Exercise Solutions to CLRS' Introduction to Algorithms – 4th Edition

Alan Sorani

January 2, 2024

Contents

Preface

These exercise solutions are being written during my reading of CLRS' Introduction to Algorithms [Cor+22] as means of assuring my understanding of the discussed material. They should also act as a more thorough reference to the subject once they are complete, as they contain solutions to exercises, and possibly explanations to things I found lacking in the original text.

1 The Role of Algorithms in Computing

1.1 Algorithms

Exercise 1.1-1. Real-world example of a problem that requires sorting are sorting films by their rating or finding the closest restaurant to a given location. The latter requires finding the shortest distance between two points.

Exercise 1.1-2. We also need to consider the size of the data. Especially today, when machine-learning algorithms use very large datasets, storing the amount of data can become an important consideration.

Exercise 1.1-3. Arrays are a very simple data structures that is widely used. The data is stored as a block in the memory, which is fast to read together, and accessing elements can be done in constant time which is e.g. not the case in lists or trees.

Arrays have the disadvantage of not being easily scalable. If we want to make an array bigger, we generally have to copy all the current elements into a larger array.

Exercise 1.1-4. The shortest paths and travelling salesperson problems both try to minimize the distance traveled under certain restrictions : for the first this is just the distance between two nodes while for the latter this is traveling through multiple points. The shortest path is thus a specific travelling-salesperson problem with just two nodes.

The big difference is that while the shortest path problem can be solved in polynomial time, we don't know of any polynomial-time algorithm for the travelling-salesperson problem.

Exercise 1.1-5. A real-life problem which requires only the best solution is determining the amount of rocket fuel needed to be used. If the wrong amounts are used, the rocket will hit the wrong target and might hurt civilians.

A problem which can have an approximate solution is restaurant recommendation. If we get a good restaurant but not the best one, nobody will be hurt and we're still likely to be happy.

Exercise 1.1-6. A problem in which we have the whole input at the start would be translation of text.

A problem in which we get the data over time is live translation of news on television.

1.2 Algorithms as a technology

Exercise 1.2-1. An artificial intelligence that navigates automatic cars would need significant algorithmic content at the application level, from processing footage from the cameras to deciding on the best route given that information.

Exercise 1.2-2. We have to find the values of n for which $8n^2 < 64n\log(n)$. Dividing by $8n$ these are the values for which $n < 8\log(n)$. Taking 2 to the power of these terms, these are the values of n for which

$$2^n < 2^{8\log(n)} = \left(2^{\log(n)}\right)^8 = n^8,$$

which are the values for which

$$f(n) := \frac{2^n}{n^8} < 1.$$

We have $f(1) = 2$ and see by direct computation that $f(n) < 1$ for all integers $n \in [2, 20]$. We have

$$\begin{aligned} f(n) - f(n-1) &= \frac{2^n}{n^8} - \frac{2^{n-1}}{(n-1)^8} \\ &= \frac{2^n - 2^{n-1} \cdot \frac{n^8}{(n-1)^8}}{n^8} \\ &= \frac{2^n - 2^{n-1} \cdot \left(\frac{n}{n-1}\right)^8}{n^8}. \end{aligned}$$

This is positive whenever $\frac{n}{n-1} < \sqrt[8]{2}$. Solving an inequality we see that this is the case for $n > \frac{\sqrt[8]{2}}{\sqrt[8]{2}-1} \approx 12.04878$. In particular, f is monotonically increasing on $[13, \infty)$. By listing values we see $f(43) < 1$ and $f(44) > 1$, so $f(n) < 1$ exactly for $n \in [2, 43]$.

Exercise 1.2-3. We have to solve $100n^2 < 2^n$, i.e. $f(n) \coloneqq \frac{100n^2}{2^n} < 1$. We see that f is monotonically-decreasing on $[4, \infty)$ since

$$\begin{aligned} f(n) - f(n-1) &= \frac{100n^2}{2^n} - \frac{100\left(n^2 - 2n + 1\right)}{2^{n-1}} \\ &= \frac{100\left(n^2 - 2n^2 + 4n - 2\right)}{2^n} \\ &= \frac{100\left(-n^2 + 4n - 2\right)}{2^n} \end{aligned}$$

and the enumerator is negative for $n > 2 + \sqrt{2}$. Listing values of f we see that $f(15)$ is the first integer value below 1, so $f(n) < 1$ exactly for $n \in [15, \infty)$.

Problems

Problem 1-1. We have to find the maximal n for which $f(n) \leq t$ where t is the specified time in miliseconds. Since all functions f are monotonically increasing on $[1, \infty)$, this is the integer n such that $f(n) \leq t$ but $f(n+1) > t$.

- For $f(n) = \log(n)$, we have $f(n) \leq t$ if and only if $2^{f(n)} < 2^t$, if and only if $n \leq 2^t$.
- For $f(n) = \sqrt{n}$, we have $f(n) \leq t$ if and only if $n \leq t^2$.
- For $f(n) = n$, the calculation is trivial.
- For $f(n) = n \log n$, we want to solve $f(n) = t$ which is equivalence to $n = \frac{t}{\log(n)}$. Taking $g(n) = \frac{t}{\log(n)}$ we want to solve $n = g(n)$, i.e. search for a fixed point of g. We see that for all positive integers m, n such that $t \geq \log(m)\log(n)$:

  $$\begin{aligned} |g(m) - g(n)| &= t\left|\frac{1}{\log(m)} - \frac{1}{\log(n)}\right| \\ &= t\left|\frac{\log(n) - \log(m)}{\log(m)\log(n)}\right| \\ &\leq |\log(n) - \log(m)| \leq |n - m|. \end{aligned}$$

  By the Banach fixed point theorem, there is a unique fixed point of g given as the limit $\lim_{n\to\infty} g^{(n)}(x_0)$ for any x_0 such that $t \geq \log(x)^2$. Taking $x_0 = t$ we manually consider a list of iterations which leads us to a guess. We then check that $f(\lfloor x_0 \rfloor) \leq t$ and $f(\lfloor x_0 \rfloor + 1) > t$.
- For $f(n) = n^2$, we have $f(n) \leq t$ if and only if $n \leq \sqrt{t}$.
- For $f(n) = n^3$, we have $f(n) \leq t$ if and only if $n \leq \sqrt[3]{t}$.

- For $f(n) = 2^n$, we have $f(n) \leq t$ if and only if $\log f(n) \leq \log t$ if and only if $n < \log t$.
- For $f(n) = n!$, we manually compute the values of $f(n)$ for small values of n to get the results.

We get the following values.

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
log n	2^{10^3}	$2^{6 \cdot 10^4}$	$2^{3.6 \cdot 10^6}$	$2^{8.64 \cdot 10^7}$	$2^{2.592 \cdot 10^9}$	$2^{3.154 \cdot 10^{10}}$	$2^{3.154 \cdot 10^{12}}$
$\sqrt{n}$	10^6	$3.6 \cdot 10^9$	$1.296 \cdot 10^{13}$	$7.46496 \cdot 10^{15}$	$6.718464 \cdot 10^{18}$	$9.94772 \cdot 10^{20}$	$9.94772 \cdot 10^{24}$
n	10^3	$6 \cdot 10^4$	$3.6 \cdot 10^6$	$8.64 \cdot 10^7$	$2.592 \cdot 10^9$	$3.154 \cdot 10^{10}$	$3.154 \cdot 10^{12}$
$n \log n$	140	$4.895 \cdot 10^3$	$2.041 \cdot 10^5$	$3.943 \cdot 10^6$	$9.766 \cdot 10^7$	$1.052 \cdot 10^9$	$8.680 \cdot 10^{10}$
n^2	31	244	$1.897 \cdot 10^3$	$9.295 \cdot 10^3$	$5.091 \cdot 10^4$	$1.776 \cdot 10^5$	$1.776 \cdot 10^6$
n^3	10	39	153	442	$1.374 \cdot 10^3$	$3.16 \cdot 10^3$	$1.467 \cdot 10^4$
2^n	9	15	21	26	31	34	41
$n!$	6	8	9	11	12	13	15

2 Getting Started

2.1 Insertion Sort

Exercise 2.1-1.

Exercise 2.1-2. We state the following loop invariant.

> Before the i^{th} step, the `sum` variable contains the sum of $A[1]$ through $A[i-1]$.

We show that this invariant holds through initialization, maintenance and termination.

Initialization: The `sum` variable contains the number 0 which is the sum of the first zero elements.

Maintenance: If the sum before the i^{th} step is $\sum_{j=1}^{i-1} A[j]$, the sum after the step is $\left(\sum_{j=1}^{j-1} A[j]\right) + A[i] = \sum_{j=1}^{i} A[j]$, since at the i^{th} step we add $A[i]$.

Temination: The loop stops when $i = n+1$, in which case the loop invariant gives us that `sum` contains the sum $\sum_{i=1}^{n} A[i]$, which is the sum of all elements of A.

Exercise 2.1-3. Instead of going in the `while` loop over values which are larger than `key`, we go over smaller ones, so that it goes to the left of those smaller than it and we get an array that is sorted from largest to smallest.

```
1  def reverse_insertion_sort(arr):
2      for j in range(1,len(arr)):
3          key = arr[j]
4          # Insert arr[j] into the sorted sequence arr[0,...,j-1]
5          i = j-1
6          while(i >= 0 and arr[i] < key):
7              arr[i+1] = arr[i]
8              i = i-1
9          arr[i+1] = key
10     return arr
```

Exercise 2.1-4. We go over all the values of A and compare them to x.

```
1      def search(A,x):
2          result = NIL
3          for j in range(len(A)):
4              if(A[j] == x):
5                  result = j
6                  return result
7          return result
```

We state the following loop invariant.

> Before the start of the j^{th} step, the value of result is NIL if none of the first $j-1$ elements equal to x, and is otherwise equal to the first such index.

We show this loop invariant holds.

Initialization: Before the first step, the value of `result` is NIL as set in line 2 of the code.

Maintenance: If the loop invariant holds before the j^{th} step and $A[j] = x$, we set and return result $= j$, so that before the $j+1^{\text{st}}$ step we'd have `result` equal to the first matching index. If instead $A[j] \neq x$, none of the first j keys equal to x, in which case result = NIL, matching the loop invariant.

Termination: The algorithm terminates in one of two ways. If a key which equals x was found at index i, we set result $= i$ and return result, and if the algorithm went over all of the array without finding a key which equals x.

In the first case, the value of result is the first index i for which $A[i] = x$, and in the second case termination is before the $n+1$ step which means that the first n elements of A, being all of them don't contain x.

Exercise 2.1-5. The problem is the following.

Input: Two binary number representations in arrays A, B.

Output: Their sum representation C as an array.

This could be solved by the following procedure.

```
1   def sum_binary_arrays(A,B):
2       carry = 0
3       i = 1
4       result = []
5       while(i <= n+1):
6           result = result + [(arr1[i]+arr2[i]+carry)%2]
7           carry = (A[i]+B[i]+carry)//2
8           i += 1
9       if(carry):
10          result = result + [1]
11      return result
```

2.2 Analyzing algorithms

Exercise 2.2-1.

Exercise 2.2-2. Set $f(n) \coloneqq \frac{n^3}{1000} + 100n^2 - 100n + 3$. Then

$$\frac{f(n)}{n^3} = \frac{1}{1000} + \frac{100}{n} - \frac{100}{n^2} + \frac{3}{n^3}$$

which approaches the constant $\frac{1}{100}$ as $n \to \infty$. Hence $f(n) = \Theta\left(n^3\right)$.

Exercise 2.2-3. We define selection sort.

```
1   def selection_sort(A,x):
2       for i in range(1, n):
3           min_index = i
4           for j in range(i, n+1):
5               if A[j] < A[min_index]:
6                   min_index = j
7           A[i] , A[min_index] = A[min_index], A[i]
8       return A
```

This has the following loop invariant.

> At the start of the i^{th} iteration, the subarray $A[1 : i-1]$ is sorted and contains the $i-1$ smallest elements of A.

The algorithm needs to run only for the $n-1$ first smallest elements, because having $A[1 : n-1]$ sorted and containing the smallest $n-1$ elements means that $A[n]$ is at least as large as the largest of these.

The worst-case running time for the algorithm is

$$nc_2 + (n-1)c_3 + \sum_{i=1}^{n-1}(n+1-i)c_4 + \sum_{i=1}^{n-1}(n-i)c_5 + \sum_{i=1}^{n-1}(n-i)c_6 + (n-1)c_7 + c_8.$$

$$\begin{aligned}\sum_{i=1}^{n-1}(n-i) &= (n-1)\,n - \sum_{i=1}^{n-1} i \\ &= (n-1)\,n - n\cdot\frac{(n-1)+1}{2} \\ &= (n-1)\,n - \frac{n^2}{2} \\ &= n^2 - n - \frac{n^2}{2} \\ &= \frac{n^2}{2} - n \\ &= \Theta\left(n^2\right).\end{aligned}$$

All the other terms are linear in n, so the worst running time is $\Theta\left(n^2\right)$. The best running time would be the same minus the term $\sum_{i=1}^{n-1}(n-i)\,c_6$, which is also $\Theta\left(n^2\right)$.

