CS5800 Algorithms ***A***

Solutions for Exam on December 10, 2019

Ravi Sundaram

- **Logistics:** Read all these instructions. Write your name on the top of every page. Write your name in uppercase in the blank provided on this page. If you need extra space, use the blank facing page.
- **Duration, Problems and Points:** This exam lasts 150 minutes and has 10 problems totaling 100 points. Make sure to read all problems to decide an order to attempt them that would allow quickest progress.
- **Known algorithms:** While describing a new algorithm, you may use any algorithm covered in class as a subroutine without elaboration.
- **Proofs and analyses:** You do not need to provide a proof of correctness and/or an analysis of the running time *unless* explicitly asked to.
- **Partial Credit:** Points are determined on the basis of both the correctness and the clarity of description. Show your work, as partial credit may be given. Present the best algorithm you can, even if it doesn't meet all desired properties.
- **Closed-book exam:** This exam is closed-book. No sources of any kind, electronic or physical, can be consulted during the course of this exam. No collaboration of any kind is permitted.
- **Policy on cheating:** Students who violate the above rules on scholastic honesty are subject to disciplinary penalties. Any student caught cheating will *immediately* receive an **F** (failing grade), and the case will be forwarded to the Office of Student Conduct and Conflict Resolution.

  Sign below and confirm that you are strictly abiding by the rules of this test.

Signature: _____

NAME:

Question	1	2	3	4	5	6	7	8	9	10	Total
Points	7	5	10	15	10	10	10	10	10	13	100

Best of luck!

Problem 1 (True/False) 7

Indicate whether the following statements are True or False by filling in the table with T or F respectively:

a) Asymptotics: $(\log n)^{\log n} = \omega(\log \log n)$

b) For any 2 functions f and g, we always have that either $f = O(g)$ or $g = O(f)$.

c) If all of the edge capacities in a graph are an integer multiple of 5, then the value of the maximum flow from source to sink will be a multiple of 5.

d) In a directed graph $G = (V, E)$ with capacities on edges and a specified source s and sink t the set of nodes in the minimum $s - t$ cut is always unique.

e) In a connected weighted graph with distinct positive weights, the edge with maximum weight is never in the minimum spanning tree.

f) Given n-bit numbers, A and B, the product $P = A \times B$ cannot be calculated in poly(n) time.

g) If there is a polynomial-time algorithm for any **NP**-Complete problem, then every problem in **NP** has a polynomial-time algorithm.

Question	a	b	c	d	e	f	g
T/F	T	F	T	F	F	F	T

Problem 2 (Recurrences) 5

Use the **recursion tree method** to solve the following recurrence:

$$T(n) = 4T(n/3) + n; \quad T(1) = 1$$

Solution:

There are a total of $\log_3 n$ levels in this tree, where work done in zeroth level $= n$, work in first level $= \frac{4}{3} \times n$, second level $= (\frac{4}{3})^2 \times n$ and so on.

Hence $(\frac{4}{3})^i \times n$ work is done at the i^{th} level. Leaf nodes do 1 work, and there are $4^{\log_3 n} = n^{\log_3 4}$ of them. Total work $= \Sigma_{i=0}^{\log_3 n} (\frac{4}{3})^i \times n + n^{\log_3 4} = O(n^{\log_3 4})$ ■

Problem 3 (Graph Traversals) 10

Give the Breadth First Search Tree for the following graph $\mathcal{G}$, starting at the node A. Use the template given on the next page to fill in the edges *with directions*.

