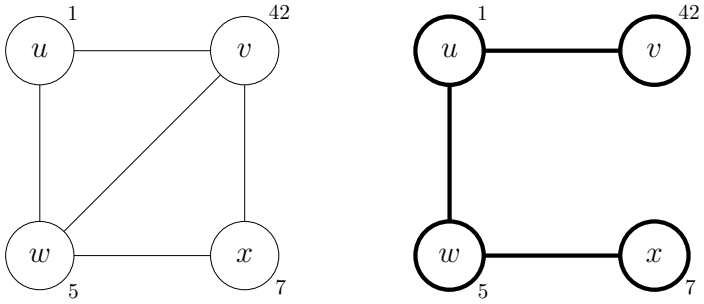
This assignment is **due on May 22** and should be submitted on Gradescope. All submitted work must be *done individually* without consulting someone else's solutions in accordance with the University's "Academic Dishonesty and Plagiarism" policies.

As a first step go to the last page and read the section: Advice on how to do the assignment.

Problem 1. (10 points)
We are given a simple connected undirected graph $G = (V, E)$. Every vertex v has a distinct positive integer p_v associated with it. We define a monotone spanning tree as a spanning tree where every root to leaf path in the tree encounters a non-decreasing sequence of integers. We want to compute such a monotone spanning tree T of graph G, if one exists (return *null* otherwise).

Example:



In the example above, the graph on the left is the input graph. The numbers next to the vertices are their associated integers. The tree on the right, rooted at u, is one of the possible monotone spanning trees of the graph. It contains two root to leaf paths: u, v $(1, 42)$ and u, w, x $(1, 5, 7)$. Both of these paths visit non-decreasing integers and are thus monotone.

We are given two algorithms for this problem:

modifiedPrim works similar to Prim's MST algorithm: we maintain a set S of vertices that are already part of our tree and in every iteration of the algorithm, we add the vertex (adjacent to a vertex in S) with smallest integer to our tree. Initially, our set S consists of only the vertex s that has the smallest integer associated with it. Of course, before adding a vertex to S, we need to check whether its integer is at least as large as that of its parent in the tree. If this isn't the case, we report that the graph has no monotone spanning tree.

modifiedDFS runs DFS starting from the vertex with the smallest integer associated with it. Every time it processes a vertex, it visits its neighbors that haven't been visited yet and have an integer that's at least as large as its own. If at the end we have visited all vertices of the graph, we return the tree, and *null* otherwise.

```
1: function modifiedPrim(G)
2:     T ← ∅
3:     H ← new heap containing only (p_s, s), i.e., the pair of vertex s and its
       associated positive integer using the integer as the key.
4:     while H ≠ ∅ do
5:         u ← H.removeMin()
6:         Add u to T, along with the edge to its parent (s has no parent)
7:         for (u, v) incident to u do
8:             if v ∉ T and v doesn't have a parent then
9:                 if p_u > p_v then
10:                    return null
11:                else
12:                    Set u to be the parent of v
13:                    Insert (p_v, v) into H
14:    Set s to be the root of T
15:    return T
```

```
1: function modifiedDFS(G)
2:     Initialize visited as in DFS
3:     Let s be the vertex with smallest integer.
4:     T ← ({s}, ∅)
5:     Set s to be the root of T
6:     modifiedDFSvisit(s)
7:     if T contains all vertices of G then
8:         return T
9:     else
10:        return null
```

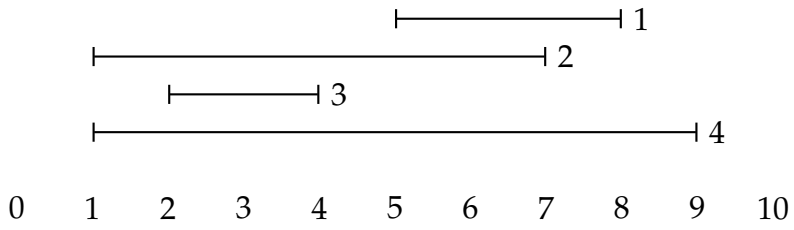
```
1: function modifiedDFSvisit(u)
2:     Set visited[u] to true
3:     for (u, v) incident to u do
4:         if not visited[v] and p_u ≤ p_v then
5:             Add vertex v and edge (u, v) to T
6:             modifiedDFSvisit(v)
```

a) Argue whether modifiedPrim always returns the correct answer by either arguing its correctness (if you think it's correct) or by providing a counterexample (if you think it's incorrect).

b) Argue whether modifiedDFS always returns the correct answer by either arguing its correctness (if you think it's correct) or by providing a counterexample (if you think it's incorrect).

Problem 2. (25 points)
You've been hired by Telstra to set up mobile phone towers along a straight road in a rural area. There are n customers. Each customer's mobile device has a specific range in which it can send and receive signals to and from a tower. The goal is to ensure that each customer is within range of a tower while using as few towers as possible. More formally, each customer has an associated interval $[\ell_i, r_i]$, where ℓ_i and r_i are positive integers ($l_i \leq r_i$). The interval of a customer specifies the range of their mobile device. Serving customer i requires us to place a tower at a location x satisfying $\ell_i \leq x \leq r_i$. A solution is a set L of tower locations and is feasible if for each customer i, there exists a location $x \in L$ satisfying $\ell_i \leq x \leq r_i$. We're interested in constructing the *smallest* set L that is feasible.

Example:



Suppose we have 4 customers with intervals $[5,8], [1,7], [2,4], [1,9]$ (see the above figure). In this case, $L = \{2,6\}$ is an optimal solution. $L = \{3\}$ isn't a solution, as we customer 1 doesn't have any tower in their range. $L = \{2,3,6\}$ is a solution, but it isn't the smallest one, as it contains three towers, while having two would have sufficed.

Your task is to design a greedy algorithm that returns the *smallest* set of tower locations that is feasible. For full marks, your algorithm should run in $O(n \log n)$ time. Remember to:

a) Describe your algorithm in plain English.

b) Prove the correctness of your algorithm.

c) Analyze the time complexity of your algorithm.

Problem 3. (25 points)
Alice and Bob want to split a log cake between the two of them. The log cake is n centimeters long and they want to make one slice with the left part going to Alice and the right part going to Bob. Both Alice and Bob have different values for different parts of the cake. In particular, if the slice is made at the i-th centimeter of the cake, Alice receives a value $A[i]$ for the first i centimeters of the cake and Bob receives a value $B[i]$ for the remaining $n - i$ centimeters of the cake. Alice and Bob receives strictly higher values for larger cuts of the cake: $A[0] < A[1] < \ldots < A[n]$ and $B[0] > B[1] \ldots > B[n]$. Ideally, they would like to cut the cake fairly, at a location i such that $A[i] = B[i]$, if it exists. Such a location is said to be *envy-free*.

Example:

When $A = [1, 4, 6, 10]$ and $B = [20, 10, 6, 4]$ then 2 is the envy-free location, since $A[2] = B[2] = 6$.

Your task is to design a divide and conquer algorithm that returns an envy-free location if it exists and otherwise, to report that no such location exists. For full marks, your algorithm should run in $O(\log n)$ time. Remember to:

a) Describe your algorithm in plain English.

b) Prove the correctness of your algorithm.

c) Analyze the time complexity of your algorithm.

Advice on how to do the assignment

- 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.
- 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.