Exercise 2.2-4. 1. The probability that the element x is $A[i]$ is $\frac{1}{n}$. To find out how many elements need to be checked on average, we calculate the expectancy

$$\mathbb{E}\left(\text{number of elements checked}\right) = \sum_{i=1}^{n} i\mathrm{P}\left(i \text{ elements checked}\right) = \sum_{i=1}^{n}\frac{i}{n} = \frac{1}{n}\cdot n\cdot\frac{1+n}{2} = \frac{1+n}{2}.$$

2. The worst case is n elements checked, which is the case where x is the last element in A.

3. Let c_i be the computation time of line i in the code. Lines 3 and 4 read i times where i is the index for which $A[i] = x$, lines $2, 5$ read once and lines $6, 7$ read at most once. We get that the average case is $\frac{1+n}{2}(c_3 + c_4) + c_2 + c_5 + c_6$, which is $\Theta(n)$ and that the worst case is $n2(c_3 + c_4) + c_2 + c_5 + c_6$, which is also $\Theta(n)$.

Exercise 2.2-5. We can check if the array is sorted before performing a sorted algorithm. This check would have computation time $\Theta(n)$, so the best-case computation time would be $\Theta(n)$.

2.3 Designing algorithm

Exercise 2.3-1. At initialization, $p \neq r$ since $1 \leq n$. If at the i^{th} step, $p = r$, we stop. Otherwise, $p < r$ so that at the $(i+1)^{\text{st}}$ step we have $q = p \leq \left\lfloor\frac{p+r}{2}\right\rfloor < r$, so that $p \leq q$ and $q + 1 \leq r$, which are the values passed to the next iteration.

Exercise 2.3-2. We state the following loop invariant.

> At the start of the (i, j) step, the array $A[1 : i + j - 1]$ contains the $i + j - 1$ smallest elements from L, R and either the elements $L[i], R[j]$ are i^{th} and j^{th} smallest elements of L, R respectively, or either $i > n_L$ or $j > n_R$.

Given the loop invariant, we see that at termination the array A contains all the elements of L, R in a sorted order, outside of the tail of one of these arrays. The following `while` loops add these elements to A. Since at termination, A contains the smallest elements in a sorted order, this results in a sorted array.

Exercise 2.3-3. Base: For $n = 2$, we have $n\log n = 2\cdot\log 2 = 2$.

Step: Assume that $T(m) = m \log m$ for all $m < n$. In particular, $T\left(\frac{n}{2}\right) = \frac{n}{2} \log\left(\frac{n}{2}\right)$, so

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + n \\ &= 2 \cdot \frac{n}{2} \log\left(\frac{n}{2}\right) + n \\ &= n \log\left(\frac{n}{2}\right) + n \\ &= \left(\log \frac{n}{2} + 1\right) n \\ &= \left(\log \frac{n}{2} + \log(2)\right) n \\ &= \log\left(\frac{n}{2} \cdot 2\right) n \\ &= n \log(n). \end{aligned}$$

Exercise 2.3-4. We write a recursive version of insertion sort.

```
1     def insertion_sort(A):
2         if(len(A) = 1):
3             return A
4         A[1 : len(A) - 1] = insertion_sort(A[1:len(A)-1])
5         i = n - 1
6         insertion = A[n]
7         while(i >= 1 and A[i] > insertion):
8             A[i+1] = A[i]
9             i = i-1
10        A[i+1] = insertion
11        return A
```

Let $T(n)$ be the worst-case running time. Then in computing $T(n)$, we call insertion sort for an array of size $n-1$, which runs in $T(n-1)$ worst-case, and we have a `while`-loop that runs at worst-case $\Theta(n)$-time. Hence a recursion formula would be

$$T(n) = T(n-1) + n.$$

Since $T(1) = \Theta(1)$, we write $T(1) = 1$. Then if $T(m) = \sum_{i=1}^{m} m$ for all $m < n$, then

$$T(n) = T(n-1) + n = \left(\sum_{i=1}^{n-1} i\right) + n = \sum_{i=1}^{n} i = \frac{(n+1)\,n}{2} = \Theta\left(n^2\right).$$

Exercise 2.3-5. We write a recursive algorithm for binary search.

```
1   def binary_search(arr, target):
2       if(len(arr) = 1):
3           return 0 if arr == [target] else -1
4       mid = int(len(arr)/2)
5       if(arr[mid] > target):
6           return binary_search(arr[:mid], target)
7       if(arr[mid] == target):
8           return mid
9       right_result = binary_search(arr[mid:], target)
10      if(right_result == -1):
11          return -1
12      return right_result + mid
```

Since each time we go over an array of length at most half the previous one, there are at most $\log(n)$ recursive calls. The worst-case running time would be when the key is found after $\log n$ calls (which is the case if it's first or last), and since each call has time complexity $\Theta(1)$ this would have time complexity $\Theta(\log n)$.

Exercise 2.3-6. No. The algorithm would need to make fewer comparisons for finding the place for insertion, but it would still need to shift $\Theta(n)$ elements to the right.

Exercise 2.3-7. Use merge-sort to sort the numbers in an array, which works in $\Theta(n \log n)$-time. Then, for each key a use binary search to search for $x - a$. There are n elements and binary search is $\Theta(\log n)$, so this is $\mathrm{O}(\log n)$. We get that the algorithm works in $\Theta(n \log n) + \mathrm{O}(n \log n) = \Theta(n \log n)$.

Problem 2-1 (Insertion sort on small arrays in merge sort)**.** a. Insertion sort sorts a list of length k in $\Theta\left(k^2\right)$ worst-case time. Since there are n/k lists, running insertion sort on each would be

$$n/k \cdot \Theta\left(k^2\right) = \Theta\left(\frac{nk^2}{k}\right) = \Theta\left(nk\right)$$

worst-case time.

b. If $n/k = 2^m$ for some integer m, merge the sublists in pairs and repeat the process for the resulting sublists. Merging two sublists of length k' is $\Theta\left(k'\right)$, and on the first step there are $\frac{n}{2k}$ such pairs. So the first step would have $\Theta\left(\frac{n}{2k} \cdot k\right) = \Theta\left(n\right)$ worst-case time. The next step would merge half as many pairs which are twice as long, which takes the same time, so all steps take the same time. If s is the number of steps, we get $k \cdot 2^s = n$ so $s = \log\left(n/k\right)$. Hence there are $\log\left(n/k\right)$ steps each with $\Theta\left(n\right)$ worst-case time, so merging everything is $\Theta\left(n \log\left(n/k\right)\right)$ worst-case time.

If n/k isn't a power of 2, we do the same, only that some of the lists will be shorted after not having lists to merge with in the previous step. This takes less than double the time if we lengthened the list to be a power of 2, as there are as many steps and each is less than twice as long. Hence worst-case time complexity is $\Theta\left(2n \log\left(n/k\right)\right) = \Theta\left(n \log\left(n/k\right)\right)$.

c. Split the list into sublists of length k, sort each of them in $\Theta\left(nk\right)$ worst-case time using insertion-sort and merge the sublists in $\Theta\left(n \log\left(n/k\right)\right)$ worst-case time. Since the complexities are independent, we get a worst-case time of $\Theta\left(nk + n \log\left(n/k\right)\right)$.

If $k = \Theta\left(\log n\right)$, we get the same running time as merge sort in terms of Θ-notation, and for $k = \omega\left(\log n\right)$ we clearly get worse running time.

d. Asymptotically, we'd want k to be sublogarithmic in n such that $k + \log\left(n/k\right)$ is minimal logarithmically. If we know approximate values of n, we could pick k to be an integer smaller than the expected value of $\log n$ which aims to do that.

Problem 2-2 (Correctness of bubblesort)**.** a. We have to show prove the existence of a loop invariant that would mean that when the algorithm terminates the array is sorted.

b.

Proposition 2.3.1. *At the start of each iteration of the* `for` *loop of lines 3-5, the smallest element of* `A[i:]` *is at most at the* j^{th} *index, and the array* `A[:i] = A[1,...,i-1]` *stays as it was at the start if the loop.*

Proof. **Initialization:** Since j starts as the last index, the index of the smallest key is at most j.

Maintenance: Assume that before the iteration with index $j + 1$ the smallest key had index at most $j + 1$. Lines 4-5 would ensure that if the index were $j + 1$ the key would switch with the j^{th} one, so that now the smallest key is at index j. The array `A[:i]` doesn't change because there isn't access to its elements.

Termination: When the loop terminates, we have $j = i$, so the smallest key of `A[i:]` is at index i, and the array `A[:i]` stays the same.

□

c.

Proposition 2.3.2. *Before the* i^{th} *iteration of loop 2-5, the array* `A[:i]` *is sorted and has the* $i - 1$ *smallest keys of* `A`*.*

Proof. **Initialization:** At the start, `A[:i]` is empty.

Maintenance: Assume that `A[:i-1]` is sorted before the $(i-1)^{\text{th}}$ iteration and has the $i - 2$ smallest keys of `A`. Since at the end of loop 3-5 the $(i-1)^{\text{th}}$ smallest element is at the $(i-1)^{\text{th}}$ place, and the array `A[:i-1]` doesn't change, we get that the array `A[:i]` is sorted with the smallest $i - 1$ keys before the i^{th} iteration.

Termination: When we are done, `i = A.length`, so the loop invariant says that the array `A[:A.length] = A[1,...,n-1]` is sorted with the $n-1$ smallest keys of `A`. This is equivalent to `A` being sorted.

□

d. The worst-case running time of bubblesort is $n-1+(n-2)+\ldots+1=\frac{n^2-n}{2}$ swaps and comparisons, which is $\Theta\left(n^2\right)$ where n is the length of `A`. But, bubblesort always has $\frac{n^2-n}{2}$ comparisons, while insertion sort on the average case need to bubble each key down only half of the way, and so has approximately $\frac{n^2-n}{4}$ comparisons on average. Hence, insertion sort is faster than bubblesort.

Problem 2-3 (Correctness of Horner's rule)**.** a. The loop 2-3 runs $n+1$ times, so the running time is $n+1=\Theta(n)$.

b. We compute a naive evaluation of the polynomial by looking at the sum expression.

```
1 def polynomial_evaluation(coefficients,x):
2     sum = 0
3     for [k,a_k] in enumerate(coefficients):
4         sum += a_k * x**k
```

This has time-complexity $\Theta(n)$ if exponantiation is in constant-time, but has more multiplications and if we consider these time-complexity is $\sum_{k=2}^{n-1}=\frac{(n-3)n+1}{2}=\Theta\left(n^2\right)$, which is worse.

c. Before the termination at $i=-1$ we would have

$$y=\sum_{k=0}^{n-(-1)-1}a_kx^k=\sum_{k=0}^{n}a_kx^k,$$

as required.

d. The loop invariant indeed holds, so by the above the code computes the polynomial correctly.

Initialization: Before $i=n$ we have $y=0=\sum_{k=0}^{-1}a_{k+n+1}x^k$.

Maintenance: If

$$y=\sum_{k=0}^{n-(i+1)-1}a_{k+(i+1+1)}x^k=\sum_{k=0}^{n-(i+2)}a_{k+i+2}x^k$$

before the iteration with index $i+1$, before the iteration with index i we have

$$\begin{aligned}
y&=a_{i+1}+x\sum_{k=0}^{n-(i+2)}a_{k+i+2}x^k\\
&=a_{i+1}+\sum_{k=0}^{n-(i+2)}a_{k+i+2}x^{k+1}\\
&=a_{i+1}+\sum_{j=i+2}^{n}a_jx^{j-i-1}\\
&=\sum_{j=i+1}^{n}a_jx^{j-i-1}\\
&=\sum_{k=0}^{n-i-1}a_{k+i+1}x^k,
\end{aligned}$$

as required.

Termination: When $i=-1$ we get $y=P(x)$ as discussed above.