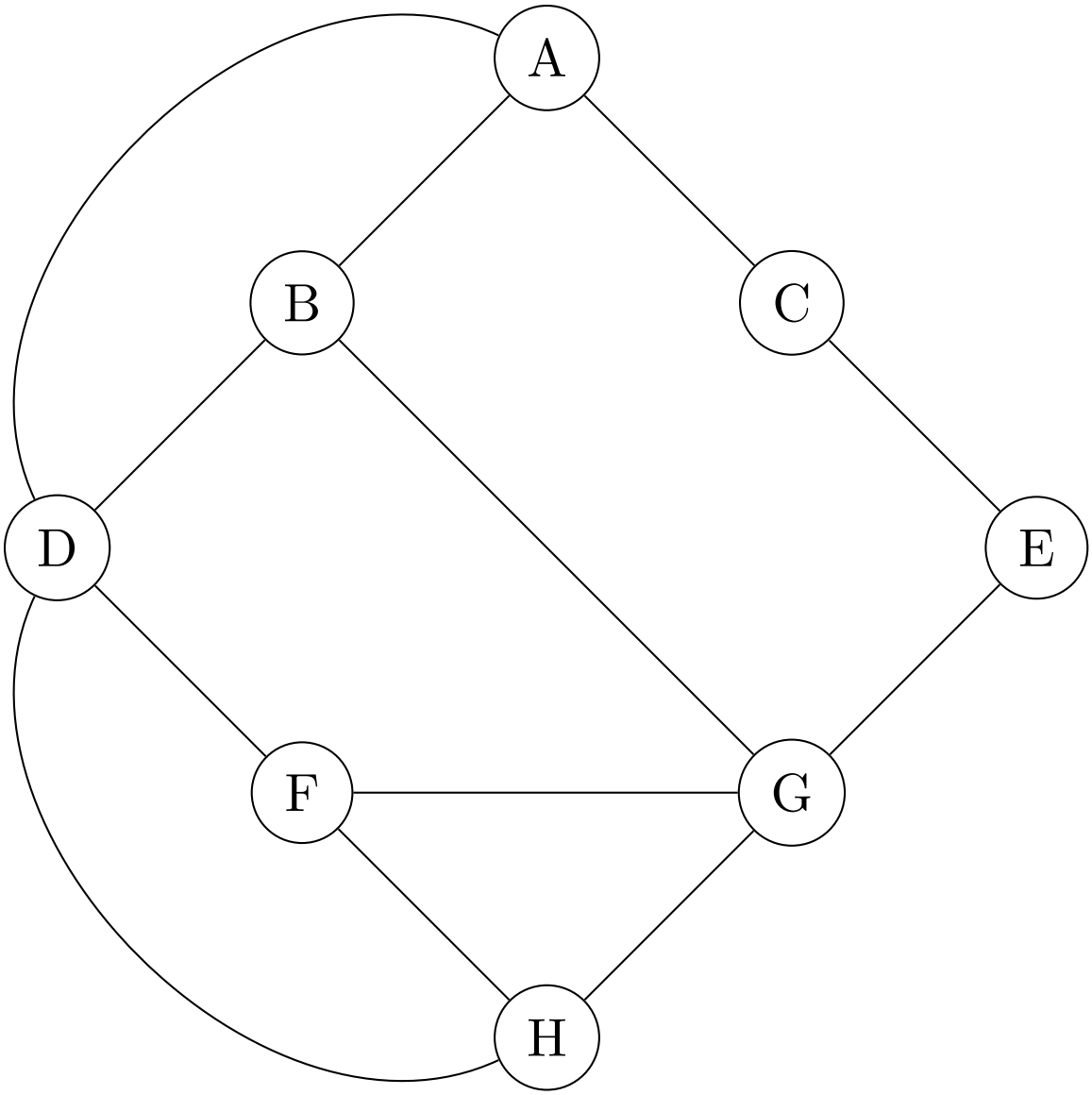


Figure 1: Graph $\mathcal{G}$

Instructions: Make sure to mark directions in the Breadth First Search Tree, away from the starting node or root A, marking the direction in which each edge is traversed on the path from the root to other nodes. If there are multiple possible trees, you can fill in any correct tree.

