M \\ &= \big((A \cap M) \cap \overline{S}\big) \cup \big((\overline{A} \cap M) \cap S\big), \end{aligned}$$

and the events $A \cap M$ and $\overline{S}$ are independent, and so are the events $\overline{A} \cap M$ and S.

Thus, by the definition (C.16) of conditional probability,

$$\begin{aligned}
\Pr\{W \mid M\} &= \frac{\Pr\{W \cap M\}}{\Pr\{M\}} \\
&= \frac{\Pr\left\{(A \cap M) \cap \overline{S}\right\} + \Pr\left\{(\overline{A} \cap M) \cap S\right\}}{\Pr\{M\}} \\
&= \frac{\Pr\{A\}\Pr\{M \mid A\}\Pr\left\{\overline{S}\right\} + \Pr\left\{\overline{A}\right\}\Pr\left\{M \mid \overline{A}\right\}\Pr\{S\}}{\Pr\{M\}} \\
&= \frac{(1/3)\,p_{\text{right}}(1 - p_{\text{switch}}) + (2/3)\,p_{\text{wrong}}\,p_{\text{switch}}}{(1/3)\,p_{\text{right}} + (2/3)\,p_{\text{wrong}}} \\
&= \frac{p_{\text{right}} - p_{\text{right}}\,p_{\text{switch}} + 2p_{\text{wrong}}\,p_{\text{switch}}}{p_{\text{right}} + 2p_{\text{wrong}}}.
\end{aligned}$$

The conditional probability $\Pr\{W \mid M\}$ is defined whenever $\Pr\{M\} \neq 0$, which holds when $p_{\text{right}} + 2p_{\text{wrong}} \neq 0$ or, equivalently, when $p_{\text{right}} \neq 0$ or $p_{\text{wrong}} \neq 0$.

i. The value of the expression for $p_{\text{switch}} = 1/2$ is

$$\begin{aligned}
\Pr\{W \mid M\} &= \frac{p_{\text{right}}/2 + p_{\text{wrong}}}{p_{\text{right}} + 2p_{\text{wrong}}} \\
&= 1/2.
\end{aligned}$$

When $p_{\text{switch}} < 1/2$, Monty can select $p_{\text{right}} = 0$ and any $p_{\text{wrong}} > 0$, so that

$$\begin{aligned}
\Pr\{W \mid M\} &= \frac{2p_{\text{wrong}}\,p_{\text{switch}}}{2p_{\text{wrong}}} \\
&= p_{\text{switch}} \\
&< 1/2.
\end{aligned}$$

Similarly, when $p_{\text{switch}} > 1/2$, Monty's best strategy is to choose any $p_{\text{right}} > 0$ and $p_{\text{wrong}} = 0$, in which case

$$\begin{aligned}
\Pr\{W \mid M\} &= \frac{p_{\text{right}} - p_{\text{right}}\,p_{\text{switch}}}{p_{\text{right}}} \\
&= 1 - p_{\text{switch}} \\
&< 1/2.
\end{aligned}$$

j. In part (i) we showed that for any value of p_{switch} other than $1/2$, Monty can always find a strategy that will reduce our chances of winning, as compared to the situation when we choose $p_{\text{switch}} = 1/2$.

In this problem we studied different strategies for Monty and for the player. The analysis of the original problem showed that switching can actually double the player's chances of winning the game, which for most people may seem counterintuitive. We have seen that for any Monty's strategy, there is an optimal strategy that the player can follow in order to maximize their chances to win, given that they know the Monty's plan. And vice versa, knowing how the player will act, Monty can always reduce the player's chances.

C-2 Balls and bins

a. We place each ball in one of the b bins. There are b ways to choose a bin for each ball. Thus, we can make n such choices in b^n ways.

b. We can view this problem as counting all possible arrangements of n distinguishable balls and $b-1$ indistinguishable sticks. Informally speaking, the sticks in such arrangements partition the balls into subsequences of balls that end up in different bins.