Problem 2-4 (Inversions)**.** a. The five inversions are

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

b. The most inversions are obtained if all pairs are inversions, which is the case for the array `[n,n-1,...,2,1]`. It has $\binom{n}{2} = \frac{n(n-1)}{2}$ inversions.

c. The number of inversions is exactly the number of times loop 5-7 of `Inversion-Sort` runs. In order for the array to be sorted, each key has to be compared with all keys greater than it to its left, and since the array of elements to its left is already sorted we compare the key with at most one element smaller than it.

d. We change `merge` to return the number of inversions it encounters. When an element from the right array is added before the left array is complete, it moves to the left of `len(left[i:])` elements, contributing that many inversions.

```
1  def merge(arr,p,q,r):
2      left = []
3      right = []
4      for i in range(q-p+1):
5          left[i] = arr[p+i]
6      for i in range(r-q):
7          right[i] = arr[q+i+1]
8      i = 0
9      j = 0
10     inversions = 0
11     for k in range(p,r+1):
12         if(i >= len(left)):
13             if(j >= len(right)):
14                 return None
15             arr[k] = right[j]
16             j += 1
17         elif(j >= len(right)):
18             arr[k] = left[i]
19         elif(left[i] \leq right[j]):
20             arr[k] = left[i]
21             i += 1
22         else:
23             arr[k] = right[j]
24             j += 1
25             inversions += len(left[i:])
26     return inversions
```

We then change `merge_sort` to return the sum of these numbers.

```
1  def merge_sort(arr,p,r):
2      if p < r:
3          inversions = 0
4          q = int((p+r)/2)
5          inversions += merge_sort(arr,p,q)
6          inversions += merge_sort(arr,q+1,r)
7          inversions += merge(arr,p,q,r)
8          return inversions
```

3 Characterizing Running Times

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

Exercise 3.1-1. Let A be an array of any size $n \in \mathbb{N}_+$. Write $n = 3m + r$ for $r \in \{0, 1, 2\}$. Suppose that in A, the m largest values occupy the first m array positions $A[1 : m]$. Once the array has been sorted, each of these m values ends up somewhere in the last m positions $A[2m + r + 1 : n]$. For that to happen, each of these m values must pass through each of the middle $n - 2m = m + r$ positions $A[m + 1 : 2m + r]$. Each of these m values passes through these middle $m + r$ positions one position at a time, by at least m executions of line 6. Because at least m values have to pass through at least m positions, the time taken by `INSERTION-SORT` in the worst case is at least proportional to m^2. We have $m = \frac{n-r}{3} \geq \frac{n}{3}$, so $m^2 \geq \left(\frac{n}{3}\right)^2 = \frac{n^2}{9}$. We get that insertion sort is $\Omega\left(\frac{n^2}{9}\right) = \Omega\left(n^2\right)$.

Exercise 3.1-2. We refer to `selection_sort` defined in Exercise 2.2-3.

The procedure has nested `for` loops. The outer loop runs $n - 1$ times and the inner loop runs n times, regardless of the values being sorted. The body of the inner loop takes constant time per iteration. This suffices to see that the algorithm runs in $\Theta((n - 1) n) = \Theta\left(n^2\right)$-time.

Exercise 3.1-3. Let $\alpha \in (0, 1)$ and let $m := \lfloor \alpha n \rfloor$. Assume that the m largest values start in the first m positions. Then each of the m largest elements has to pass through the middle $n - 2m$ elements, which takes $m(n - 2m) = -2m^2 + mn$ computations.

Let $f(n, \alpha) := -2m^2 + mn$. We find the value of α which maximizes $f(n, \alpha)$. Deriving by m and comparing to 0 we get $-4m + n = 0$ so that $m = \frac{n}{4}$. Since $\frac{n}{4}$ isn't necessarily an integer, the actual maximum is obtained at the value $\lfloor \frac{n}{4} \rfloor$ or $\lceil \frac{n}{4} \rceil$ closer to $\frac{n}{4}$. If $\lfloor \frac{n}{4} \rfloor$ is closer, we can take $\alpha = \frac{1}{4}$, such that $m = \lfloor \frac{n}{4} \rfloor$. Otherwise, we can find α such that $\alpha n = \lfloor \frac{n}{4} \rfloor + 1$, which is given by $\alpha = \frac{\lfloor \frac{n}{4} \rfloor + 1}{n} = \frac{1}{4} + \frac{1}{n}$.

To generalize the lower bound, we compute

$$\begin{aligned} f(n, \alpha) &= -2 \lfloor \alpha n \rfloor^2 + \lfloor \alpha n \rfloor n \\ &\geq -2 (\alpha n)^2 + (\alpha n - 1) n \\ &= -2\alpha^2 n^2 + \alpha n^2 - n \\ &= \left(\alpha - 2\alpha^2\right) n^2 - n. \end{aligned}$$

When $\alpha - 2\alpha^2 > 0$, we get that $f(n, \alpha) = \Omega\left(\left(\alpha - 2\alpha^2\right) n^2 - n\right) = \Omega\left(n^2\right)$. This is always the case, since $\alpha - 2\alpha^2 = \alpha(1 - \alpha)$ and since $\alpha \in (0, 1)$.

3.2 Asymptotic notation: formal definitions

Exercise 3.2-1. We have to prove there exist constants $c_1, c_2 > 0$ and $n_0 \in \mathbb{N}$ such that

$$c_1 (f(n) + g(n)) \leq \max(f(n), g(n)) \leq c_2 (f(n) + g(n))$$

for all $n \geq n_0$.

Taking $c_1 = \frac{1}{2}$ we get

$$\forall n \in \mathbb{N} : \max(f(n), g(n)) \geq \frac{f(n) + g(n)}{2} = c_1 (f(n) + g(n)),$$

since the maximum of non-negative numbers is at least as big as their average. Taking $c_2 = 1$ we get

$$\forall n \in \mathbb{N} : \max(f(n), g(n)) \leq f(n) + g(n) = c_2 (f(n) + g(n)),$$

since the maximum of non-negative numbers is no larger than their sum.

Hence taking $c_1 = \frac{1}{2}, c_2 = 1, n_0 = 0$ gives the result.

Exercise 3.2-2. The class $P\left(n^2\right)$ is the class of functions that grow asymptotically at most as large as n^2. Therefore, saying that something is "at least $O\left(n^2\right)$" is saying that it grows "at least not as large as n^2", which is nonsensical.

Exercise 3.2-3. We have $2^{n+1} = 2 \cdot 2^n$. Since $O(\alpha f(n)) = O(f(n))$ for all constant α and function f, we get $O\left(2^{n+1}\right) = O(2 \cdot 2^n) = O(2^n)$.

However,

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

so

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

It follows that $2^{2n} = \omega(2^n)$ which would contradict $2^{2n} = O(2^n)$.

Exercise 3.2-4. We prove Theorem 3.1. which states that $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$.

Proof.

- Assume that $f(n) = \Theta(g(n))$. There are constants c_1, c_2, n_0 such that

  $$\forall n \geq n_0 : 0 < c_1 g(n) \leq f(n) \leq c_2 g(n).$$

  Ignoring the term $c_1 g(n)$ in the expression we get $0 \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$, so that $f(n) = O(g(n))$. Ignoring the term $c_2 g(n)$ we get $0 \leq c_1 g(n) \leq f(n)$ for all $n \geq n_0$, so that $f(n) = \Omega(g(n))$.

- Assume that $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. By the first assumption, there are constants n_2, c_2 such that $0 \leq f(n) \leq c_2 g(n)$ for all $n \geq n_1$. By the second assumption, there are constants n_1, c_1 such that $0 \leq c_1 g(n) \leq f(n)$ for all $n \geq n_2$. Taking $n_0 = \max(n_1, n_2)$ we get that

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

  for all $n \geq n_0$, which means that $f(n) = \Theta(g(n))$.

□

Exercise 3.2-5. Let $f_-(n), f_+(n)$ be the best an worst running time for the algorithm given input of size n, respectively.

If the running time of the algorithm is $\Theta(g(n))$, so are the best and worst running times, so $f_-(n), f_+(n) = \Theta(g(n))$ and in particular $f_+(n) = O(g(n))$ and $f_-(n) = \Omega(g(n))$.

Assume that $f_+(n) = O(n)$ and $f_-(\Omega)(g(n))$. For any input a of size n the running time $f(a)$ satisfies $f_-(n) \leq f(a) \leq f_+(n)$. Let c_1, n_1 be such that $0 \leq c_1 g(n) \leq f_-(n)$ for all $n \geq n_1$, and let c_2, n_2 be such that $0 \leq f_+(n) \leq c_2 g(n)$ for all $n \geq n_2$. Taking $n_0 = \max(n_1, n_2)$ we get

$$0 \leq c_1 g(n) \leq f_-(n) \leq f(a) \leq f_+(n) \leq c_2 g(n)$$

for all $n \geq n_0$. Hence the running time of the running time of the algorithm is $\Theta(g(n))$.

Exercise 3.2-6. Assume towards a contradiction there exists $f \in o(g(n)) \cap \omega(g(n))$. Since $f = o(g(n))$, we have

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

Since $f = o(g(n))$, we have

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty.$$

By the uniqueness of the limit, we get $0 = \infty$, a contradiction.

Exercise 3.2-7. We give the following corresponding definitions.

$$\Omega(g(n,m)) = \{f(n,m) \mid \exists c, n_0, m_0 > 0 : \forall n, m : (n \geq n_0 \vee m \geq m_0) \to 0 \leq cg(n,m) \leq f(n)\}$$
$$\Omega(g(n,m)) = \{f(n,m) \mid \exists c_1, c_2, n_0, m_0 > 0 : \forall n, m : (n \geq n_0 \vee m \geq m_0) \to 0 \leq c_1 g(n,m) \leq f(n) \leq c_2 g(n,m)\}$$

3.3 Standard notation and common functions

Exercise 3.3-1. Assume that f, g are monotonically increasing.

- Let $h(n) = f(n) + g(n)$. Since for $n > m$ we have $f(n) > f(m), g(n) > g(m)$, and since $a > c, b > d$ implies $a + b > c + d$, we have

  $$h(n) = f(n) + g(m) > f(m) + g(m) = h(m),$$

  so h is monotonically increasing.

- Let $h(n) \coloneqq f(g(n))$ and let $n > m \in \mathbb{N}$. Since g is monotonically increasing, we have $g(n) > g(m)$. Since f is monotonically increasing, it follows that

  $$h(n) = f(g(n)) > f(g(m)) = h(m),$$

  so h is also monotonically increasing.

- Assume that f, g are non-negative, and let $n > m \in \mathbb{N}$. Then $f(n) > f(m), g(n) > g(m)$. Since $a > c, b > d$ implies $a \cdot b > c \cdot d$ for positive numbers, we get that

  $$h(n) = f(n) \cdot g(n) > f(m) \cdot g(m) = h(m).$$

  Hence h is monotonically increasing.

Exercise 3.3-2. Let $n \in \mathbb{N}$ and let $\alpha \in [0, 1]$ be a real number. We have

$$\lfloor \alpha n \rfloor + \lceil (1 - \alpha) n \rceil = \lfloor \alpha n \rfloor + \lceil n - \alpha n \rceil.$$

We have

$$\forall x \in \mathbb{R} : \lceil n - x \rceil = \lceil n + (-x) \rceil = n + \lceil -x \rceil = n - \lfloor x \rfloor,$$

so that

$$\lfloor \alpha n \rfloor + \lceil n - \alpha n \rceil = \lfloor \alpha n \rfloor + n - \lfloor \alpha n \rfloor = n.$$

Exercise 3.3-3. Let $k \in \mathbb{R}$ and let $f(n) = \mathrm{o}(n)$. We have

$$\frac{(n + f(n))^k}{n^k} = \left(1 + \frac{f(n)}{n}\right)^k \leq e^{k \frac{f(n)}{n}}.$$

Since $f(n) = \mathrm{o}(n)$, we have $\lim_{n \to \infty} \frac{f(n)}{n} = 0$, hence also $\lim_{n \to \infty} k \frac{f(n)}{n} = 0$ so that

$$\lim_{n \to \infty} e^{k \frac{f(n)}{n}} = 1.$$

We get that

$$\lim_{n \to \infty} \frac{(n + f(n))^k}{n^k} = 1 \in (0, \infty),$$

which implies that $(n + f(n))^k = \Theta\left(n^k\right)$. Since this is true for every $f(n) = \mathrm{o}(n)$, we get that $(n + \mathrm{o}(n))^k = \Theta\left(n^k\right)$.

In particular, we have $(n - 1)^k, (n + 1)^k = \Theta\left(n^k\right)$ so this is also the case for any function between these, such as $\lfloor n \rfloor^k, \lceil n \rceil^k$.

Exercise 3.3-4. a. We prove Equation (3.21).

Proof. We have

$$a^{\log_b c} = \left(b^{\log_b a}\right)^{\log_b c} = \left(b^{\log_b c}\right)^{\log_b a} = c^{\log_b a}.$$

□

b. We prove Equations (3.26)-(3.28).

Proof (Equation (3.26)). Using Stirling's formula we get

$$\begin{aligned}\lim_{n\to\infty}\frac{n!}{n^n} &= \lim_{n\to\infty}\frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^e\left(1+\Theta\left(\frac{1}{n}\right)\right)}{n^n}\\ &= \lim_{n\to\infty}\frac{\sqrt{2\pi n}\left(1+\Theta\left(\frac{1}{n}\right)\right)}{e^n}\\ &= \lim_{n\to\infty}\left(\frac{\sqrt{2\pi n}}{e^n}+\frac{\Theta\left(\frac{1}{n}\right)}{e^n}\right).\end{aligned}$$

We have

$$\lim_{n\to\infty}\frac{\sqrt{2\pi n}}{e^n}=0$$

and if $f=\Theta\left(\frac{1}{n}\right)$ we have

$$\begin{aligned}\frac{f(n)}{e^n} &= \frac{\frac{f(n)}{\frac{1}{n}}}{\frac{e^n}{\frac{1}{n}}}\\ &= \frac{\frac{f(n)}{\frac{1}{n}}}{ne^n}\end{aligned}$$

where the numerator is bounded between positive numbers and the denominator approaches infinity. Hence $\lim_{n\to\infty}\frac{f(n)}{e^n}=0$, so $\lim_{n\to\infty}\frac{\Theta(\frac{1}{n})}{e^n}=0$. We get that $\lim_{n\to\infty}\frac{n!}{n^n}=0$, so that $n!=o\left(n^n\right)$. □

Proof (Equation (3.27)). We have

$$\frac{n!}{2^n}=\frac{n}{2}\cdot\frac{n-1}{2}\cdot\ldots\frac{4}{2}\cdot\frac{3}{2}\cdot1\cdot\frac{1}{2}.$$

For $n\geq5$ the terms $\frac{4}{2},\frac{1}{2}$ cancel, so $\frac{n!}{2^n}\geq\frac{n}{2}\xrightarrow{n\to\infty}\infty$, hence $n!=\omega\left(2^n\right)$. □

Proof (Equation (3.28)). We have

$$\log(n!)=\sum_{k\in[n]}\log(k)\leq\sum_{k\in[n]}\log(n)=n\log(n),$$

hence $\log(n!)=O(n\log n)$.