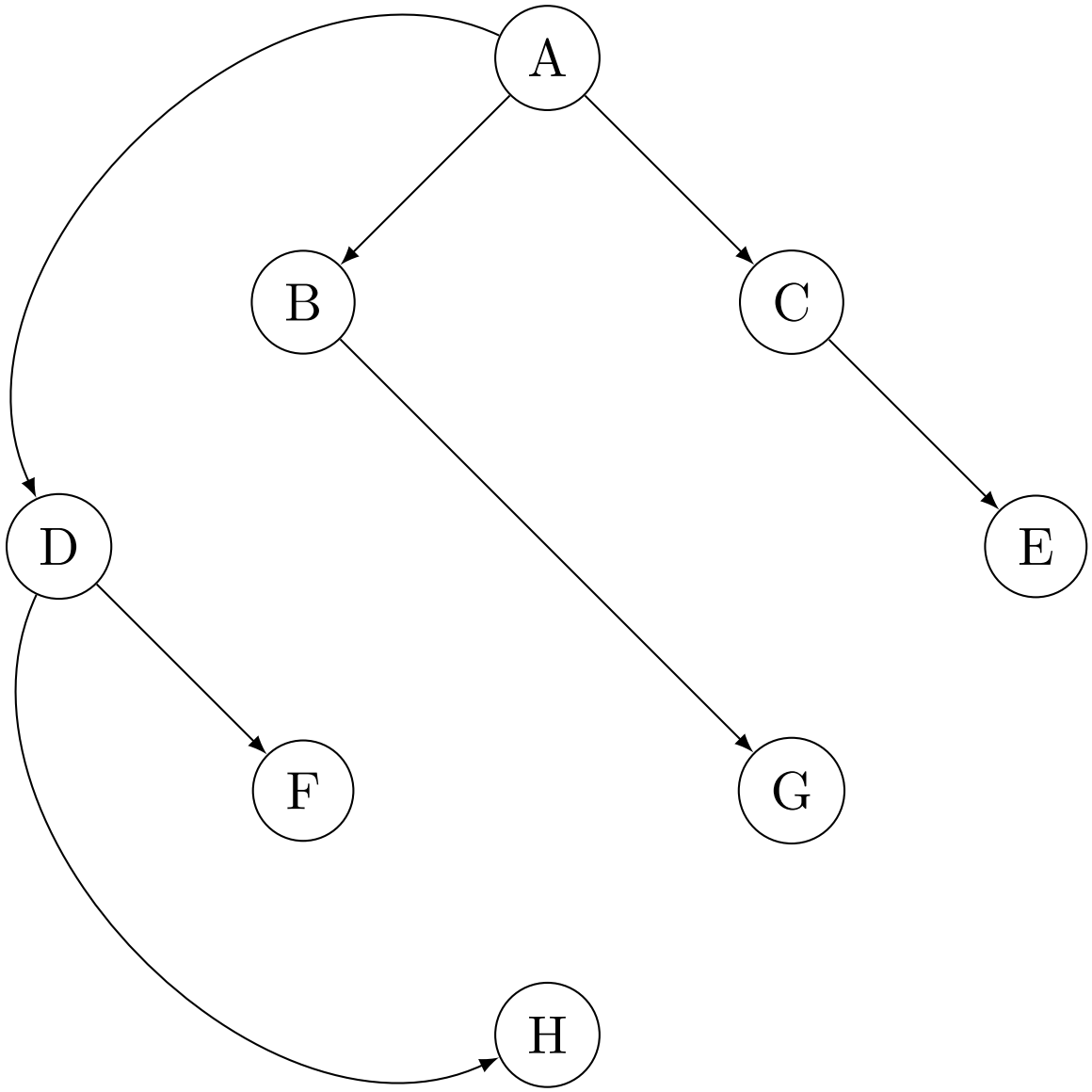


Figure 2: Breadth First Search Tree starting from A.

