Suggested practice exercises for DA4006

Compiled by Anna Lindeberg % Patricia Ebert

April 16, 2026

The following PDF contains a collection of suggested practice exercises for the course Datastructures and algorithms. It will be updated throughout the course — suggestions and comments are welcomed.

Contents

1 Fundamentals **2**
1.1 Correctness of algorithms and runtime . . . 2
1.2 Big-O and related notations . . . 3
1.3 Elementary data structures . . . 5
1.4 Graphs and trees . . . 5
1.5 Miscellaneous . . . 7

2 Sorting **8**
2.1 Merge-Sort . . . 8
2.2 Heap-Sort . . . 8
2.3 Quick-Sort . . . 9
2.4 Counting sort . . . 9
2.5 Applications of sorting . . . 9

3 Searching **10**
3.1 . . . in arrays . . . 10
3.2 . . . with search trees . . . 11

Section 1 Fundamentals

1.1 Correctness of algorithms and runtime

Problem 1.1 [1]
Adapt `Insertion-Sort` so that it sorts non-increasing instead of non-decreasing (i.e. sort large to small). Adapt the proof of correctness from the lecture so that it works for your new psuedocode.

Problem 1.2
Adapt `Insertion-Sort` so that it returns a *copy* of the input array. Adapt the proof of correctness from the lecture so that it works for your new psuedocode.

Problem 1.3
Implement `Insertion-Sort` in some programming language of your choice. Compare the running time of your implementation on random arrays of length n for $n = 10^3, 10^4, 10^5, 10^6$. How does your implementation perform in comparison to the built-in sorting function of your programming language?

Problem 1.4
Your younger sibling has messed up your sorted list! You need to restore the sorted order. Due to limited time for this mischievous act, you know that your sibling has only managed to swap two elements in the list. Provide an algorithm that restores the sorted order of the list in $O(n)$ time.

Problem 1.5
Alas, your sibling has messed up your sorted list again! This time, you know that your sibling has only managed to place elements at most k positions away from their correct position in the sorted order. What is the runtime of `Insertion-Sort` on such a list? **Hint:** start by proving that the inner loop of `Insertion-Sort` runs in $O(k)$ time on such a list.

Problem 1.6 [1]
Consider the *searching problem*:

> **Input:** A sequence of n numbers $A = \langle a_1, a_2, \ldots, a_n \rangle$ and a value v.
>
> **Output:** An index i such that $a_i = v$, or a special value `NIL` if v does not appear in A.

Write pseudocode for `linear search`, which scans through the sequence A looking for v. Using a loop invariant, prove that your algorithm is correct.

Problem 1.7
For each of the following functions $f(n)$, provide an algorithm that runs in time $\Theta(f(n))$. (The input is not important for this problem, you can assume it is a single integer n).

(a) $f(n) = n$	(b) $f(n) = n^2$	(c) $f(n) = n \log n$
(d) $f(n) = 2^n$	(e) $f(n) = n!$	(f) $f(n) = 2^{(2^n)}$

When looking at an algorithm, before trying to prove that it is correct, it can be useful to try to find a counterexample. If you do find a counterexample, then you know that the algorithm is not correct and you can try to fix it. If you cannot find a counterexample, then you have some evidence that the algorithm might be correct, and you can try to prove it. In the following problems, we will practice finding counterexamples.

Problem 1.8 Adapted from [3]

(a) Show that $a + b$ can be less than $\min(a, b)$.

(b) Show that $a \times b$ can be less than $\min(a, b)$.

(c) Can you characterize all pairs of real numbers a and b for which $a + b < \min(a, b)$? Can you characterize all pairs of real numbers a and b for which $a \times b < \min(a, b)$? That is, $a + b < \min(a, b)$ if and only if ....