On the other hand,

$$\begin{aligned}\log(n!) &= \sum_{k\in[n]}\log(k)\\ &\geq \sum_{k=\lfloor\frac{n}{2}\rfloor}^{n}\log(k)\\ &\geq \sum_{k=\lfloor\frac{n}{2}\rfloor}^{n}\log\left(\left\lfloor\frac{n}{2}\right\rfloor\right)\\ &= \left\lceil\frac{n}{2}\right\rceil\log\left(\left\lfloor\frac{n}{2}\right\rfloor\right)\\ &\geq \left\lfloor\frac{n}{2}\right\rfloor\log\left(\left\lfloor\frac{n}{2}\right\rfloor\right)\end{aligned}$$

and since $n\log n$ is monotonic, this is at least $\frac{n}{4}\log\left(\frac{n}{4}\right)$ for $n\geq4$. We get Hence Then

$$\begin{aligned}\log(n!) &= \Omega\left(\frac{n}{4}\log\left(\frac{n}{4}\right)\right)\\ &= \Omega\left(n\left(\log(n)-\log(4)\right)\right)\\ &= \Omega\left(n\log n\right).\end{aligned}$$

□

c. We prove that $\log(\Theta(n)) = \Theta(\log(n))$.

Let $f(n) = \Theta(n)$ and let $h(n) = \log(f(n))$. We have

$$\begin{aligned}\lim_{n\to\infty} \frac{h(n)}{\log n} &= \lim_{n\to\infty} \frac{\log(f(n))}{\log n} \\ &= \lim_{n\to\infty} \log_n(f(n)).\end{aligned}$$

Since $f(n) = \Theta(n)$, there are constants c_1, c_2, n_0 such that

$$\forall n \geq n_0 : 0 < c_1 n \leq f(n) \leq c_2 n.$$

Since $\log_n(\cdot)$ is monotonic, we get that

$$\log_n(c_1) + 1 = \log_n(c_1 n) \leq \log_n(f(n)) \leq \log_n(c_2 n) \leq \log_n(c_2) + 1.$$

Since $\log_n(c_i)$ approaches ∞ for both $i \in \{1, 2\}$, we get that

$$\lim_{n\to\infty} \frac{h(n)}{\log n} = \lim_{n\to\infty} \log_n(f(n)) = 1.$$

Since the limit is finite and positive, we get that $h(n) = \Theta(\log(n))$.

Exercise 3.3-5. 1. Let $m = \lceil \log n \rceil$. By Stirling's formula,

$$\begin{aligned}m! &= \sqrt{2\pi m}\left(\frac{m}{e}\right)^m \left(1 + \Theta\left(\frac{1}{m}\right)\right) \\ &= \Theta\left(\sqrt{m}\left(\frac{m}{e}\right)^m\right) \\ &= \Theta\left(\sqrt{\lceil \log n \rceil}\left(\frac{\lceil \log n \rceil}{e}\right)^{\lceil \log n \rceil}\right) \\ &= \Omega\left(\left(\frac{\log n}{e}\right)^{\log n}\right).\end{aligned}$$

Now,

$$\begin{aligned}\left(\frac{\log n}{e}\right)^{\log n} &= \frac{\log(n)^{\log n}}{e^{\log n}} \\ &= \frac{2^{\log\log n \cdot \log n}}{n^{\log e}} \\ &= \frac{n^{\log\log n}}{n^{\log e}}\end{aligned}$$

which isn't polynomially bounded, hence neither is $\lceil \log n \rceil!$.

2. Let $m = \lceil \log\log n \rceil$. By Equation (3.26) we have

$$\begin{aligned}\log(m!) &= \Theta(m \log m) \\ &= \Theta(\lceil \log\log n \rceil \log \lceil \log\log n \rceil) \\ &= \mathrm{O}(2\log\log n \cdot \log(2\log\log n)) \\ &= \mathrm{O}(\log\log n \cdot (1 + \log\log\log n))\end{aligned}$$

where the last function is is polynomially-bounded as a polylogarithmic function.

Exercise 3.3-6. We have

$$\begin{aligned}\log^*(\log n) &= \min\left\{i \;\middle|\; \log^{(i)}(\log n) \le 1\right\}\\ &= \min\left\{i \;\middle|\; \log^{(i+1)}(n) \le n\right\}\\ &= \min\left\{i-1 \;\middle|\; \log^{(i)}(n) \le 1\right\}\\ &= \log^*(n) - 1,\end{aligned}$$

so this is asymptotically larger than $\log(\log^*(n))$.

Exercise 3.3-7. We have

$$\phi^2 = \frac{1+2\sqrt{5}+5}{4} = \frac{2+2\sqrt{5}}{4} + 1 = \phi + 1$$

and

$$\hat{\phi}^2 = \frac{1-2\sqrt{5}+5}{4} = \frac{2-2\sqrt{5}}{4} + 1 = \hat{\phi} + 1.$$

Exercise 3.3-8. Base: We have $F_0 = 0 = \frac{1-1}{5}$.

Step: Let $n \in \mathbb{N}_+$ and assume that $F_i = \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}}$ for all $i < n$. Then

$$\begin{aligned}F_n &= F_{n-1} + F_{n-2}\\ &= \frac{\phi^{n-1} - \hat{\phi}^{n-1}}{\sqrt{5}} + \frac{\phi^{n-2} - \hat{\phi}^{n-2}}{\sqrt{5}}\\ &= \frac{\phi^{n-1} - \hat{\phi}^{n-1} + \phi^{n-2} - \hat{\phi}^{n-2}}{\sqrt{5}}\\ &= \frac{\phi^{n-2}(\phi+1) - \hat{\phi}^{n-2}\left(\hat{\phi}+1\right)}{\sqrt{5}}\\ &= \frac{\phi^{n-2}\cdot\phi^2 - \hat{\phi}^{n-2}\cdot\hat{\phi}^2}{\sqrt{5}}\\ &= \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}},\end{aligned}$$

where in the second-to-last equation we used Exercise 3.3-7.

Exercise 3.3-9. Let $c_1, c_2 > 0$ be such that $c_1 n \le k \ln k \le c_2 n$. Then $\ln(c_1) + \ln(n) \le \ln k + \ln(\ln k) = \Theta(\ln k)$ so $\ln n = \Theta(\ln k)$. Let $d_1, d_2 > 0$ such that $d_1 \ln k \le \ln n \le d_2 \ln k$. Then

$$\begin{aligned}\frac{k}{c_2 d_2} &= \frac{k \ln k}{c_2 d_2 \ln k}\\ &\le \frac{n}{d_2 \ln k}\\ &\le \frac{n}{\ln n}\\ &\le \frac{n}{d_1 \ln k}\\ &\le \frac{k \ln k}{c_1 d_1 \ln k}\\ &= \frac{k}{c_1 d_1}.\end{aligned}$$

We get

$$c_1 d_1 \frac{n}{\ln n} \le k \le c_2 d_2 \frac{n}{\ln n},$$

which gives $k = \Theta\left(\frac{n}{\ln n}\right)$.

Problems

Problem 3-1 (Asymptotic behaviour of polynomials)**.** a. We have $\lim_{n\to\infty} \frac{n^i}{n^k} = 0$ for all $i < k$, hence we'd get

$$\limsup_{n\to\infty} \frac{p(n)}{n^k} = \begin{cases} 0 & k > d \\ a_d & k = d \end{cases} < \infty$$

b. We have

$$\liminf_{n\to\infty} \frac{p(n)}{n^k} = \begin{cases} \infty & k < d \\ a_d & k = d \end{cases} > 0.$$

c. We have both the previous conditions, hence by **??** we get the result.

d. From the above, we have $\limsup_{n\to\infty} \frac{p(n)}{n^k} = 0$.

e. From the above, we have $\liminf_{n\to\infty} \frac{p(n)}{n^k} = \infty$.

Problem 3-2 (Relative asymptotic growths)**.** We fill in the table as follows.

A	B	O	o	Ω	ω	Θ
$\log^k n$	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
$n^{\log c}$	$c^{\log n}$	yes	no	yes	no	yes
$\log(n!)$	$\log(n^n)$	yes	no	yes	no	yes

We've shown that polylogarithmic functions are smaller asymptotically of polynomial functions, and that $\log(n!)$ and $\log(n^n) = n\log n$ are both $\Theta(n\log n)$. The rest are easily verified.

Problem 3-3 (Ordering by asymptotic growth rates)**.** We notice a few facts about the listed functions, and then list them according to order.

a. (1) We have $2^{\log n} = n$ and $4^{\log n} = \left(2^2\right)^{\log n} = \left(2^{\log n}\right)^2 = n^2$.

(2) We saw in [Cor+22, (3.19)] that $\log(n!) = \Theta(n\log n)$.

(3) We have $(n+1)! = (n+1)\cdot n!$ so $\lim_{n\to\infty} \frac{(n+1)!}{n!} = \lim_{n\to\infty} n+1 = \infty$, hence $n! = \mathrm{o}((n+1)!)$.

(4) We have $n^{\log\log n} = 2^{\log n\cdot\log\log n} = (\log n)^{\log n}$. Furthermore, we easily see that this is ω of any polynomial function, because the power goes to infinity.

(5) We saw in Exercise 3.3-6 that $\log^*(\log n) = \Theta(\log^* n)$.

(6) We have $\left(\sqrt{2}\right)^{\log n} = 2^{\frac{1}{2}\log n} = n^{\frac{1}{2}} = \sqrt{n}$.

(7) We show that if $f(n) = (\log^*(n))^k$ for some $k \geq 1$, then $f(n) = \mathrm{o}\left(\log^{(i)} n\right)$ for all $i \in \mathbb{N}_+$, which implies that any poly-$\log^*$ function is $\mathrm{o}\left(\log^{(i)} n\right)$.

Let $a_0 = 1$ and $a_n = 2^{a_{n-1}}$ for all $n \geq 1$. We get that $\log^* a_n = n$ for all $n \in \mathbb{N}$ and that $\log^{(i)} a_n = a_{n-i}$. We also notice that since $\log^*$ doesn't increase on the interval $(a_n, a_{n+1}]$, we have

$$\frac{(\log^* m)^k}{\log^{(i)} m} \leq \frac{(\log^* a_{n+1})^k}{\log^{(i)} m} = \frac{(n+1)^k}{\log^{(i)} a_n} = \frac{(n+1)^k}{a_{n-1}}$$

for all m in this interval. Since $(n+1)^k$ is $\mathrm{o}(a_n)$, this expression goes to 0 as $n \to \infty$ and therefore as $m \to \infty$. Hence $(\log^* n)^k = \mathrm{o}\left(\log^{(i)} m\right)$.

We now list the functions and show the relevant relations.

