Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement of Computer Science
Johannes Lengler, Markus Püschel, David Steurer
Kasper Lindberg, Kostas Lakis, Lucas Pesenti, Manuel Wiedmer

3 November 2025

Algorithms & Data Structures — Exercise sheet 7 — HS 25

The solutions for this sheet are submitted on Moodle until 9 November 2025, 23:59.

Exercises that are marked by * are challenge exercises. They do not count towards bonus points.

You can use results from previous parts without solving those parts.

Exercise 7.1 *1-3 subset sums* **(1 point)**.

Let $A[1, \ldots, n]$ be an array containing n positive integers, and let $b \in \mathbb{N}$. We want to know if there exists a subset $I \subseteq \{1, 2, \ldots, n\}$, together with multipliers $c_i \in \{1, 3\}$, $i \in I$ such that:

$$b = \sum_{i \in I} c_i \cdot A[i].$$

If this is possible, we say b is a 1-3 subset sum of A. For example, if $A = [16, 4, 2, 7, 11, 1]$ and $b = 61$, we could write $b = 3 \cdot 16 + 4 + 3 \cdot 2 + 3 \cdot 1$.

Describe a DP algorithm that, given an array $A[1, \ldots, n]$ of positive integers, and a positive integer $b \in \mathbb{N}$ returns True if and only if b is a 1-3 subset sum of A. Your algorithm should have asymptotic runtime complexity at most $O(b \cdot n)$.

In your solution, address the following aspects:

1. *Dimensions of the DP table*: What are the dimensions of the DP table?
2. *Subproblems*: What is the meaning of each entry?
3. *Recursion*: How can an entry of the table be computed from previous entries? Justify why your recurrence relation is correct. Specify the base cases of the recursion, i.e., the cases that do not depend on others.
4. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps?
5. *Extracting the solution*: How can the solution be extracted once the table has been filled?
6. *Running time*: What is the running time of your solution?

Exercise 7.2 *Longest ascending subsequence* **(1 point)**.

In this problem we will explore the Longest ascending subsequence problem discussed in lecture. We are given an array of integers $A[1 \ldots n]$ and we want to find length of a longest ascending subsequence of this array. Consider the partial problem

$$L(i) = \text{length of a longest ascending subsequence that ends in position } i.$$

(a) You are given the following:

1. *Dimensions of the DP table*: $DP[1 \dots n]$, a DP table of length n.
2. *Subproblems*: $DP[i] = L(i)$.

Now answer the next part:

3. *Recursion*: How can an entry of the table be computed from previous entries? Justify why your recurrence relation is correct. Specify the base cases of the recursion, i.e., the cases that do not depend on others.

(b) Compute the DP table for the following example, $A = [3, 10, 2, 1, 19, 4, 6, 21, 7]$. Answer with both the DP table and the length of a longest ascending subsequence of A.

Exercise 7.3 *Road trip.*

You are planning a road trip for your summer holidays. You want to start from city C_0, and follow the only road that goes to city C_n from there. On this road from C_0 to C_n, there are $n-1$ other cities $C_1, \dots, C_{n-1}$ that you would be interested in visiting (all cities $C_1, \dots, C_{n-1}$ are on the road from C_0 to C_n). For each $0 \leq i \leq n$, the city C_i is at kilometer k_i of the road for some given $0 = k_0 < k_1 < \dots < k_{n-1} < k_n$.

You want to decide in which cities among $C_1, \dots, C_{n-1}$ you will make an additional stop (you will stop in C_0 and C_n anyway). However, you do not want to drive more than d kilometers without making a stop in some city, for some given value $d > 0$ (we assume that $k_i < k_{i-1} + d$ for all $i \in [n]$ so that this is satisfiable), and you also don't want to travel backwards (so from some city C_i you can only go forward to cities C_j with $j > i$).

(a) Provide a *dynamic programming* algorithm that computes the number of possible routes from C_0 to C_n that satisfy these conditions, i.e., the number of allowed subsets of stop-cities. Your algorithm should have $O(n^2)$ runtime.

In your solution, address the following aspects:

1. *Dimensions of the DP table*: What are the dimensions of the DP table?
2. *Subproblems*: What is the meaning of each entry?
3. *Recursion*: How can an entry of the table be computed from previous entries? Justify why your recurrence relation is correct. Specify the base cases of the recursion, i.e., the cases that do not depend on others.
4. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps? Describe the calculation order in pseudocode.
5. *Extracting the solution*: How can the solution be extracted once the table has been filled?
6. *Running time*: What is the running time of your solution?

(b) If you know that $k_i > k_{i-1} + d/10$ for every $i \in [n]$, how can you turn the above algorithm into a linear time algorithm (i.e., an algorithm that has $O(n)$ runtime) ?

Exercise 7.4 *String counting* **(1 point)**.

Given a binary string $S \in \{0,1\}^n$ of length n, let $f(S)$ be the number of times "11" occurs in the string, i.e. the number of times a 1 is followed by another 1. In particular, the occurrences do not need to be disjoint. For example $f(\text{“}\underline{11}10\underline{1}1\text{”}) = 3$ because the string contains three 1 (underlined) that are followed by another 1. Given n and k, the goal is to count the number of binary strings S of length n with $f(S) = k$.

Describe a DP algorithm that, given positive integers n and k with $k < n$, reports the required number. Your solution should have complexity at most $O(nk)$.

In your solution, address the following aspects:

1. *Dimensions of the DP table*: What are the dimensions of the DP table?
2. *Subproblems*: What is the meaning of each entry?
3. *Recursion*: How can an entry of the table be computed from previous entries? Justify why your recurrence relation is correct. Specify the base cases of the recursion, i.e., the cases that do not depend on others.
4. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps? Describe the calculation order in pseudocode.
5. *Extracting the solution*: How can the solution be extracted once the table has been filled?
6. *Running time*: What is the running time of your solution?

Hint: *Define a three dimensional DP table* $DP[1 \dots n][0 \dots k][0 \dots 1]$.

Hint: *The entry* $DP[i][j][l]$ *counts the number of strings of length* i *with* j *occurrences of* "11" *that end in* l *(where* $1 \le i \le n$*,* $0 \le j \le k$ *and* $0 \le l \le 1$*).*

Exercise 7.5 *Searching a doubly sorted array.*

You are given a 2D array of numbers $A[1 \dots n][1 \dots n]$. The goal is to understand the search problem: given x, determine whether x occurs in A, and if so, find it. Without any structure on A, this would take $O(n^2)$ time since we would have to check every entry. However, by assuming more structure on A, we can hope to bring this runtime down. All sorting in this problem will be in ascending order.

Suppose A satisfies the following property: Every row is sorted left-to-right.

(a) Design a search algorithm that runs in time $O(n \log(n))$.

Now suppose A satisfies a stronger property: Every row is sorted left-to-right and every column is sorted top-to-bottom.

(b) Design a search algorithm that runs in time $O(n)$.

Hint: *Say we want to search for an element* x*. Suppose that* $A[i,j] < x$*, for some* $1 \le i, j \le n$*, then what can you say about entries* $A[k,j]$ *for* $1 \le k < i$ *and* $A[i,\ell]$ *for* $1 \le \ell < j$*?*

(c)* Argue that this bound is tight. Namely, that any search algorithm must run in time $\Omega(n)$.

Hint: *Consider an assignment of numbers to the off-diagonal entries. Off-diagonal entries are entries at* $A[i, n-i+1]$ *for* $1 \le i \le n$*. When do we have a 2D array of numbers which satisfies that off-diagonal assignment and the stronger property?*