Problem 1.9 Adapted from [3]
The *Knapsack problem* is as follows: given a set of integers $S = \{s_1, s_2, \ldots, s_n\}$, and a target number T, find a subset of S which adds up exactly to T. For example, there exists a subset within $S = \{1, 2, 5, 9, 10\}$ that adds up to $T = 22$ but not $T = 23$. For each of the following descriptions of algorithms, write pseudocode that formalize the approach. Then, find a counterexample for its correctness. That is, give an S and T where the algorithm does not find a solution which leaves the knapsack completely full, even though a full-knapsack solution exists.

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

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

(c) Put the elements of S in the knapsack from largest to smallest.

The Knapsack problem is of a class of especially "difficult" problems. These are further investigated in the course Algorithms and Complexity (DA4005).

1.2 Big-O and related notations

Problem 1.10
In the following, explicitly use the definition of $O(n^3)$.

(a) Prove that $(n-1)^3 + 2n \in O(n^3)$. Are the constants c and n_0 you provide in the proof unique?

(b) Prove that $(n-1)^3 + 2n \notin O(n^2 \log n)$.

Problem 1.11
Show the following statement mentioned in the lecture.

(a) $f(n) \in \Theta(g(n))$ if and only if $g(n) \in \Theta(f(n))$

(b) $f(n) \in O(g(n))$ if and only if $g(n) \in \Omega(f(n))$

Problem 1.12
Show the following statement mentioned in the lecture, for each $\Upsilon \in \{O, \Omega, \Theta\}$.

(a) $c \cdot \Upsilon(f(n)) = \Upsilon(c \cdot f(n))$

(b) $\Upsilon(f(n)) \cdot \Upsilon(g(n)) = \Upsilon(f(n) \cdot g(n))$

Problem 1.13
For each of the functions f in Problem 1.7, define a function $g(n)$ such that $g(n) \in O(f(n))$ but $g(n) \notin \Omega(f(n))$ and a function $h(n)$ such that $h(n) \in \Omega(f(n))$ but $h(n) \notin O(f(n))$.

Problem 1.14
Let $f : \mathbb{N} \to \mathbb{R}$ be a function such that $f(n) \geq 1$ for all n. Prove that $g(n) \in O(f(n))$ if and only if $g(n) \in O(\lfloor f(n) \rfloor)$. Is the same statement true if we replace $\lfloor f(n) \rfloor$ with $\lceil f(n) \rceil$? Is the same statement true if we allow $f(n)$ to take values smaller than 1?

Problem 1.15
Let b be a positive real number. Show that $\lfloor \log_b(n) \rfloor + 1 = \lceil \log_b(n+1) \rceil$ for all integers $n \geq 1$. Is the equality true for all *real* numbers $n \geq 1$?

Problem 1.16
Using the substitution method, prove that $T(n) \in O(f(n))$, for the following T and f:

(a)

$$f(n) = n\log(n) \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n & \text{if } n > 1 \end{cases}$$

(b)

$$f(n) = n^2 \qquad T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n-1) + n & \text{if } n > 1 \end{cases}$$

Problem 1.17
Consider the function $T(n) = 2T(\sqrt{n}) + \log n$ with $T(k) = 1$ for $k \leq 2$. Find and rigorously prove an asymptotically tight bound for $T(n)$, with the substitution method. Can the master theorem be applied to T? Why or why not?

Problem 1.18
For the three T below, all where $T(1) = 1$, determine $f(n)$ such that $T(n) \in \Theta(f(n))$ using the (simplified) master theorem.

(a) $T(n) = 2T(n/2) + n$

(b) $T(n) = 3T(n/3) + \sqrt{n}$

(c) $T(n) = 2T(n/2) + n^3 + 12n^2 - 7$

Problem 1.19 [2]
For each of the following recurrences, give an expression for the runtime $T(n)$ if the recurrence can be solved with the (simplified) Master Theorem. Otherwise, indicate that it does not apply.