1. Let $g_1(n) = 1$.

2. Let $g_2(n) = n^{1/\log n}$. Then $g_2(n) = \left(2^{\log n}\right)^{1/\log n} = 2^{\frac{\log n}{\log n}} = 2$, so $g_1(n) = \Theta(g_2(n))$.
3. Let $g_3(n) = \log(\log^* n)$. Since $\log(\log^* n) \xrightarrow{n\to\infty} = 0$, we have $g_2(n) = \mathrm{o}(g_3(n))$.
4. Let $g_4(n) = \log^* n$. Since $\log^* n$ approaches ∞ as $n \to \infty$, we have

$$\lim_{n\to\infty} \frac{\log(\log^* n)}{\log^* n} = \lim_{m\to\infty} \frac{\log m}{m} = 0,$$

   hence $\log(\log^* n) = \mathrm{o}(\log^* n)$, i.e. $g_3(n) = \mathrm{o}(g_4(n))$.
5. Let $g_5(n) = \log^*(\log n)$. We saw that $\log^*(\log n) = \Theta(\log^* n)$, so $g_4(n) = \Theta(g_5(n))$.
6. Let $g_6(n) = 2^{\log^* n}$. Since $\log^* n \xrightarrow{n\to\infty} 0$, we have

$$\lim_{n\to\infty} \frac{g_5(n)}{g_6(n)} = \lim_{n\to\infty} \frac{\log^* n}{2^{\log^* n}} = \lim_{m\to\infty} \frac{m}{2^m} = 0,$$

   so $g_5(n) = \mathrm{o}(g_6(n))$.
7. Let $g_7(n) = \ln\ln n$. We have $\ln\ln a_n = a_{n-2}$ and $2^{\log^* a_n} = 2^n$. For all $m \in (a_n, a_{n+1}]$ we have

$$\frac{2^{\log^* m}}{\ln\ln m} \leq \frac{2^{n+1}}{a_{n-2}} \xrightarrow{n\to\infty} 0,$$

   hence

$$\lim_{m\to\infty} \frac{2^{\log^* m}}{\ln\ln m} = 0$$

   so $g_6(n) = \mathrm{o}(g_7(n))$.
8. Let $g_8(n) = \sqrt{\log n}$. Then $g_8(n) = \Theta\left(\sqrt{\ln n}\right)$. We have $\sqrt{\ln n} \xrightarrow{n\to\infty} \infty$ so

$$\lim_{n\to\infty} \frac{g_7(n)}{g_8(n)} = \lim_{n\to\infty} \frac{\ln\ln n}{\sqrt{\ln n}} = \lim_{m\to\infty} \frac{\ln m}{\sqrt{m}} = 0,$$

   so $g_7(n) = \mathrm{o}(g_8(n))$.
9. Let $g_9(n) = \ln n$. We have

$$\lim_{n\to\infty} \frac{g_8(n)}{g_9(n)} = \lim_{n\to\infty} \frac{\sqrt{\ln n}}{\ln n} = \lim_{n\to\infty} \frac{1}{\sqrt{n}} = 0$$

   so $g_8(n) = \mathrm{o}(g_9(n))$.
10. Let $g_{10}(n) = \log^2(n)$. We have $\ln n = \frac{\log n}{\log e}$, hence

$$\lim_{n\to\infty} \frac{g_9(n)}{g_{10}(n)} = \lim_{n\ to\infty} \frac{\ln n}{\log^2 n} = \lim_{n\to\infty} \frac{1}{\log e \log n} = 0,$$

    so $g_9(n) = \mathrm{o}(g_{10}(n))$.
11. Let $g_{11}(n) = 2^{\sqrt{2\log n}}$. We have

$$\begin{aligned}
&\log\left(\frac{2^{\sqrt{2\log n}}}{\log^2(n)}\right) \\
&\qquad = \log\left(2^{\sqrt{2\log n}}\right) - \log\left(\log^2(n)\right) \\
&\qquad = \sqrt{2}\sqrt{\log n} - 2\log\log n \\
&\qquad \xrightarrow{n\to\infty} \infty
\end{aligned}$$

    so $g_{10}(n) = \mathrm{o}(g_{11}(n))$.

12. Let $g_{12}(n) = \left(\sqrt{2}\right)^{\log n} = \sqrt{n}$. For all $\alpha \in \mathbb{R}_+$ we have

$$\begin{aligned}\log\left(\frac{2^{\sqrt{2\log n}}}{n^\alpha}\right) &= \log\left(2^{\sqrt{2\log n}}\right) - \log\left(n^\alpha\right) \\ &= \sqrt{2}\sqrt{\log n} - \alpha\log n \\ &\xrightarrow{n\to\infty} -\infty\end{aligned}$$

so

$$\frac{2^{\sqrt{2\log n}}}{n^\alpha} \xrightarrow{n\to\infty} 0,$$

so $2^{\sqrt{2\log n}} = o\left(n^\alpha\right)$. Hence in particular $g_{11}(n) = \mathrm{o}\left(g_{12}(n)\right)$.

13. Let $g_{13}(n) = 2^{\log n} = n$. We have $\frac{g_{13}(n)}{g_{12}(n)} = \sqrt{n} \xrightarrow{n\to\infty} \infty$ so $g_{12}(n) = \mathrm{o}\left(g_{13}(n)\right)$.
14. Let $g_{14}(n) = n = g_{13}(n)$. Clearly $g_{13}(n) = \Theta\left(g_{14}(n)\right)$.
15. Let $g_{15}(n) = n\log n$. Then $\frac{g_{15}(n)}{g_{14}(n)} = \log n \xrightarrow{n\to\infty} \infty$, so $g_{14}(n) = \mathrm{o}\left(g_{15}(n)\right)$.
16. Let $g_{16}(n) = \log(n!)$. We saw that $g_{16}(n) = \Theta(n\log n)$, hence $g_{15}(n) = \Theta\left(g_{16}(n)\right)$.
17. Let $g_{17}(n) = 4^{\log n} = n^2$. We have

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

since $\log n = \mathrm{o}(n)$. Hence also $n\log n = \mathrm{o}\left(n^2\right)$. Since $g_{16} = \Theta(n\log n)$ we get that $g_{16}(n) = \mathrm{o}\left(n^2\right)$.

18. Let $g_{18}(n) = n^2$. Clearly $g_{17}(n) = \Theta\left(g_{18}(n)\right)$.
19. Let $g_{19}(n) = n^3$. We have

$$\lim_{n\to\infty}\frac{g_{18}(n)}{g_{19}(n)} = \lim_{n\to\infty}\frac{1}{n} = 0$$

hence $g_{18}(n) = \mathrm{o}\left(g_{19}(n)\right)$.

20. Let $g_{20}(n) = (\log n)!$. We have

$$(\log n)! = (\log n)(\log n - 1)\cdot\ldots\cdot 4\cdot 2 = (\log n)\left(\log\left(\frac{n}{2}\right)\right)\cdot\ldots\cdot 4\cdot 2.$$

Hence

$$\begin{aligned}g_{20}(n) &\geq \left(\frac{\log n}{2}\right)^{\frac{\log n}{2}} \\ &= \frac{(\log n)^{\frac{\log n}{2}}}{2^{\frac{\log n}{2}}} \\ &= \frac{\sqrt{\log n}^{\log n}}{\sqrt{n}}.\end{aligned}$$

Since $(\log n)^{\log n} = n^{\log\log n}$ by (4), we get

$$g_{20}(n) \geq \frac{n^{\frac{\log\log n}{2}}}{\sqrt{n}}.$$

Since the power $\frac{\log\log n}{2}$ approaches ∞, we get that $n^\alpha = \mathrm{o}\left(g_{20}(n)\right)$ for all $\alpha \geq 0$, hence $g_{19}(n) = \mathrm{o}\left(g_{20}(n)\right)$.

21. Let $g_{21}(n) = (\log n)^{\log n}$. We know that $n! = \mathrm{o}(n^n)$, hence $(\log n)! = \mathrm{o}\left((\log n)^{\log n}\right)$, so $g_{20}(n) = \mathrm{o}(g_{21}(n))$.
22. Let $g_{22}(n) = n^{\log\log n}$. We saw in (4) that $g_{21}(n) = g_{22}(n)$, hence $g_{21}(n) = \Theta(g_{22}(n))$.
23. Let $g_{23}(n) = \left(\frac{3}{2}\right)^n$. Let $a > 0$, we have

$$\begin{aligned}\log_a\left(\frac{a^n}{n^{\log\log n}}\right) &= n - \log_a\left(n^{\log\log n}\right)\\ &= n - \log\log n \log_a n \xrightarrow{n\to\infty} \infty\end{aligned}$$

so $\frac{a^n}{n^{\log\log n}} \xrightarrow{n\to\infty} \infty$, so $g_{22}(n) = \mathrm{o}(g_{23}(n))$.

24. Let $g_{24}(n) = 2^n$. For $b > a > 0$ we have

$$\frac{b^n}{a^n} = \left(\frac{b}{a}\right)^n \xrightarrow{n\to\infty} \infty.$$

Hence $a^n = \mathrm{o}(b^n)$, so $g_{23}(n) = \mathrm{o}(g_{24}(n))$.

25. Let $g_{25}(n) = n2^n$. We have $\frac{g_{24}(n)}{g_{25}(n)} = \frac{1}{n} \xrightarrow{n\to\infty} 0$, hence $g_{24}(n) = \mathrm{o}(g_{25}(n))$.
26. Let $g_{26}(n) = e^n$. Denote $\beta \coloneqq \frac{2}{e} < 1$. We have

$$\begin{aligned}\frac{g_{25}(n)}{g_{26}(n)} &= n\left(\frac{2}{e}\right)^n\\ &= \frac{n}{\beta^n}.\end{aligned}$$

Hence,

$$\begin{aligned}&\log_\beta\left(\frac{g_{25}(n)}{g_{26}(n)}\right)\\ &\qquad = \log_\beta n - n \xrightarrow{n\to\infty} -\infty\end{aligned}$$

hence

$$\lim_{n\to\infty}\frac{g_{25}(n)}{g_{26}(n)} = 0$$

so $g_{25}(n) = \mathrm{o}(g_{26}(n))$.

27. Let $g_{27}(n) = n!$. Let $a > 0$, we show that $a^n = \mathrm{o}(n!)$ which shows that $g_{26}(n) = \mathrm{o}(g_{27}(n))$. Indeed,

$$n! \geq \left(\frac{n}{2}\right)^{\frac{n}{2}} = \left(\frac{\sqrt{n}}{\sqrt{2}}\right)^n$$

so

$$\frac{a^n}{n!} = \frac{\left(\sqrt{2}a\right)^n}{\sqrt{n}^n} = \left(\frac{\sqrt{2}a}{\sqrt{n}}\right)^n.$$

Since $\sqrt{n} \xrightarrow{n\to\infty} \infty$, the fraction goes to 0 and in particular

$$\frac{a^n}{n!} \leq \frac{1}{n} \xrightarrow{n\to\infty} 0.$$

We get that $\frac{a^n}{n!} \xrightarrow{n\to\infty} 0$ so $a^n = \mathrm{o}(n!)$ as required.

28. Let $g_{28}(n) = (n+1)!$. We have $\frac{g_{27}(n)}{g_{28}(n)} = \frac{1}{n+1} \xrightarrow{n\to\infty} 0$, so $g_{27}(n) = \mathrm{o}(g_{28}(n))$.

29. Let $g_{29}(n) = 2^{2^n}$. We have

$$\log\left(\frac{g_{28}(n)}{g_{29}(n)}\right) = \log(n!) - 2^n.$$

We saw that $\log(n!) = \Theta(n \log n)$ so $\log(n!) = \mathrm{o}\left(n^2\right)$, and since $n^2 = \mathrm{o}(2^n)$ (since any polynomial function is o of any exponential one) we have that

$$\log\left(\frac{g_{28}(n)}{g_{29}(n)}\right) \xrightarrow{n\to\infty} -\infty$$

so

$$\frac{g_{28}(n)}{g_{29}(n)} \xrightarrow{n\to\infty} 0$$

so $g_{28}(n) = \mathrm{o}(g_{29}(n))$.

30. Let $g_{30}(n) = 2^{2^{n+1}}$. We have

$$\begin{aligned}\frac{g_{29}(n)}{g_{30}(n)} &= 2^{2^n - 2^{n+1}} \\ &= 2^{2^n(1-2)} \\ &= 2^{-2^n} \\ &= \left(2^{2^n}\right)^{-1}.\end{aligned}$$

Since $2^{2^n} \xrightarrow{n\to\infty} \infty$ we get that $\frac{g_{29}(n)}{g_{30}(n)} \xrightarrow{n\to\infty} 0$, so $g_{29}(n) = \mathrm{o}(g_{30}(n))$.

b. Take $f(n) = n g_{30}(n) \chi_{2\mathbb{Z}} + \frac{1}{n}(n) \chi_{2\mathbb{Z}+1}$. Then the partial limits of each $\frac{f(n)}{(g_i(n))}$ are 0 and ∞. Hence

$$\begin{aligned}\limsup_{n\to\infty} \frac{f(n)}{g_i(n)} &= \infty \\ \liminf_{n\to\infty} \frac{f(n)}{g_i(n)} &= 0\end{aligned}$$

which are equivalent to $g_i(n) \neq \Omega(f(n))$ and $g_i(n) \neq \mathrm{O}(f(n))$, respectively.