There are $(b+n-1)!$ of all such arrangements, but since we do not distinguish sticks, we divide this number by the number of permutations of the sticks, $(b-1)!$, to obtain the number $\frac{(b+n-1)!}{(b-1)!}$ of ways to place the balls in the bins in this variant.

c. Compared to the situation from part (b), here we are not distinguishing the balls, so every permutation of them — with the positions of the sticks unchanged — represents the same arrangement of the balls in the bins. Thus, we have $\frac{(b+n-1)!}{n!\,(b-1)!} = \binom{b+n-1}{n}$ ways to place the balls.

d. Out of b bins we choose n that will contain a ball. There are $\binom{b}{n}$ ways to do this.

e. First, we place a ball in each bin, so that none of the bins are empty. By part (c), we can place the remaining $n-b$ balls in the b bins in $\binom{b+(n-b)-1}{n-b} = \binom{n-1}{n-b} = \binom{n-1}{b-1}$ ways.

D Matrices

D.1 Matrices and matrix operations

D.1-1 By the definition of a symmetric matrix, $A = A^{\mathrm{T}}$ and $B = B^{\mathrm{T}}$ or, equivalently, for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, n$, $a_{ij} = a_{ji}$ and $b_{ij} = b_{ji}$. Let $C = A + B$ and $D = A - B$. Then for all $i = 1, 2, \ldots, n$ and all $j = 1, 2, \ldots, n$,

$$\begin{aligned} c_{ij} &= a_{ij} + b_{ij} \\ &= a_{ji} + b_{ji} \\ &= c_{ji}, \end{aligned}$$

and, similarly,

$$\begin{aligned} d_{ij} &= a_{ij} - b_{ij} \\ &= a_{ji} - b_{ji} \\ &= d_{ji}, \end{aligned}$$

so C and D are symmetric.

D.1-2 Suppose that A is a $p \times q$ matrix and B is a $q \times r$ matrix. We'll denote $A^{\mathrm{T}} = (a_{ij}^{\mathrm{T}}) = (a_{ji})$ and $B^{\mathrm{T}} = (b_{ij}^{\mathrm{T}}) = (b_{ji})$. Let $C = (AB)^{\mathrm{T}}$. Then, C is an $r \times p$ matrix, such that for $i = 1, 2, \ldots, p$ and $j = 1, 2, \ldots, r$,

$$\begin{aligned} c_{ji} &= \sum_{k=1}^{q} a_{ik} b_{kj} \\ &= \sum_{k=1}^{q} b_{jk}^{\mathrm{T}} a_{ki}^{\mathrm{T}}, \end{aligned}$$

so $C = B^{\mathrm{T}} A^{\mathrm{T}}$.

This fact implicates that

$$\begin{aligned}(A^{\mathrm{T}}A)^{\mathrm{T}} &= A^{\mathrm{T}}(A^{\mathrm{T}})^{\mathrm{T}} \\ &= A^{\mathrm{T}}A,\end{aligned}$$

and so the matrix $A^{\mathrm{T}}A$ is symmetric.

D.1-3 Let $L' = (l'_{ij})$ and $L'' = (l''_{ij})$ be $n \times n$ lower-triangular matrices. Let $L = L'L''$. For $1 \le i < j \le n$,

$$\begin{aligned}l_{ij} &= \sum_{k=1}^{n} l'_{ik} l''_{kj} \\ &= \sum_{k=1}^{j-1} l'_{ik} l''_{kj} + \sum_{k=j}^{n} l'_{ik} l''_{kj} \\ &= \sum_{k=1}^{j-1} l'_{ik} \cdot 0 + \sum_{k=j}^{n} 0 \cdot l''_{kj} \\ &= 0,\end{aligned}$$

so L is lower-triangular.

D.1-4 Let $PA = A' = (a'_{ij})$. Suppose that for $i = 1, 2, \ldots, n$, row i of the matrix P has 1 in column c_i (i.e., $p_{ic_i} = 1$). Then,

$$\begin{aligned}a'_{ij} &= \sum_{k=1}^{n} p_{ik} a_{kj} \\ &= p_{ic_i} a_{c_i j} \\ &= a_{c_i j}.\end{aligned}$$

Therefore, the element a'_{ij} in the matrix PA is an element from A from the same row but possibly from a different column. This means that PA is formed by permuting rows of A.

Similarly, let $AP = A'' = (a''_{ij})$ and for $j = 1, 2, \ldots, n$ let r_j be the row number, such that $p_{r_j j} = 1$. Then,

$$\begin{aligned}a''_{ij} &= \sum_{k=1}^{n} a_{ik} p_{kj} \\ &= a_{ir_j} p_{r_j j} \\ &= a_{ir_j},\end{aligned}$$

which implies that AP is A with its columns permuted.