Problem 4 (Manipulating Linear Programs) 5+5+5

Consider the following LP:

$$\begin{array}{ll} \min & 2x - 5y \\ \text{s. t.} & -x - 7y \leq -3 \\ & 2x + y \geq 11 \\ & y \geq 0 \end{array}$$

1. Write the LP in Standard Form: a max objective, equality constraints and non-negative variables. *Note: If you need to use slack variables, call them $s_1, s_2, \ldots$ and if you need to enforce a non-negativity condition on an unconstrained variable z use the z^+ and z^- superscript notation.*

   Solution:

$$\begin{array}{ll} \max & -2(x^+ - x^-) + 5y \\ \text{s. t.} & -(x^+ - x^-) - 7y + s_1 = -3 \\ & 2(x^+ - x^-) + y - s_2 = 11 \\ & x^+, x^-, y, s_1, s_2 \geq 0 \end{array}$$

   ■

2. Write the LP in Canonical Form: a min objective, $\geq$ constraints and non-negative variables.

   Solution:

$$\begin{array}{ll} \min & 2(x^+ - x^-) - 5y \\ \text{s. t.} & (x^+ - x^-) + 7y \geq 3 \\ & 2(x^+ - x^-) + y \geq 11 \\ & x^+, x^-, y \geq 0 \end{array}$$

   ■

3. Write the dual of the Canonical form of the LP. Use $a, b, c, d, \ldots$ (as many as needed) as the new variables in the dual.

   Solution:

$$\begin{array}{ll} \max & 3a + 11b \\ \text{s. t.} & a + 2b \leq 2 \\ & -a - 2b \leq -2 \\ & 7a + b \leq -5 \\ & a, b \geq 0 \end{array}$$

   ■

Problem 5 (Vertex Cover to Clique) 10

Give a polynomial-time reduction from VERTEX COVER to CLIQUE, that is, VERTEX COVER $\leq_p$ CLIQUE. You only need to give a polynomial time reduction and briefly explain why it is correct. We give the definitions of Vertex Cover, Clique, the corresponding decision problems, as well as a useful lemma here:

Definition (Vertex Cover)**.** *A* vertex cover *in a graph $G = (V, E)$ is a subset of vertices $V' \subseteq V$ such that for all edges $\{u, v\} \in E$, at least one of u and v belong to V'.*

Problem (VERTEX COVER)**.** *Given a graph $G = (V, E)$ and a positive integer $k \leq |V|$, does G contain a vertex cover V' with $|V'| \leq k$?*

Definition (Clique)**.** *A* clique *in a graph $G = (V, E)$ is a subset of vertices $V' \subseteq V$ such that for every pair of vertices $u, v \in V'$, the edge $\{u, v\} \in E$.*

Problem (CLIQUE)**.** *Given a graph $G = (V, E)$ and a positive integer $k \leq |V|$, does G contain a clique V' with $|V'| \geq k$?*

Lemma. *For any graph $G = (V, E)$ and subset $V' \subseteq V$, the following statements are equivalent:*

a. *V' is a vertex cover for G.*

b. *$V \setminus V'$ is a clique in the complement G^c of G, where $G^c = (V, E^c)$ with $E^c = \{\{u, v\} : u, v \in V \text{ and } \{u, v\} \notin E\}$.*

Solution:

We are going to a give reduction from VERTEX COVER to CLIQUE. The reduction from CLIQUE to VERTEX COVER is similar. For our first step, we need to reduce an arbitrary instance of VERTEX COVER to some instance of CLIQUE.

So suppose we are given an arbitrary instance of VERTEX COVER. By the definition of the problem, that means we are given some graph $G = (V, E)$ and some parameter k. We convert this to an instance of CLIQUE. That means we need to create a new graph G' and a new parameter k'. Using G, we create the graph $G' = (V, E^c)$. And from k, we create the new parameter $k' = n - k$, where $n = |V|$. Now our instance of CLIQUE is $G' = (V, E^c)$ and $n - k$.