Problem 3-4 (Asymptotic notation properties)**.** a. No. $n = \mathrm{O}\left(n^2\right)$ since $\limsup_{n\to\infty} \frac{n}{n^2} < \infty$, but $\limsup_{n\to\infty} \frac{n^2}{n} = \limsup_{n\to\infty} n = \infty$ so $n^2 \neq \mathrm{O}(n)$.

b. No. Taking $f(n) = n$ and $g(n) = n^2$, we have $f(n) + g(n) = n + n^2 = \Theta\left(n^2\right)$ but $\Theta\left(\min\left(n, n^2\right)\right) = \Theta(n)$ and $n \neq \Theta\left(n^2\right)$.

c. Yes. Taking f, g as described, we have $\limsup_{n\to\infty} \frac{f(n)}{g(n)} < \infty$ so there's $C > 0$ such that for all n large enough $f(n) \leq C g(n)$. Then for all such n we have

$$\frac{\log(f(n))}{\log(g(n))} \leq \frac{\log(cg(n))}{\log(g(n))} = \frac{\log c + \log(g(n))}{\log(g(n))} \xrightarrow{n\to\infty} 1.$$

Hence

$$\limsup_{n\to\infty} \frac{\log(f(n))}{\log(g(n))} \leq 1,$$

so $\log(f(n)) = \mathrm{O}(\log(g(n)))$.

d. No. Let $f(n) = 2n$ and $g(n) = n$. Then clearly $f(n) = \mathrm{O}(g(n))$, but

$$\frac{2^{f(n)}}{2^{g(n)}} = \frac{2^{2n}}{2^n} = 2^n \xrightarrow{n\to\infty} \infty$$

and since the limit is not finite, $2^{f(n)}$ is not $\mathrm{O}\left(2^{g(n)}\right)$.

e. No. Take $f(n) = \frac{1}{n}$. Then

$$\frac{f(n)}{f(n)^2} = n \xrightarrow{n\to\infty} \infty.$$

f. Yes. Assume $f(n) = \mathrm{O}(g(n))$. This is equivalent to $\limsup_{n\to\infty} \frac{f(n)}{g(n)} < \infty$. Then $\liminf_{n\to\infty} \frac{g(n)}{f(n)} > 0$, which is equivalent to $g(n) = \Omega(f(n))$.

g. No. Take $f(n) = 2^{2n}$. Then $f(n/2) = 2^n$. We saw in Item d. that these are asymptotically different.

h. Yes. Let $g(n) = \mathrm{o}(f(n))$. We have

$$\lim_{n\to\infty} \frac{f(n) + g(n)}{f(n)} = \lim_{n\to\infty} \frac{f(n)}{f(n)} + \lim_{n\to\infty} \frac{g(n)}{f(n)} = 1 \in (0, \infty).$$

Problem 3-5 (Manipulating Asymptotic Notation)**.** a. Let $g(n) = \Theta(\Theta(f(n)))$. I.e. there's $h(n) = \Theta(f(n))$ such that $g(n) = \Theta(h(n))$. By transitivity we get $g = \Theta(f(n))$.

b. Let $g(n) = \Theta(f(n))$ and $h(n) = O(f(n))$. We have

$$\limsup_{n\to\infty} \frac{f(n) + h(n)}{f(n)} = 1 + \limsup_{n\to\infty} \frac{h(n)}{f(n)},$$

and this is a finite positive number since $h = \mathrm{O}(f(n))$ is asymptotically positive. Hence $g(n) + h(n) = \Theta(f(n))$.

c. Let $F(n) = \Theta(f(n))$ and $G(n) = \Theta(g(n))$. We have to show that $F(n) + G(n) = \Theta(f(n) + g(n))$. Indeed,

$$\begin{aligned}\limsup_{n\to\infty} \frac{F(n) + G(n)}{f(n) + g(n)} &\le \limsup_{n\to\infty} \left(\frac{F(n)}{f(n) + g(n)}\right) + \limsup_{n\to\infty} \left(\frac{G(n)}{f(n) + g(n)}\right) \\ &\ge \limsup_{n\to\infty} \left(\frac{F(n)}{f(n)}\right) + \limsup_{n\to\infty} \left(\frac{G(n)}{g(n)}\right) < \infty\end{aligned}$$

so $F(n) + G(n) = \mathrm{O}(f(n) + g(n))$. Similarly $f(n) + g(n) = \mathrm{O}(F(n) + G(n))$ by reversing the roles, which implies $F(n) + G(n) = \Omega(f(n) + g(n))$ and thus the result.

d. Let $F(n) = \Theta(f(n))$ and $G(n) = \Theta(g(n))$. We have to show that $F(n) \cdot G(n) = \Theta(f(n) \cdot g(n))$. Indeed,

$$\limsup_{n\to\infty} \frac{F(n) \cdot G(n)}{f(n) \cdot g(n)} \le \limsup_{n\to\infty} \left(\frac{F(n)}{f(n)}\right) \cdot \limsup_{n\to\infty} \left(\frac{G(n)}{g(n)}\right) < \infty$$

as both factors are finite. This implies $F(n) \cdot G(n) = \mathrm{f(n)} \cdot \mathrm{g(n)}$. Reversing the roles, we get $f(n) \cdot g(n) = \mathrm{O}(F(n) \cdot g(n))$ which gives also $F(n) \cdot G(n) = \Omega(f(n) \cdot g(n))$, as required.

e. Let $a_1, b_1 > 0$ and let $k_1, k_2 \in \mathbb{Z}$. We have

$$\begin{aligned}(a_1 n)^{k_1} \log^{k_2}(a_2 n) &= a_1^{k_1} n^{k_1} (\log n + \log a_2)^{k_2} \\ &= a_1^{k_1} n^{k_1} \sum_{i=0}^{k_2} \binom{k_2}{i} \log^{k_2 - i} a_2 \log^i n.\end{aligned}$$

Ignoring constants and the lower powers of $\log n$, we get that the expression is $\Theta\left(n^{k_1} \log^{k_2} n\right)$.

f. We are asked to prove that

$$\sum_{k\in S}\Theta\left(f\left(k\right)\right)=\Theta\left(\sum_{k\in S}f\left(k\right)\right).$$

If $g\left(k\right)=\Theta\left(f\left(k\right)\right)$, both terms $\sum_{k\in S}g\left(k\right),\sum_{k\in S}f\left(k\right)$ are constant, so we understand both as functions of $S\subseteq\mathbb{Z}$, and study the asymptotics where S_n are strictly increasing sets. By grouping the elements of $S_n\setminus S_{n-1}$ together, we may assume that this set contains a single element. Since absolute convergence isn't affected by ordering of the terms, we may assume that

$$S_0=[n]$$

We then have to prove that

$$\sum_{k=1}^{n}g\left(k\right)=\Theta\left(\sum_{k=1}^{n}f\left(k\right)\right)$$

for all $g\left(k\right)=\Theta\left(f\left(k\right)\right)$, where the asymptotics are with respect to n, and assuming the sums converge.

Let $c_1,c_2>0$ and $n_0\in\mathbb{N}$ be such that

$$0<c_1f\left(k\right)\leq g\left(k\right)<c_2f\left(k\right).$$

We have

$$\sum_{k=1}^{n}g\left(k\right)=\sum_{k=1}^{n_0-1}g\left(k\right)+\sum_{k=n_0}^{n}g\left(k\right)$$

so

$$\sum_{k=1}^{n_0-1}g\left(k\right)+c_1\sum_{k=n_0}^{n}f\left(k\right)\leq\sum_{k=1}^{n}g\left(k\right)\leq\sum_{k=1}^{n_0-1}g\left(k\right)+c_2\sum_{k=n_0}^{n}f\left(k\right),$$

which we rewrite as

$$\sum_{k=1}^{n_0-1}\left(g\left(k\right)-f\left(k\right)\right)+c_1\sum_{k=1}^{n}f\left(k\right)\leq\sum_{|k|\leq n}g\left(k\right)\leq\sum_{k=1}^{n_0-1}\left(g\left(k\right)-f\left(k\right)\right)+c_2\sum_{k=1}^{n}f\left(k\right).$$

Since the tail of a series determines the asymptotics, we get that $\sum_{k=1}^{n}g\left(k\right)=\Theta\left(\sum_{k=1}^{n}f\left(k\right)\right)$, as functions of n.

g. Assume an analogous interpretation to that of the previous item.

Taking $f\left(k\right)=2$ and $S_n=[n]$ we get that $\prod_{k\in S_n}f\left(k\right)=2^n$. Taking $g\left(k\right)=4$, we get $\prod_{k\in S_n}g\left(k\right)=4^n$. Since $4^n\neq\Theta\left(2^n\right)$, we get that

$$\prod_{k\in S_n}\Theta\left(f\left(k\right)\right)\neq\Theta\left(\prod_{k\in S_n}f\left(k\right)\right).$$

Problem 3-6 (Variations on O and Ω)**.** a. Let $f\left(n\right),g\left(n\right)$ be two asymptotically non-negative functions and assume that $f\left(n\right)\neq\mathrm{O}\left(g\left(n\right)\right)$. Then $\limsup_{n\to\infty}\frac{f(n)}{g(n)}=\infty$, so for any value $c>0$ there are infinitely many $n\in\mathbb{N}$ such that $f\left(n\right)\geq cg\left(n\right)\geq 0$.

b. Let $f\left(n\right)=n^2\chi_{2\mathbb{Z}}+\frac{1}{n^2}\chi_{2\mathbb{Z}+1}$ and let $g\left(n\right)=n$. Then

$$\limsup_{n\to\infty}\frac{f\left(n\right)}{g\left(n\right)}=\limsup_{n\to\infty}\frac{n^2}{n}=\infty$$

so $f\left(n\right)\neq\mathrm{O}\left(g\left(n\right)\right)$, and

$$\liminf_{n\to\infty}\frac{f\left(n\right)}{g\left(n\right)}=\liminf_{n\to\infty}\frac{\frac{1}{n^2}}{n}=\liminf_{n\to\infty}\frac{1}{n}=0$$

so $f\left(n\right)\neq\Omega\left(g\left(n\right)\right)$.

c. An advantage of using $\overset{\infty}{\Omega}$ instead of Ω is that there is a sub-sequence of input size for which the running time is Ω of some function. This can be advantageous because it says in particular that the worst-case running time is no better than that of another function.

An advantage of using Ω is that it says that the running time is always as bad as a given function. It is helpful when we want to say that the running time cannot be asymptotically than some function, even for a special sub-sequence of input sizes.

d. It follows from the definitions of $\mathrm{O}, \Omega, \Theta$ that both functions in the expression $f(n) = \Omega(g(n))$ or $f(n) = \Theta(g(n))$ are asymptotically non-negative (since $0 \leq c_1 g(n) \leq f(n)$ for large enough n). Hence we wouldn't have $f(n) = \mathrm{O}'(g(n)), \Omega(g(n))$ implying $f = \Theta(g(n))$ but we'd have $f = \Theta(g(n))$ implying both $f = \mathrm{O}'(g(n)), \Omega(g(n))$.

e. We define $\tilde{\Omega}$ and $\tilde{\Theta}$ analogously.

$$\tilde{\Omega}(g(n)) = \left\{ f(n) \;\middle|\; \exists c, k, n_0 > 0 \forall n \geq n_0 : 0 \leq c g(n) \log^k(n) \leq f(n) \right\} \tilde{\Theta}(g(n)) \quad = \left\{ f(n) \;\middle|\; \exists c_1, c_2, k_1, k_2, n_0 >$$

Assume $f(n) = \tilde{\Theta}(g(n))$. Looking at only part of the inequalities in the definition of $\tilde{\Theta}$ we get that $f(n) = \tilde{\mathrm{O}}(g(n))$ and $f(n) = \tilde{\Omega}(g(n))$.

Assume that $f(n) = \tilde{\mathrm{O}}(g(n))$ and $f(n) = \tilde{\Omega}(g(n))$. There are $c_1, c_2, k_1, k_2, n_1, n_2$ such that

$$\forall n \geq n_1 : 0 \leq c_1 g(n) \log^{k_1}(n) \leq f(n)$$
$$\forall n \geq n_2 : 0 \leq f(n) \leq c_2 g(n) \log^{k_2}(n).$$

Taking $n_0 = \max(n_1, n_2)$, we get

$$\forall n \geq n_0 : 0 \leq c_1 g(n) \log^{k_1}(n) \leq f(n) \leq c_2 g(n) \log^{k_2}(n), \tag{3.1}$$

so that $f(n) = \tilde{\Theta}(g(n))$.

Problem 3-7 (Iterated Functions).

- For $f(n) = \sqrt{n}$, we have $f^{(2)}(n) = \left(n^{\frac{1}{2}}\right)^{\frac{1}{2}} = n^{\frac{1}{4}}$, and similarly $f^{(i)}(n) = n^{\left(\frac{1}{2}\right)^i}$. Hence $f^{(i)}(n) \leq c$ if and only if $n^{\left(\frac{1}{2}\right)^i} \leq c$ if and only if $\left(\frac{1}{2}\right)^i \leq \log_n c$ if and only if $2^i \geq \frac{1}{\log_n c} = \log_c n$, if and only if $i \geq \log_2 \log_c n$. For $c = 2$ this means $f_c^*(n) = \lceil (\log \log n) \rceil$, and for $c = 1$ this is undefined unless $n = 1$ in which case the solution is trivial.
- For $f(n) = n^{1/3}$ we similarly get $f^{(i)}(n) \leq c$ if and only if $3^i \geq \log_c n$, so $i \geq \log_3 \log_c n$.
- For $f(n) = n/\log n$, we have $f(n) \leq n/2$ so $f_c^*(n) \leq \lceil \log_2(n) \rceil - 1$, and $f(n) \geq \sqrt{n}$ so $f_c^*(n) \geq \lceil \log \log n \rceil$. We get that $f_c^* = \mathrm{O}(\log n)$ and $f_c^* = \Omega(\log \log n)$.