Now suppose that A is a permutation matrix. Permuting rows of A preserves the property of permutation matrices that each row and each column has exactly one 1, and 0s elsewhere. Therefore, PA is also a permutation matrix.

D.2 Basic matrix properties

D.2-1

$$\begin{aligned} B &= BI \\ &= B(AC) \\ &= (BA)C \\ &= IC \\ &= C \end{aligned}$$

D.2-2

Let L be an $n \times n$ lower-triangular matrix. We'll prove by induction on n that the determinant of L is the product of its diagonal elements. The base case is when $n = 1$, and from the definition of determinant we have $\det(L) = l_{11}$. Now suppose that $n \geq 2$. The minor $L_{[11]}$ is a lower-triangular matrix, so let's assume as an inductive hypothesis, that $\det(L_{[11]}) = \prod_{i=2}^{n} l_{ii}$. Then we have

$$\begin{aligned} \det(L) &= \sum_{j=1}^{n} (-1)^{1+j} l_{1j} \det(L_{[1j]}) \\ &= (-1)^{1+1} l_{11} \det(L_{[11]}) \\ &= l_{11} \prod_{i=2}^{n} l_{ii} \\ &= \prod_{i=1}^{n} l_{ii}, \end{aligned} \tag{9}$$

which concludes the inductive proof.

Now let U be an $n \times n$ upper-triangular matrix. There exists a lower-triangular matrix

L, such that $U = L^{\mathrm{T}}$. Then U and L have the same diagonal elements, and so

$$\begin{aligned}\det(U) &= \det(L^{\mathrm{T}}) \\ &= \det(L) && \text{(by Theorem D.4)} \\ &= \prod_{i=1}^{n} l_{ii} && \text{(by equation (9))} \\ &= \prod_{i=1}^{n} u_{ii}\,.\end{aligned}$$

Now let L be a nonsingular $n \times n$ lower-triangular matrix and let $L' = (l'_{ij})$ be the inverse of L. We use induction to show that L' is a lower-triangular matrix. When $n = 1$, L' is a 1×1 matrix, which is trivially lower-triangular. For the inductive step, let $n \geq 2$ and assume that the minor $L'_{[11]}$ itself is lower-triangular. For $j = 1, 2, \ldots, n$, the element in the first row and the jth column of I_n is equal to $\sum_{k=1}^{n} l_{1k} l'_{kj} = l_{11} l'_{1j}$. If $j = 1$, then $l_{11} l'_{11} = 1$, so $l'_{11} = 1/l_{11} \neq 0$. Note that the existence of the inverse of L implies $l_{11} \neq 0$. If $j > 1$, $l_{11} l'_{1j} = 0$, so l'_{1j} has to be 0. Combining these results with the inductive hypothesis, we have that L' is lower-triangular.

D.2-3 Suppose that P is an $n \times n$ matrix. We'll prove that P is invertible by showing that $P^{\mathrm{T}} = (p^{\mathrm{T}}_{ij})$ is its inverse. Suppose that for $i = 1, 2, \ldots, n$, c_i is the number of column, such that $p_{ic_i} = 1$. Then $p^{\mathrm{T}}_{c_i i} = 1$ is the only nonzero element in column i of P^{T}.

If we let $A = PP^{\mathrm{T}}$, then for every $1 \leq i \leq n$ and $1 \leq j \leq n$, we have that

$$\begin{aligned}a_{ij} &= \sum_{k=1}^{n} p_{ik} p^{\mathrm{T}}_{kj} \\ &= \sum_{k=1}^{n} p_{ik} p_{jk} \\ &= p_{ic_i} p_{jc_i} \\ &= p_{jc_i}\,.\end{aligned}$$

By the fact that P is a permutation matrix, we have that if $i = j$, then $a_{ij} = 1$, otherwise $a_{ij} = 0$. That is, $A = I_n$, and so $P^{\mathrm{T}} = P^{-1}$.

In a permutation matrix P each row and each column has exactly one 1, and 0s elsewhere. The rows of P are the columns of P^{T}, and the columns of P are the rows of P^{T}, so the permutation matrix property is preserved in P^{T}.