(a) $T(n) = 3T(n/2) + n^2$
(b) $T(n) = 4T(n/2) + n^2$
(c) $T(n) = T(n/2) + n^2$
(d) $T(n) = 2^nT(n/2) + n^n$
(e) $T(n) = 16T(n/4) + n$
(f) $T(n) = 2T(n/2) + n\log n$
(g) $T(n) = 2T(n/2) + n/\log n$
(h) $T(n) = 2T(n/4) + n^{0.51}$
(i) $T(n) = 0.5T(n/2) + 1/n$
(j) $T(n) = 6T(n/3) + n^2\log n$
(k) $T(n) = 64T(n/8) - n^2\log n$
(l) $T(n) = 7T(n/3) + n^2$
(m) $T(n) = 4T(n/2) + \log n$
(n) $T(n) = 16T(n/4) + n!$
(o) $T(n) = T(n/2) + \log n$
(p) $T(n) = 3T(n/2) + n$
(q) $T(n) = 3T(n/3)$
(r) $T(n) = 4T(n/2) + cn$
(s) $T(n) = 3T(n/4) + n\log n$
(t) $T(n) = 3T(n/3) + n/2$

(After this you can probably apply the Master Theorem in your sleep.)

Problem 1.20 Adapted from [2]
Prove that if $T(n) = T(\alpha n) + T((1-\alpha)n) + \beta n$ for some constants $0 < \alpha < 1$ and $\beta > 0$ then $T(n) \in O(n\log n)$.

Problem 1.21
Consider the function `fun`(n) defined as follows:

```
1: j ← 1
2: while j^2 ≤ n do
3:     k ← j
4:     while k > 1 do
5:         k ← k/100
6:     j ← j + 1
7: return j
```

Show that the runtime of `fun`(n) is $O(\sqrt{n}\log(n))$.

1.3 Elementary data structures

Problem 1.22
Draw a sketch, like in the slides, of the pointers and values of a single linked list L containing the values $4, 7, -2, 13$ and 34. Then repeat for a doubly linked list with the same values.

Problem 1.23
Let Q be a queue, initially empty. What elements does Q have, and in what order, after the following operations?
Q.enqueue(4); Q.enqueue(7); Q.enqueue(8); Q.dequeue(); Q.front(); Q.enqueue(1); Q.enqueue(2); Q.dequeue(); Q.dequeue();

Problem 1.24
Let S be a queue, initially empty. What elements does S have, and in what order, after the following operations?
S.push(4); S.push(7); S.push(8); S.pop(); S.top(); S.push(1); S.push(2); S.pop(); S.pop();

Problem 1.25
Think about queues and stacks. What are the advantages and disadvantages of implementing these two types of data structures using a contiguously-allocated array respectively some type of linked list? Sketch an outline of how this could be done.

Problem 1.26
Suppose your programming language only has access to the stack data structure. How can you implement a queue? Conversely, how can you implement a stack with only access to the queue data structure?

Problem 1.27
Do some research on how memory allocation works in your favorite programming language. What type of lists and/or arrays exist? How does one access queues and stacks?

1.4 Graphs and trees

Problem 1.28

(a) How many vertices can a rooted tree with ℓ leaves have?

(b) How many vertices can a binary rooted tree with ℓ leaves have?

(c) How many vertices can a complete rooted tree with ℓ leaves have? Is it possible to answer this question for all ℓ?

(d) How many vertices can a nearly-complete rooted tree with ℓ leaves have?

(e) Exchange the words "vertices" and "leaves" in the questions above and then answer them.

Many of the answers will be a range of integers.

Problem 1.29
Let $G = (V, E)$ be a graph. Show that the following statements are equivalent (a classic in graph theory)

(i) G is a tree (per definition: acyclic and connected)

(ii) For all $x, y \in V$, there is precisely one (simple) xy-path in G

(iii) G is connected and $|E| = |V| - 1$

(iv) G is acyclic and $|E| = |V| - 1$

Problem 1.30
For a graph $G = (V, E)$, let $\deg_G(v)$ denote the number of edges in G that contains $v \in V$.

(a) Prove the classical handshake lemma: $\sum_{v \in V} \deg_G(v) = 2|E|$ for all graphs $G = (V, E)$.