4 Divide-and-Conquer

4.1 Multiplying square matrices

Exercise 4.1-1. We generalize `Matrix-Multiply-Recursive` as follows, using the existing algorithm and the fact that

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

```
1    def general_matrix_multiply_recursive(A,B,C,n):
2        let A', B', C' be block-diagonal matrices of size 2^m where 2^(m-1) < n <= 2^m and
             where the first block is A, B or C respectively, and the rest is zero.
3
4        Matrix_Multiply_Recursive(A', B', C', 2^m)
5        C = C'[:n, :n]
```

A recursion for the algorithm is

$$T(n) = \begin{cases} T(2^m) + (2^m)^2 + 3n^2 & \exists m : 2^{m-1} < n < 2^m \\ 8T(n/2) & n \text{ is a power of 2} \end{cases}$$

In case where n is a power of 2, we get $T(n) = \Theta\left(n^3\right)$ because this is the computation time of `Matrix-Multiply-Recursive`. In the case where n is not a power of 2, we have $(2^m)^2 + 3n^2$ computations in order to create the large matrices and copy the data from A, B and then copy back the data to C. Since $2^m < 2n$, this is at most $(2n)^2 + 3n^2 = 7n^2$ computations. Since $\Theta\left(n^3\right) + 7n^2 = \Theta\left(n^3\right)$, the total time complexity is $\Theta\left(n^3\right)$.

Exercise 4.1-2.

- To multiply a $kn \times n$ matrix $A = \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_k \end{pmatrix}$ by an $n \times kn$ matrix $B = \begin{pmatrix} B_1 & B_2 & \cdots & B_k \end{pmatrix}$ using `Matrix-Multiply-Recursive` as a subroutine, we have to compute $\sum_{i \in [k]} A_i B_i$, which uses `Matrix-Multiply-Recursive` k times. This would have $\Theta\left(kn^3\right)$ time complexity.
- To multiply BA instead, we have to compute the block matrix

  $$\left(A_{i,j} B_{i,j}\right)_{i,j \in [k]}$$

  which uses `Matrix-Multiply-Recursive` k^2 times and therefore has $\Theta\left(k^2 n^3\right)$ time complexity, which is larger by a factor of k.

Exercise 4.1-3. The recursive formula (4.9) changes by adding the computation time for copying the 12 matrices of size $n/2 \times n/2$, which takes $3n^2$ computations. We thus get the equation

$$T(n) = 8T(n/2) + 3n^2.$$

This adds a total computation time of

$$3\left(n^2 + \left(\frac{n}{2}\right)^2 + \left(\frac{n}{4}\right)^2 + \ldots\right) = 3n^2 \sum_{i=1}^{\log n} \left(\frac{1}{i}\right)^2 = \Theta\left(n^2\right),$$

so the solution for the recursion is still $\Theta\left(n^3\right)$.

Exercise 4.1-4. We write a program to recursively sum two matrices by partitioning them.

```
1         def matrix_add_recursive(A,B,C,n):
2             // Base case.
3             if(n == 0):
4                 c[1,1] = a[1,1] + b[1,1]
5                 return
6
7             // Divide.
8             partition A,B,C into matrices X_{1,1}, X_{1,2}, X_{2,1}, X_{2,2} of size (n/2 x n
                  /2), where X is either A,B or C
9
10            // Conquer
11
12            matrix_add_recursive(A_{1,1}, B_{1,1}, C_{1,1})
13            matrix_add_recursive(A_{1,2}, B_{1,2}, C_{1,2})
14            matrix_add_recursive(A_{2,1}, B_{2,1}, C_{2,1})
15            matrix_add_recursive(A_{2,2}, B_{2,2}, C_{2,2})
```

A recurrence for the worst-case running time would be

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

If it instead takes $\Theta\left(n^2\right)$-time to implement the partitioning instead of index calculation, the recursion is instead

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

4.2 Strassen's algorithm for matrix multiplication

Exercise 4.2-1. Let $A = \begin{pmatrix} 1 & 3 \\ 7 & 5 \end{pmatrix}$ and $B = \begin{pmatrix} 6 & 8 \\ 4 & 2 \end{pmatrix}$. Using Strassen's algorithm, we define

$$\begin{aligned}
s_1 &= b_{1,2} - b_{2,2} = 8 - 2 = 6 \\
s_2 &= a_{1,1} + a_{1,2} = 1 + 3 = 4 \\
s_3 &= a_{2,1} + a_{2,2} = 7 + 5 = 12 \\
s_4 &= b_{2,1} - b_{1,1} = 4 - 6 = -2 \\
s_5 &= a_{1,1} + a_{2,2} = 1 + 5 = 6 \\
s_6 &= b_{1,1} + b_{2,2} = 6 + 2 = 8 \\
s_7 &= a_{1,2} - a_{2,2} = 3 - 5 = -2 \\
s_8 &= b_{2,1} + b_{2,2} = 4 + 2 = 6 \\
s_9 &= a_{1,1} - a_{2,1} = 1 - 7 = -6 \\
s_{10} &= b_{1,1} + b_{1,2} = 6 + 8 = 14.
\end{aligned}$$

We then compute

$$\begin{aligned}
p_1 &= a_{1,1} \cdot s_1 = 1 \cdot 6 = 6 \\
p_2 &= s_2 \cdot b_{2,2} = 4 \cdot 2 = 8 \\
p_3 &= s_3 \cdot b_{1,1} = 12 \cdot 6 = 72 \\
p_4 &= a_{2,2} \cdot s_4 = 5 \cdot (-2) = -10 \\
p_5 &= s_5 \cdot s_6 = 6 \cdot 8 = 48 \\
p_6 &= s_7 \cdot s_8 = -2 \cdot 6 = -12 \\
p_7 &= s_9 \cdot s_{10} = -6 \cdot 14 = -84.
\end{aligned}$$

Strassen's algorithm says that $C = A \cdot B$ has the following coefficients

$$\begin{aligned} c_{1,1} &= p_5 + p_4 - p_2 + p_6 = 48 - 10 - 8 - 12 = 18 \\ c_{1,2} &= p_1 + p_2 = 6 + 8 = 14 \\ c_{2,1} &= p_3 + p_4 = 72 - 10 = 62 \\ c_{2,2} &= p_5 + p_1 - p_3 - p_7 = 48 + 6 - 72 + 84 = 66. \end{aligned}$$

Hence

$$A \cdot B = \begin{pmatrix} 18 & 14 \\ 62 & 66 \end{pmatrix}.$$

Exercise 4.2-2.

Exercise 4.2-3.

Exercise 4.2-4.

Exercise 4.2-5. Compute the following

$$\begin{aligned} m_1 &= ac \\ m_2 &= bd \\ m_3 &= (a+b)(c+d) = ac + bc + ad + bd. \end{aligned}$$

Then

$$\begin{aligned} ac - bd &= m_1 - m_2 \\ ad + bc &= m_3 - m_1 - m_2. \end{aligned}$$

Exercise 4.2-6. Let $C = \begin{pmatrix} A & B \\ -B & A \end{pmatrix}$. Then

$$C^2 = \begin{pmatrix} A^2 - B^2 & 2AB \\ -2BA & A^2 - B^2 \end{pmatrix}.$$

Dividing the submatrices $2AB, -2BA$ by $2, -2$ respectively, we get the products AB, BA. The computation time of C^2 is $(2n)^\alpha = \Theta(n^\alpha)$.

4.3 The substitution method for solving recurrences

Exercise 4.3-1. a. Let $T(n) = T(n-1) + n$. We search for constants $c > 0$ and $n_0 \in \mathbb{N}$ such that $T(n) \leq cn^2$ for all $n \geq n_0$. Assume that $T(n) \leq cn^2$ for all $n < m$. Then

$$T(m) = T(m-1) + m \leq c(m-1)^2 + m = cm^2 + (1-2c)m + 1.$$

Assuming $(1-2c)m + 1 < 0$, we get $T(m) \leq cm^2$. This condition is equivalent to $c > 1/2m + 1/2$ and therefore holds for all $m \geq 1$ and $c > 1$. Hence, taking $n_0 = 1$ and $c > 1$ large enough so that $T(n) \leq cn^2$ for $n = n_0$, we get that $T(n) \leq cn^2$, and so $T(n) = O(n^2)$.

b. Let $T(n) = T(n/2) + \Theta(1)$. We search for constants $c > 0$ and $n_0 \in \mathbb{N}$ such that $T(n) \leq c \log n$ for all $n \geq n_0$. Assume that $T(n) \leq c \log n$ for all $n < m$. Then

$$\begin{aligned} T(m) &= T(m/2) + \Theta(1) \\ &\leq c \log(m/2) + \Theta(1) \\ &= c \log m - c + \Theta(1) \\ &< c \log m \end{aligned}$$

where in the first inequality we assume $m/2 \geq n_0$ and in the last inequality we assume that c is larger than the constant in $\Theta(1)$. Taking $n_0 = 2$ guarantees that $\log n > 0$ for all $n \geq n_0$. We assumed that $n \geq 2n_0 = 4$, so taking $c > 0$ large enough so that $T(n) \leq c \log n$ for $n \in \{2, 3\}$ gets us the result.

c. Let $T(n) = 2T(n/2) + n$. We search for $c > 0$ and $n_0 \in \mathbb{N}$ such that $T(n) \leq cn \log n$ for all $n \geq n_0$. Assume that $T(n) \leq cn \log n$ for all $n_0 \leq n < m$. Then, if $m \geq 2n_0$,

$$\begin{aligned} T(m) &= 2T(m/2) + m \\ &\leq 2c\frac{m}{2}\log\left(\frac{m}{2}\right) + m \\ &= cm\log m - cm + m. \end{aligned}$$

Taking $c \geq 1$ and $m \geq 1$, we get $T(m) < cm \log m$. We take $n_0 = 2$ so that $n \log n > 0$, so are requirement that $m \geq 2n_0$ means that $m \geq 4$. We take c large enough so that $T(n) \leq cn \log n$ for $n \in \{3, 4\}$, which satisfies the base case.

d. Let $T(n) = 2T(n/2 + 17) + n$. We search for $c, d > 0$ and $n_0 \in \mathbb{N}$ such that $T(n) \leq cn \log n - d \log n$ for all $n \geq n_0$. Assume that $T(n) \leq cn \log n - d \log n$ for all $n_0 \leq n < m$. Then, if $m \geq 2n_0 - 17$,

$$\begin{aligned} T(m) &= 2T(m/2 + 17) + m \\ &\leq 2c\left(\frac{m}{2} + 17\right)\log\left(\frac{m}{2} + 17\right) - 2d\log m + m \\ &= (cm + 34)\log\left(\frac{m}{2} + 17\right) - 2d\log m + m \\ &= cm\log\left(\frac{m}{2} + 17\right) + 34\log\left(\frac{m}{2} + 17\right) - 2d\log m + m \\ &\leq cm\log\left(\frac{m}{2} + 17\right) + m, \end{aligned}$$

where in the last inequality we assumed the $m \geq 34$ and $d \geq 17$. Taking $m \geq 72$, we get $\frac{m}{2} + 17 \leq \frac{3m}{4}$, for which

$$\begin{aligned} T(m) &\leq cm\log\left(\frac{3m}{4}\right) + m \\ &= cm\log(m) + \left(c\log\left(\frac{3}{4}\right) + 1\right)m. \end{aligned}$$

Fixing $d = 17$, since $\log\left(\frac{3}{4}\right)$ is negative, we can take c, m large enough such that $\left(c\log\left(\frac{3}{4}\right) + 1\right)m < -2d\log m$, so that

$$T(m) \leq cm\log m - 2d\log m.$$

Taking c to also be large enough for the inequality $T(n) \leq cn \log n - d \log n$ for values $n_0 \leq n < 2n_0 - 17$, we get the basis case.

e. Let $T(n) = 2T(n/3) + \Theta(n)$. This has a solution if there's $f(n) = \Theta(n)$ such that $T(n) = 2T(n/3) + f(n)$ has a solution. Choose $f(n) = \alpha n = \Theta(n)$ for some $\alpha > 0$. We search for $c > 0$ and $n_0 \in \mathbb{N}$ such that $T(n) \leq cn$ for all $n \geq n_0$. Assume that $T(n) \leq cn$ for all $n_0 \leq n < m$. Then, if $m \geq 3n_0$,

$$\begin{aligned} T(m) &= 2T(m/3) + \alpha m \\ &\leq 2\frac{cm}{3} + \alpha m \\ &= \frac{2cm + 3\alpha m}{3} \\ &= \frac{(2c + 3\alpha)m}{3}. \end{aligned}$$

Solving $2c + 3\alpha < 3c$ we get $c > 3\alpha$. Picking $\alpha = 1$ and $c > 3$ we get that $T(m) \leq \frac{3cm}{3} = cm$. We take $n_0 = 1$ so that n is positive for all $n \geq n_0$. Then $m \geq 3n_0$ means $m \geq 3$. We take c to be big enough such that $T(n) \leq cn$ for $n \in \{1, 2\}$, which gives the basis case.

f. Let $T(n) = 4T(n/2) + \Theta(n)$. This has a solution if there's $f(n) = \Theta(n)$ such that $T(n) = 4T(n/2) + f(n)$ has a solution. Choose $f(n) = \alpha n = \Theta(n)$ for some $\alpha > 0$. We search for $c > 0$ and $n_0 \in \mathbb{N}$ such

that $T(n) \leq cn^2$ for all $n \geq n_0$. Assume that $T(n) \leq cn^2$ for all $n_0 \leq n < m$. Then, if $m \geq 2n_0$,

$$\begin{aligned} T(m) &= 4T(m/2) + \alpha m \\ &\leq 4\frac{cm^2}{4} + \alpha m \\ &= cm^2 + \alpha m. \end{aligned}$$

The inequality $cm^2 + \alpha m < cm^2$ has no solutions, so we tighten our assumption: Assume instead that $T(n) \leq cn^2 - \beta\alpha m$ for all $n \leq m$ and for some $\beta > 0$. For $m \geq 2n_0$.

$$\begin{aligned} T(m) &= 4T(m/2) + \alpha m \\ &\leq 4\left(\frac{cm^2}{4} - \beta\alpha\frac{m}{2}\right) + \alpha m \\ &= cm^2 - 2\beta\alpha + \alpha m. \end{aligned}$$

Solving $-2\beta\alpha m + \alpha m \leq -\beta\alpha m$ we get $\alpha m \leq \beta\alpha m$ which is solved by $\beta \geq 1$. Hence, take $\beta = 1$ such that the assumption is $T(n) \leq cn^2 - \alpha n$ for $n < m$, and such that $T(m) \leq cm^2 - \alpha m$. We take $n_0 = 1$ soc that n is positive for $n \geq n_0$. Then $m \geq 2n_0$ means $m \geq 2$. We take c to be big enough such that $T(n) \leq cn^2$ for $n = 1$, which gives the basis case.

