This assignment is **due on October 27** and should be submitted on Gradescope. All submitted work must be *done individually* without consulting someone else's solutions in accordance with the University's "Academic Dishonesty and Plagiarism" policies.

Before you read any further, go to the last page of this document and read the Written Assignment Guidelines section.

Problem 1. (10 points)
We're building a new software project out of existing pieces. Each of these pieces offers a set of functionalities if it's included in the final product and we want to ensure that our final product has all functionalities our pieces have to offer (i.e., at least one piece that has the functionality is part of our final product).

Formally, we are given a non-empty set of pieces P and a collection of sets (each set representing a different functionality we want) $F = \{F_1, ...F_m\}$, such that every $F_i \subseteq P$ and for every piece $p \in P$, there exists a F_i that contains it. We need to find that smallest number of pieces such that we have at least one piece in each F_i.

For example, if $P = \{1, 2, 3\}$ and $F = \{\{1\}, \{2, 3\}\}$, returning either $\{1, 2\}$ or $\{1, 3\}$ would be an optimal solution. Returning $\{1, 2, 3\}$ would not be optimal, as this isn't the smallest number of pieces that we could return. Returning $\{1\}$ is incorrect, as we don't return any piece of the second set in F.

Construct a counterexample for the following algorithm for this problem: Sort the pieces in non-increasing order based on the number of sets a piece occurs in. Process the pieces one at a time and add a piece to the solution if it is part of a set that doesn't have any piece in the solution yet.

Remember to:

a) describe your instance,

b) show what solution the algorithm gives for the instance and briefly explain why, and

c) show what the optimal solution of the instance is.

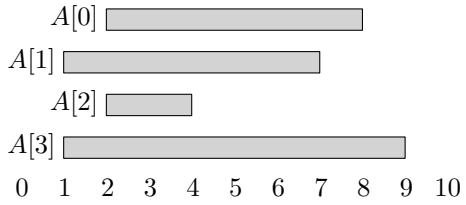
Problem 2. (25 points)
We're helping out with a scientific experiment where we need to take samples of a given set of specimens. These specimens are generally given as 2-dimensional areas on for example Petri dishes. Unfortunately for us, your clumsy lecturer André dropped the petri dishes, and some of the Petri dishes were broken and a number specimens have gotten mixed. However, this does present us with a unique opportunity to take samples of all specimens faster than taking these samples one at a time: when we take a sample at some location p, our sample includes a little bit of all specimens that got mixed at this location. Your task is

to design an algorithm that returns (the locations of) the minimum number of samples needed to sample all specimens.

Since this problem is a bit difficult to solve in 2D, we'll look at the slightly simplified 1D version instead: You're given an array A of length n consisting of 1-dimensional line segments along the x-axis (i.e., tuples of left and right endpoints) and you can assume that each specimen $A[i] \in A$ is given as $[l_i, r_i]$. Your task is to compute (the locations of) the minimum number of samples we need to take to ensure that our samples cover all specimens.

Example:
$A = [[2,8],[1,7],[2,4],[1,9]]$

In other words, it looks like this:



In this example, the smallest set of samples is $\{3\}$, as taking a single sample there would ensure we cover all four specimens. The set $\{1,4\}$ would also cover all specimens, but this isn't the smallest such set. The set $\{1\}$ doesn't cover all specimens, so it isn't a valid solution.

Design a greedy algorithm that returns the locations of the minimum number of samples needed to sample all specimens. For full marks, your algorithm should run in $O(n \log n)$ time. Remember to:

a) describe your algorithm in plain English,

b) argue its correctness, and

c) analyze its time complexity.

Problem 3. (25 points)
For the Winter Olympics of 2086, a new extreme sport will be introduced: snowboarding with rocket-boosted snowboards! To make this event a success we're looking for a good place in the Snowy Mountains to organise this. The location we use should meet a number of conditions, but for security reasons we aren't told what exactly we're looking for yet. For example, the event may need to start from a location that has a downward slope, or we may want to finish on a flat part so the competitors can be interviewed. Another condition may be that we want to include at least one ramp for the competitors to perform a nice jump for the cameras. While we don't know what the organisers will ask for yet, we do know that there will be exactly three things that the organisation is looking for.

After scouting the Snowy Mountains, we've been provided with a string A of length n encoding what each location can be used for. We're also given a 3-character string S, which encodes what the organisation is looking for. You can treat the strings as arrays containing a single character at each position. We need to determine the number of different ways to extract S (in order) from A to give the organisation a good idea of the number of options they have to consider. The characters of S do **not** have to be consecutive in A, as long as they occur in the correct order.

Example:
$A = acabdcb$
$S = acb$
In this example, we can extract S in four ways: 1. $A[0], A[1], A[3]$, 2. $A[0], A[1], A[6]$, 3. $A[0], A[5], A[6]$, and 4. $A[2], A[5], A[6]$. Thus, your algorithm should return 4.

Design a divide and conquer algorithm for the above problem. For full marks, your algorithm should run in $O(n)$ time. Remember to:

a) describe your algorithm in plain English,

b) argue its correctness, and

c) analyze its time complexity. (If you use the Master Theorem, don't forget to specify the parameters you get when you apply, so we can see you understand how this works.)

Written Assignment Guidelines

- Assignments should be typed and submitted as pdf (no pdf containing text as images, no handwriting).
- Start by typing your student ID at the top of the first page of your submission. Do **not** type your name.
- Submit only your answers to the questions. Do **not** copy the questions.
- When asked to give a plain English description, describe your algorithm as you would to a friend over the phone, such that you completely and unambiguously describe your algorithm, including all the important (i.e., non-trivial) details. It often helps to give a very short (1-2 sentence) description of the overall idea, then to describe each step in detail. At the end you can also include pseudocode, but this is optional.
- In particular, when designing an algorithm or data structure, it might help you (and us) if you briefly describe your general idea, and after that you might want to develop and elaborate on details. If we don't see/understand your general idea, we cannot give you marks for it.
- Be careful with giving multiple or alternative answers. If you give multiple answers, then we will give you marks only for "your worst answer", as this indicates how well you understood the question.
- Some of the questions are very easy (with the help of the slides or book). You can use the material presented in the lecture or book without proving it. You do not need to write more than necessary (see comment above).
- When giving answers to questions, always prove/explain/motivate your answers.
- When giving an algorithm as an answer, the algorithm does not have to be given as (pseudo-)code.
- If you do give (pseudo-)code, then you still have to explain your code and your ideas in plain English.
- Unless otherwise stated, we always ask about worst-case analysis, worst-case running times, etc.
- As done in the lecture, and as it is typical for an algorithms course, we are interested in the most efficient algorithms and data structures, though slower solutions may receive partial marks.
- If you use further resources (books, scientific papers, the internet,...) to formulate your answers, then add references to your sources and explain it in your own words. Only citing a source doesn't show your understanding and will thus get you very few (if any) marks. Copying from any source without reference is considered plagiarism.