(b) In the lecture we defined leaves in a *rooted* tree. Alternatively, a vertex v is a leaf in *arbitrary* tree $T = (V, E)$ with $|V| > 1$ vertices if only if $\deg_T(v) = 1$. Based on this definition, prove that every tree with at least two vertices has at least two leaves.

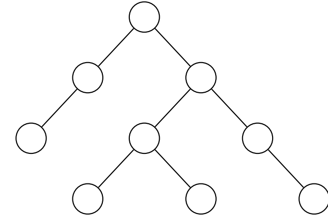
Problem 1.31
Sketch a tree with n vertices and ...

(a) ...the minimum possible number of leaves

(b) ...the maximum possible number of leaves.

Problem 1.32

Assign labels to the following tree so that reading the labels ...



(a) ...in pre-order ...

(b) ...in post-order ...

(c) ...in in-order ...

...yields the string MATEMATIK.

Problem 1.33
Let $T = (V, E)$ be a rooted tree. Let $h(v)$ resp. $d(v)$ denote the *height* respectively *depth* of a vertex $v \in V$. Recall that $h(T) := h(r)$, where r is the root of T. Show that the following statements are equivalent

(i) $h(v) + d(v) = h(T)$ for all $v \in V$

(ii) $d(x) = d(y)$ for all leaves x, y of T

(iii) $d(x) = h(T)$ for all leaves x of T

Use this to prove that T is a complete rooted tree if and only if T is a fully binary rooted tree and $h(v)+d(v) = h(T)$ for all $v \in V(T)$.

Want something a little bit more complicated? Characterize nearly-complete trees in a similar manner.

Problem 1.34
Write an algorithm and prove its correctness for the following two problems.

(a) `Depth`(T, v)
Input: A rooted tree T, a vertex $v \in V(T)$
Output: The integer $d(v)$ (the depth of v in T)

(b) `Height`(T, v)
Input: A rooted tree T, a vertex $v \in V(T)$
Output: The integer $h(v)$ (the height of v in T)

Hint: use recursion. You may assume that T`.root` stores the root of T, and that every vertex v can access its parent in v`.parent` resp. a list of its children in v`.children`.

1.5 Miscellaneous

Problem 1.35
Consider the following pseudocode for `Collatz`(n), whose input is an integer $n \geq 1$.

1: **print** n
2: **if** $n = 1$ **then**
3:     **return**
4: **if** n is even **then**
5:     `Collatz`($n/2$)
6: **else**
7:     `Collatz`($3n + 1$)

It is not known if this is actually an algorithm; it is still unclear if it will terminate for each input n.

(a) What does `Collatz`(n) print to the terminal for $n = 12$, $n = 13$ and $n = 14$? This you can do by hand.

(b) Use a computer to investigate which n in the range $1 \leq n \leq 1000$ that makes the maximum number of recursive calls.

Problem 1.36
Consider the following recursive function `foo`(n), whose input is an integer $n \geq 1$.

1: **if** $n = 1$ **then**
2:     **return** 1
3: **if** $n = 2$ **then**
4:     **return** 3
5: **print** "n is " + n
6: $x \leftarrow$ `foo`($n - 2$)
7: **print** "x is " + x
8: $y \leftarrow$ `foo`($\lfloor n/3 \rfloor$)
9: **print** "y is " + y
10: **return** $x + y$

(a) Why can we be sure that this algorithm indeed always terminates on all valid inputs (in contrast to `Collatz`(n))?

(b) Draw the recursion tree and indicate which order the recursive calls are performed. What does `foo`(9) return?

(c) What is printed to the terminal upon executing `foo`(9)?

Problem 1.37
We define *valid xy-strings* recursively:

- The empty string is a valid xy-string, and
- if S is a valid xy-string, then $x + S + y$ a valid xy-string, and
- if S and T are valid xy-strings, then $S + T$ is a valid xy-string.