D.2-4 Let C be the $n \times n$ matrix with $c_{ij} = 1$, and 0s elsewhere. Observe that CA is the matrix, in which the ith row is the jth row of A, while other rows consist of 0s. Similarly, BC

is the matrix, in which the jth column is the ith column of B, while other columns consist of 0s. Let $D = I + C$ and $D' = I - C$. Then, $A' = DA$ and $B' = BD'$.

Observe, that since $i \neq j$, for any $1 \leq p, q \leq n$ the terms c_{pk} and c_{kq} can't be both nonzero. Thus, $\sum_{k=1}^{n} c_{pk} c_{kq} = 0$, and so CC is a zero matrix. Then,

$$\begin{aligned} DD' &= (I + C)(I - C) \\ &= II + CI - IC - CC \\ &= I + C - C - CC \\ &= I, \end{aligned}$$

which means that $D' = D^{-1}$.

Returning to the matrices A' and B', we get

$$\begin{aligned} A'B' &= DABD' \\ &= D(AB)D^{-1} \\ &= (DI)D^{-1} \\ &= DD^{-1} \\ &= I, \end{aligned}$$

hence $B' = (A')^{-1}$.

D.2-5 We need only prove the forward direction, since the inverse of A^{-1} is $(A^{-1})^{-1} = A$ and the backward direction is symmetric. Suppose that every entry of $A^{-1} = (a_{ij}^{-1})$ is real. We'll use induction to show that every entry of A is real.

If $n = 1$, A^{-1} has a single real entry a_{11}^{-1}, and the only entry of A is $a_{11} = 1/a_{11}^{-1}$, so it is real. Now let $n \geq 2$ and assume that every entry of $A_{[11]}$ is real. Suppose that there exists $2 \leq i \leq n$, such that a_{i1} is complex but not real. Since $AA^{-1} = I_n$, we have

$$\begin{aligned} 1 &= \sum_{k=1}^{n} a_{ik} a_{ki}^{-1} \\ &= a_{i1} a_{1i}^{-1} + \sum_{k=2}^{n} a_{ik} a_{ki}^{-1}. \end{aligned}$$

The term $a_{i1} a_{1i}^{-1}$ is not real, and so can't be the last summation, in order to cancel out the imaginary components on the right-hand side and sum up to 1. This means that some other element from the ith row of A can't be real, which contradicts our assumption. Using the identity $A^{-1}A = I_n$ and a similar reasoning we can show that all elements a_{1j}, where $2 \leq j \leq n$, are real. This fact implies, that in the following

equation

$$1 = a_{11}a_{11}^{-1} + \sum_{k=2}^{n} a_{1k}a_{k1}^{-1}$$

the last summation is a real number, so a_{11} has to be real as well.

D.2-6 *In the second part of the exercise we also need to assume that A is a symmetric $n \times n$ matrix. Corrected in fourth printing.*

We have

$$\begin{aligned}(A^{-1})^{\mathrm{T}} &= (A^{\mathrm{T}})^{-1} \\ &= A^{-1},\end{aligned}$$

so A^{-1} is symmetric.

In the second part, since A is symmetric,

$$\begin{aligned}(BAB^{\mathrm{T}})^{\mathrm{T}} &= (B^{\mathrm{T}})^{\mathrm{T}}A^{\mathrm{T}}B^{\mathrm{T}} \\ &= BAB^{\mathrm{T}},\end{aligned}$$

so the product BAB^{T} gives a symmetric matrix.

D.2-7 Let A be an $m \times n$ matrix that has full column rank and let x be an n-vector such that $Ax = 0$. If we denote the columns of A by $a_1, a_2, \ldots, a_n$, then the product Ax can be expressed as the linear combination $\sum_{j=1}^{n} a_j x_j$. Because the column rank of the matrix A is n, all the column vectors are linearly independent, so x_j — viewed as coefficients from the definition of linear dependency — are all 0. Therefore, $x = 0$.

For the other direction, assume that for an $m \times n$ matrix A, the condition $Ax = 0$ implies $x = 0$, and that the matrix A has the rank less than n. This means that there is some set of column vectors of A which are linearly dependent. Note that we can extend this set to include all column vectors of A, $a_1, a_2, \ldots, a_n$, and they all will remain linearly dependent. Let $c_1, c_2, \ldots, c_n$ be coefficients, not all of which are 0, such that $\sum_{j=1}^{n} a_j c_j = 0$. If we consider the n-vector $c = (c_j)$, then the equation can be viewed as $Ac = 0$. But $c \neq 0$, so we have a contradiction.