Exercise 4.3-2. This is done as part of Item f. of Exercise 4.3-1.

Exercise 4.3-3. Let $T(n) = 2T(n-1) + 1$.

- We show that a substitution proof fails with the assumption $T(n) \leq c2^n$ where $c > 0$ is constant. Assume that $T(n) \leq c2^n$ for $n \leq m$. The induction hypothesis gives

  $$T(m) = 2T(m-1) + 1 \leq 2c2^{m-1} + 1 = c2^m + 1.$$

  So, we could have $T(m) \in (c2^m, c2^m + 1]$, which contradicts the induction statement for m.

- Assume instead that $T(n) \leq c2^n - f(n)$ for some function f and for all $n \leq m$. We have

  $$\begin{aligned} T(m) &= 2T(m-1) + 1 \\ &\leq 2c2^{m-1} - f(m-1) + 1 \\ &= c2^m - f(m-1) + 1. \end{aligned}$$

  In order to get $T(m) \leq c2^m - f(m)$, we need $-f(m-1) + 1 \leq -f(m)$, i.e. $f(m) \leq f(m-1) - 1$. In particular, f has to be asymptotically negative, so "substracting a lower-order term" isn't the best phrasing for the exercise. Choosing $f(n) = -2^{n-1}$, we have $f(m) = -2^{m-1} \leq -2^{m-1} - 1$ for all m. Taking $n_0 = 0$ and c large enough such that $T(0) \leq c$, we get the inequality. Taking $n_0 = 1$ and c large enough so that $T(1) \leq 2c$, we get that $T(n) = \Theta(2^n)$.

4.4 The recursion-tree method for solving recurrences

Exercise 4.4-1. a. • We have the following unary tree.

$$
\begin{array}{c}
n^3 \\
| \\
\left(\frac{n}{2}\right)^3 \\
| \\
\left(\frac{n}{4}\right)^3 \\
| \\
\left(\frac{n}{8}\right)^3 \\
| \\
\vdots \\
| \\
2^3 \\
| \\
1
\end{array}
$$

where summing the nodes gives

$$
\begin{aligned}
T(n) &= \sum_{i}^{\log_2 n} \left(2^i\right)^3 \\
&= \sum_{i=0}^{\log_2 n} 8^i
\end{aligned}
$$

which, according to the formula for the geometric sum, gives

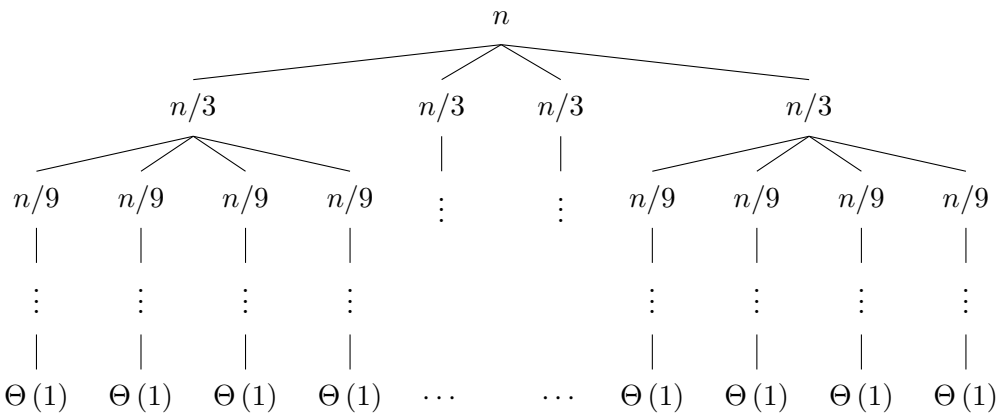
$$
T(n) = \frac{1 - 8^{\log_2 n}}{1 - 8} = \frac{n^3 - 1}{7} = \Theta\left(n^3\right).
$$

- Assume that $T(n) \leq cn^3$ for all $n < m$, we show that $T(m) \leq cm^3$ for a right choice of $c > 0$ (independent of m). Indeed,

$$
\begin{aligned}
T(m) &= T(m/2) + m^3 \\
&\leq c\frac{m^3}{8} + m^3 \\
&= m^3\left(\frac{c}{8} + 1\right).
\end{aligned}
$$

This is less than cm^3 if $\frac{c}{8} + 1 < c$, which is the case whenever $c > \frac{8}{7}$. Choosing $n_0 = 2$, such that $n/2 > 0$, and c large enough such that $c > \frac{8}{7}$ and also $T(n) = \mathrm{O}(1)$ for $n \leq n_0$, we get that $T(n) = \mathrm{O}\left(n^3\right)$.

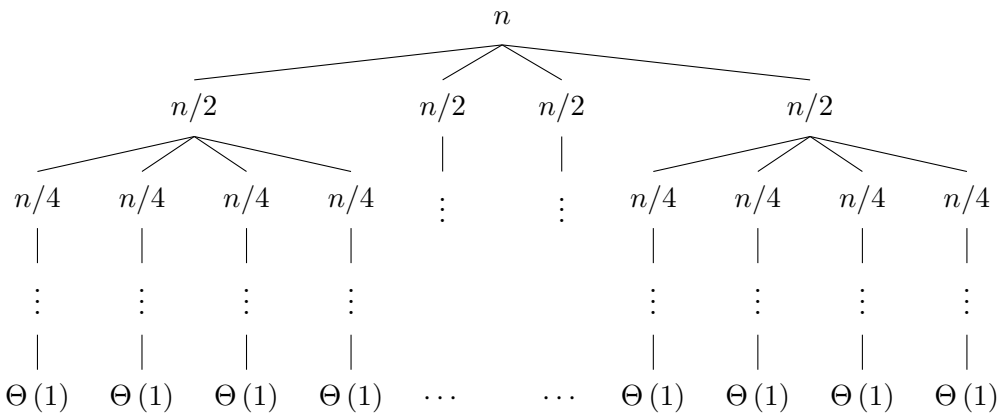
b. • We have the following quadratic tree.



Since at each depth we divide the previous value by 3, the tree has depth $\log_3 n$. At the i^{th} level there are 4^i nodes, and the sum of the expressions is $4^i \cdot \frac{n}{3^i} = \left(\frac{4}{3}\right)^i n$. Denoting $k \coloneqq \log_3 n - 1$, the sum of the inner nodes is therefore

$$\begin{aligned}\sum_{i=0}^{k} \left(\frac{4}{3}\right)^i n &= \frac{\left(\frac{4}{3}\right)^{k+1} - 1}{\frac{4}{3} - 1} n \\ &= 3n \cdot \left(\left(\frac{4}{3}\right)^{\log_3 n} - 1\right) \\ &= 3n \cdot \left(n^{\log_3 \frac{4}{3}} - 1\right).\end{aligned}$$

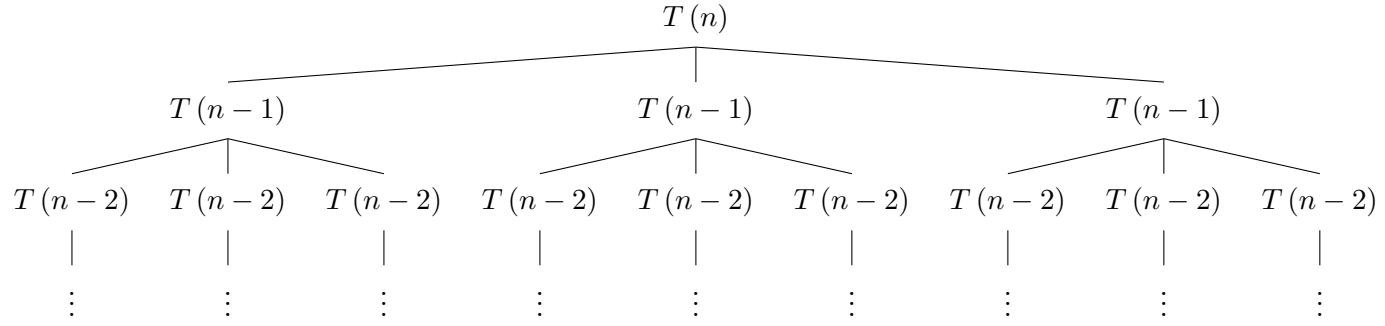
c. We have the following quadratic tree.



Since at each depth we divide the previous value by 2, the tree has depth $\log_2 n$. At the i^{th} level there are 4^i nodes, and the sum of the expressions is $4^i \cdot \frac{n}{2^i} = 2^i n$. Denoting $k \coloneqq \log_2 n - 1$, the sum of the inner nodes is therefore

$$\sum_{i=0}^{k} 2^i n = \frac{2^{k+1} - 1}{2 - 1} \cdot n = \left(2^{\log_2 n} - 1\right) n = (n-1)\, n = n^2 - n = \Theta\left(n^2\right).$$

d. We have the following ternary tree.



Since the value given to T is decreased by 1 at each level, the tree has depth $n - 1$. The number of nodes in the tree is then

$$\sum_{i=0}^{n-1} 3^i = \frac{3^n - 1}{3 - 1} = 2 \cdot 3^n - 2 = \Theta\left(3^n\right).$$

Exercise 4.4-2. Let

$$L(n) = \begin{cases} 1 & n < n_0 \\ L(n/3) + L(2n/3) & n \geq n_0. \end{cases}$$

Let $m > n_0$ and assume that $L(n) \geq cn$ for all $n < m$, we show that $L(n) \geq cm$ for suitable $c > 0$. Indeed,

$$\begin{aligned} L(m) &= L(m/3) + L(2m/3) \\ &\geq \frac{cm}{3} + \frac{2cm}{3} \\ &= cm. \end{aligned}$$

We choose c small enough such that $L(n) \geq cn$ for all $n \leq n_0$, thus showing that $L(b) \geq cb$ for all $n \in \mathbb{N}$, which implies $L(n) = \Omega(n)$. We've seen in the book that also $L(n) = \mathrm{O}(n)$, hence $L(n) = \Theta(n)$.

Exercise 4.4-3. Let

$$T(n) = T(n/3) + T(2n/3) + \Theta(n).$$

Let $\alpha n = \Theta(n)$ and let $m > n_0$. Assume that $T(n) \geq cn \log n + f(n)$ for $n < m$. We have

$$\begin{aligned} T(m) &= T(m/3) + T(2m/3) + \alpha m \\ &\geq c\left(\frac{m}{3}\log\left(\frac{m}{3}\right) + \frac{2m}{3}\log\left(\frac{2m}{3}\right)\right) + (\alpha + 2c)\, m \\ &= c\left(\frac{m}{3}\log\left(\frac{m}{3}\right) + \frac{2m}{3}\left(1 + \log\left(\frac{m}{3}\right)\right)\right) + (\alpha + 2c)\, m \\ &= c\left(\frac{m}{3}\log\left(\frac{m}{3}\right) + \frac{2m}{3} + \frac{2m}{3}\log\left(\frac{m}{3}\right)\right) + (\alpha + 2c)\, m \\ &= cm\log\left(\frac{m}{3}\right) + \frac{2m}{3} + (\alpha + 2c)\, m \\ &= cm\log(m) - cm\log(3) + \left(\alpha + 2c + \frac{2}{3}\right) m \\ &= cm\log(m) + \left(\alpha + 2c + \frac{2}{3} - c\log(3)\right) m. \end{aligned}$$

Bibliography

[Cor+22] T.H. Cormen et al. *Introduction to Algorithms, fourth edition*. MIT Press, 2022. ISBN: 9780262046305.