Provide an algorithm that returns `True` if the input string is a valid xy-string, `False` otherwise. Hint: You may want to use a stack data structure for this problem.

Problem 1.38
We define *valid `BB8`-strings* recursively:

- The empty string is a valid `BB8`-string, and
- if S is a valid `BB8`-string, then '`BB`'+S+'`8`' a valid `BB8`-string, and
- if S and T are valid `BB8`-strings, then $S + T$ is a valid `BB8`-string.

Provide an algorithm that returns `True` if the input string is a valid xy-string, `False` otherwise.

Problem 1.39
Suppose you have a very long text document with many occurences of different types of brackets: `()`, `[]`, and `{}`. You want to check if the brackets are properly matched. For example, the string `[()]` is properly matched, while the string `[(])` is not. Formalize the problem and provide an algorithm that checks if the brackets in a given string are properly matched. What if you would like to generate a string of properly matched brackets to test your code, how would you do it?

Problem 1.40
Read the description of the Project Euler problem 24.

(a) Provide an algorithm that solves the problem. Can you do better than a brute-force solution? In particular, can you solve the problem without generating *all* permutations of the digits 0 through 9?

(b) Generalize your algorithm to find the k-th permutation of a given list of n distinct characters (you may assume these characters can be compared in constant time).

(c) What is the runtime of your algorithm in part (b)? Can you make it depend on k only?

(d) Browse the Project Euler website and enhance your studies by solving some more problems.

Section 2 Sorting

2.1 Merge-Sort

Problem 2.1 Adapted from [1]
Illustrate the operation of merge sort on an array initially containing the sequence $[3, 41, 52, 26, 38, 57, 9, 49]$. What are the values for p, q and r in the different calls? What is the content of the array after each call to `Merge`?

Problem 2.2 [1]
The test in line 1 of the `Merge-Sort` procedure reads "if $p \geq r$" rather than "if $p \neq r$". If `Merge-Sort` is called with $p > r$, then the subarray $A[p : r]$ is empty. Argue that as long as the initial call of `Merge-Sort`$(A, 1, n)$ has $n \geq 1$, the test "if $p \neq r$" suffices to ensure that no recursive call has $p > r$.

2.2 Heap-Sort

Problem 2.3 [1]
Illustrate the operation of `Heap-sort` on the array $A = [5, 13, 2, 25, 7, 17, 20, 8, 4]$.

Problem 2.4 [1]
What are the minimum and maximum numbers of elements in a heap of height h?

Problem 2.5 [1]
Is the array with values $[33, 19, 20, 15, 13, 10, 2, 13, 16, 12]$ a max-heap?

Problem 2.6 [3]
You wish to store a set of n numbers in either a max-heap or a sorted array. For each application below, state which data structure is better, or if it does not matter. Explain your answers.

(a) Want to find the maximum element quickly.

(b) Want to be able to delete an element quickly.

(c) Want to be able to form the structure quickly.

(d) Want to find the minimum element quickly.

Problem 2.7 [1]
Illustrate the operation of `Max-Heapify`$(A, 3)$ and `Heap-Sort` on the array

$$A = [27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0].$$

2.3 Quick-Sort

Problem 2.8 [1]
Illustrate the operation of `Partition` on the array $A = [13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11]$.

Problem 2.9 [1]
Modify `Quick-Sort` to sort into monotonically decreasing order.

Problem 2.10 [3]
Use the partitioning idea of `Quick-Sort` to give an algorithm that finds the median element of an array of n integers in expected $O(n)$ time. (**Hint:** must you look at both sides of the partition?)

Problem 2.11 [1]

(a) What is the running time of `Quick-Sort` when all elements of array A have the same value?

(b) Show that the running time of `Quick-Sort` is $\Theta(n^2)$ when the array A contains distinct elements and is sorted in decreasing order.

2.4 Counting sort

Problem 2.12 [1]
Illustrate the operation of `Counting-Sort` on the array $A = [6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2]$.