D.2-8 Let A be an $m \times p$ matrix and let B be $p \times n$ matrix. From the alternate definition of the rank, $r = \mathrm{rank}(AB)$ is the smallest number such that $AB = CD$ for some matrices C and D of respective sizes $m \times r$ and $r \times n$. Let $r' = \mathrm{rank}(A)$ and $r'' = \mathrm{rank}(B)$ and let C', D', C'', D'' be matrices of respective sizes $m \times r'$, $r' \times p$, $p \times r''$, $r'' \times n$ such that $A = C'D'$ and $B = C''D''$. Since $AB = (C'D'C'')D''$ and $C'D'C''$

is an $m \times p$ matrix, we have that $r \leq r'$. Similarly, if we represent the product as $AB = C'(D'C''D'')$, we conclude that $r \leq r''$. Therefore, $r \leq \min\{r', r''\}$.

Now consider a special case, where $p = m$ and A is nonsingular. Let r, r', r'', C and D have the same meaning as before. Of course, $r'' \leq \min\{m, n\} \leq m$ and, by Theorem D.1, $r' = m$. Also,

$$\begin{aligned} B &= A^{-1}AB \\ &= (A^{-1}C)D, \end{aligned}$$

and since $A^{-1}C$ is an $m \times r$ matrix, we have that $r'' \leq r$. Combining this with the inequality $r \leq r''$ shown in the proof for a general case, we obtain $r = r'' = \min\{r', r''\}$. The proof for the case in which $p = n$ and B is nonsingular, is symmetric.

Problems

D-1 *Vandermonde matrix*

The proof is by induction on n and is based on the fact that if we add to a column of a matrix the product by a scalar of another column, then the determinant remains unchanged. This fact can be devised from Theorem D.4 — first multiply a column by a scalar, then add this column to another column, then divide the original column by the scalar.

For the base case of the induction, let $n = 1$. Then the single entry of $V(x_0)$ is 1. Clearly, $\det(V(x_0)) = 1$, that matches the empty product $\prod_{0 \leq j < k \leq 0}(x_k - x_j)$, by convention equal to 1.

For the inductive step, let $n \geq 2$. As the hint suggests, for $i = n-1, n-2, \ldots, 1$, we will multiply column i by $-x_0$ and add it to column $i+1$, to obtain the matrix W shown below:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & x_1 - x_0 & x_1(x_1 - x_0) & x_1^2(x_1 - x_0) & \cdots & x_1^{n-2}(x_1 - x_0) \\ 1 & x_2 - x_0 & x_2(x_2 - x_0) & x_2^2(x_2 - x_0) & \cdots & x_2^{n-2}(x_2 - x_0) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} - x_0 & x_{n-1}(x_{n-1} - x_0) & x_{n-1}^2(x_{n-1} - x_0) & \cdots & x_{n-1}^{n-2}(x_{n-1} - x_0) \end{pmatrix}.$$

By the definition of determinant, $\det(W) = \det(W_{[11]})$. As all the entries in the ith row of $W_{[11]}$ have a factor of $x_i - x_0$, using Theorem D.4 we can take these factors out

and obtain

$$\begin{aligned}\det(V(x_1, x_2, \ldots, x_{n-1})) &= \det(W) \\ &= \det(W_{[11]}) \\ &= \prod_{i=1}^{n-1}(x_i - x_0) \cdot \det(V(x_1, x_2, \ldots, x_{n-1})) \\ &= \prod_{i=1}^{n-1}(x_i - x_0) \prod_{1 \le j < k \le n-1}(x_k - x_j) \\ &= \prod_{0 \le j < k \le n-1}(x_k - x_j).\end{aligned}$$

D-2 ***Permutations defined by matrix-vector multiplication over GF(2)***

a. Let $a_0, a_1, \ldots, a_{n-1}$ be the columns of A and let $J \subseteq \{0, 1, \ldots, n-1\}$ be the set of indices of all linearly independent columns of A. Consider vectors $x, x' \in S_n$, such that there exists $j \in J$ for which $x_j \neq x'_j$, and that for all $j \in \{0, 1, \ldots, n-1\} - J$, $x_j = x'_j = 0$. Then,

$$\begin{aligned}Ax - Ax' &= A(x - x') \\ &= \sum_{j=0}^{n-1} a_j (x_j - x'_j) \\ &= \sum_{j \in J} a_i (x_j - x'_j) \\ &\neq 0,\end{aligned}$$

so $Ax \neq Ax'$. Because $|J| = r$, there are 2^r ways of picking the vector x, and each such x is mapped to a distinct value Ax. Therefore, $|R(A)| \geq 2^r$.

To get the upper bound, observe that any column a_j, where $j \notin J$, is in fact a linear combination of the r linearly independent columns of A. Thus, for any vector x,

$$\begin{aligned}Ax &= \sum_{j=0}^{n-1} a_j x_j \\ &= \sum_{j \in J} a_j x'_j\end{aligned}$$

for some coefficients x'_j. In other words, for any x there exists a vector x', with zeros on positions $j \notin J$, mapped to the same value as x. There are 2^r ways of choosing

such x', so $|R(A)| \leq 2^r$.

If $\text{rank}(A) < n$, then $|R(A)| < 2^n = |S_n|$, so A can't define a permutation on S_n.

b. We showed in part (a), that the value produced by multiplying a vector x by A does not depend on the $n-r$ entries of x occupying the positions corresponding to the linear dependent columns of A. Therefore, for any $y \in R(A)$, $|P(A, y)| \geq 2^{n-r}$.

On the other hand, there are 2^r elements in $R(A)$, each with the preimage consisting of at least 2^{n-r} elements. This means, there are at least $2^r \cdot 2^{n-r} = 2^n$ elements in all preimages in total. Since $|S_n| = 2^n$, each preimage must have exactly 2^{n-r} elements.

c. First observe that $B(S', m)$ consists of size-2^m blocks which contain a value produced by any $x \in S$. Since the first m rows of A only affect the first m entries of Ax, they can affect the value of Ax by at most $2^m - 1$. The block where Ax ends up is therefore determined by the last $n - m$ rows of A. What is more, the numbers in block i differ from the numbers in block 0 by $i2^m$, therefore the produced values are also shifted by a constant offset. Thus, without loss of generality, we may assume that S is block 0, so that only the first m columns of A are relevant during multiplication.

By similar reasoning as in part (a), only the linearly independent rows of the lower left $(n-m)\times m$ submatrix of A can determine the block where Ax will end up, since the entries of Ax on the positions that correspond to the other rows are linear combinations of the other entries of Ax. Thus, Ax spans 2^r blocks, so $|B(S', m)| = 2^r$.

Similarly, we could identify the r linearly independent columns of the submatrix that decide on the block for Ax. A given block is uniquely chosen by a combination of the r entries of x that are multiplied by the elements of these columns, while the remaining $m - r$ entries (besides zeros at positions $m, m + 1, \ldots, n - 1$) can be arbitrary. Thus, each block must be hit by the same number of times, equal to 2^{m-r}.

d. The number of linear permutations is bounded above by the number of pairs (A, c), where A is an $n \times n$ 0-1 matrix and c is an n-bit vector. There are $2^{n^2} \cdot 2^n = 2^{n(n+1)}$ of such pairs. On the other hand, there are $(2^n)!$ permutations of S_n. For $n \geq 3$, $(2^n)^{n+1} \leq (2^n)!$.

e. Observe that $A \cdot 0 + c = c$, and $Ax = \sum_{j=0}^{n-1} a_j x_j$, where $a_0, a_1, \ldots, a_{n-1}$ are the columns of A, so if we let $x = 2^i$, then $Ax = a_i$. Therefore, the linear permutation induced by the pair $(A, 0)$ maps $x = 0$ to 0, and maps $x = 2^i$ to the number whose binary representation is the ith column of A.

Consider a permutation $\pi_{A,c}$ of S_2, such that $\pi_{A,c}(x) = x$, for $x \in \{0, 1, 2\}$. Based on the above observation, this is enough to determine both the matrix A and the vector c: $A = I$, $c = 0$. These in turn determine the value $\pi_{A,c}(3) = 3$, which makes $\pi_{A,c}$

the identity permutation. Therefore, any permutation of S_2 that maps each $x \in \{0, 1, 2\}$ to x and 3 to anything else than 3, is impossible to achieve by any linear permutation.

Bibliography

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 4th edition, 2022.