2.5 Applications of sorting

Problem 2.13 [1]
Describe an algorithm that, given a set S of n integers and another integer x, determines whether S contains two elements that sum to exactly x. Your algorithm should take $\theta(n \log n)$ time in the worst case.

Problem 2.14 [1]
Let A be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an *inversion* of A.

(a) List the five inversions of the array $[2, 3, 8, 6, 1]$.

(b) What array with elements from the set $\{1, 2, \ldots, n\}$ has the most inversions? How many does it have?

(c) What is the relationship between the running time of insertion sort and the number of inversions in the input array? Justify your answer.

(d) Give an algorithm that determines the number of inversions in any permutation[1] on n elements. (Hint: Modify merge sort.)

Problem 2.15
A *k-cutoff value* C of an array A is the largest integer C such that *exactly* k entries of A are strictly larger than C.

(a) Prove that if A consists of n distinct integers and $0 < k \leq n$, then there exists a k-cutoff value. Can there exist a k-cutoff value of A if $k = 0$ or $k > n$?

(b) Provide an algorithm `CutOff`(A, k) that computes the k-cutoff value for a given array A and integer k, assuming that A consists of n distinct integers and $0 < k \leq n$.

(c) Give example of an array A of length 4 and an integer $0 < k \leq 4$ such that there is no k-cutoff value of A.

(d) Adapt the algorithm in part (b) to correctly handle the case when A consists of any collection of integers.

Section 3 Searching

3.1 ...in arrays

index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
value	3	7	12	18	18	23	27	31	35	40	44	44	49	53	58	62	67	71

index	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
value	75	80	84	89	93	97	102	106	111	115	119	124	128	133	137	142	146	150

index	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
value	153	159	164	168	172	177	181	186	190	193	195	198	198	199	200	200	200	200

Problem 3.1
Consider the big array A at the top of this section.

(a) How many comparisons between integers will `Linear-Search` do before (i) finding the element 44 in A? (ii) declaring that 182 is not in A?

(b) Illustrate the operation of `Binary-Search` while searching for the value (i) $x = 0$ (ii) $x = 44$ (iii) $x = 182$.

(c) Illustrate the operation of `Jump-Search` while searching for the value (i) $x = 0$ (ii) $x = 44$ (iii) $x = 182$.

(d) Illustrate the operation of `Exponential-Search` while searching for the value (i) $x = 0$ (ii) $x = 44$ (iii) $x = 182$.

(e) Technically, both `Jump-Search` and `Exponential-Search` requires the input array to consist of distinct elements, which our array A does not satisfy. Reflect on this: why is this assumption needed? Is it for the correctness, for the runtime analysis or both?

Problem 3.2
In the lecture, (simple) `Jump-Search` was outlined: rewrite this with pseudocode involving loop(s).

---

[1] Clarification: the input is an array with elements $1, 2, \ldots, n$ in some order.

3.2 . . . with search trees

Problem 3.3
Construct the binary search tree for the sequence $13, 19, 9, 5, 12, 8, 7, 4$ (insert in this order). Draw the tree after each insertion. After this, delete the value 5 from the search tree using `Tree-delete`.

Problem 3.4
Draw five different rooted trees with 8 vertices each.

(a) Determine the balance factor of each of the $5 \cdot 8 = 40$ nodes. Which of your trees are balanced?

(b) Pick a balanced tree (or construct one). Assign keys to the vertices so that (i) it is an AVL tree (ii) it is not an AVL tree

(c) For the AVL tree T from (b)(i), find a sequence of insertion of the keys into an initially empty tree that would result in T without any re-balancing operations.

. . . in progress . . . (suggestions from students welcomed)

References

[1] Thomas H. Cormen et al. *Introduction to Algorithms*. 4th ed. The MIT Press, 2022.

[2] Narasimha Karumanchi. *Data Structures and Algorithms Made Easy*. CareerMonk, 2021.

[3] Steven S. Skiena. *The Algorithm Design Manual*. Springer International Publishing, 2020.