Next we need to show the proof of correctness. To do this, we show that there's a vertex cover in G of size at most k if and only if there's a clique in G' of size at least $n - k$. In other words, we need to show that CLIQUE(G, k) = YES $\iff$ VERTEX COVER$(G', n - k)$ = YES. There are two directions we need to show.

First, suppose there's a vertex cover $U \subseteq V$ in G of size $l \leq k$. Then by the lemma, $V \setminus U$ is a clique in G' of size $n - l \geq n - k$.

Second, suppose there's a clique $R \subseteq V$ in G' of size $m \geq n - k$. Then by the lemma, $V \setminus R$ is a vertex cover in G of size $n - m \leq k$.

■

Problem 6 (k^{th} smallest element and Keaps) 3+7

You want to create a new data structure called a `Keap` in which you can insert elements and that can return the k^{th} smallest element among the elements that have been inserted at any given point in time. Your `Keap` needs to support the following two operations

- `Insert`(x): Insert element x into the `Keap`.
- `Peek`(): Returns the value of the k^{th} smallest element in the `Keap`.

For example, suppose $k = 2$ and that you are first given the elements $4, 6, 2$. Then on your `Keap` $\mathcal{K}$ you need to run $\mathcal{K}$.`Insert`$(4), \mathcal{K}$.`Insert`$(6), \mathcal{K}$.`Insert`(2). Now $\mathcal{K}.Peek()$ should return 4. Now say your next integer is 1. Then you run $\mathcal{K}$.`Insert`(1). Now $\mathcal{K}.Peek()$ should return 2.

a) Suppose we implement the `Keap` by inserting all the n elements in a min-heap. To `Peek`(), we extract the minimum element k times, return the last extracted minimum element, and finally re-insert the first k extracted elements back into the heap. What are the time complexities of the `Keap` operations - `Insert`(x) and `Peek`()?

- Time Complexity of `Insert`(x):

  Solution:

  $O(\log n)$ ■

- Time Complexity of `Peek`():

  Solution:

  $O(k \log n)$ ■

b) Show how you can implement a `Keap` so that `Insert`(x) takes $O(\log k)$ time and `Peek`() takes $O(1)$ time (even when the number of integers to be inserted into the `Keap` continues to be n). You can assume black box access to standard data structures without implementing them. Briefly explain the time complexity of the operations in your `Keap` implementation.

Solution:

Create a max heap to keep track of the smallest k elements. Insertion into a heap of size k takes $O(\log k)$ time, this will be the time for `Insert`(x) in the `Keap`. Peeking in the `Keap` takes $O(1)$ time since all we need to do is return value of root of the heap. When our heap is full with k elements, when we insert a new element x we check it with the root value. If x is larger than the root, then we ignore it. Otherwise, we remove the root and insert x in $O(\log k)$ time.

■

Problem 7 (Local maximum) 3+7

Given an array $A = [a_1, a_2, \ldots, a_n]$ of distinct positive integers which is not necessarily sorted, find any local maximum in the array A. An element at index $1 < i < n$ is a local maximum if a_i is at least as big as elements on both side of it. That is $a_i \geq a_{i-1}$ and $a_i \geq a_{i+1}$. For $i = 1$ or $i = n$, we only compare $a_1 \geq a_2$ and $a_n \geq a_{n-1}$ respectively.

a) Give a $\Theta(n)$ time algorithm.

Solution:

A possible solution : A global maxima is also a local optima.

Algorithm 1 localOptima

```
1: max = −∞
2: for a ∈ A do
3:     if a > max then
4:         max = a
   return max
```

■

b) Give an $O(\log n)$ algorithm using divide-and-conquer.

Solution:

Algorithm 2 localOptima(A,n,lo,hi)

```
1: if lo ≥ hi then return A[lo]
2: mid = (lo+high)/2
3: if A[mid + 1] > A[mid] then return localOptima(A, n, mid + 1, hi)
4: else if A[mid − 1] > A[mid] then return localOptima(A, n, lo, mid − 1)
5: elsereturn A[mid]
```

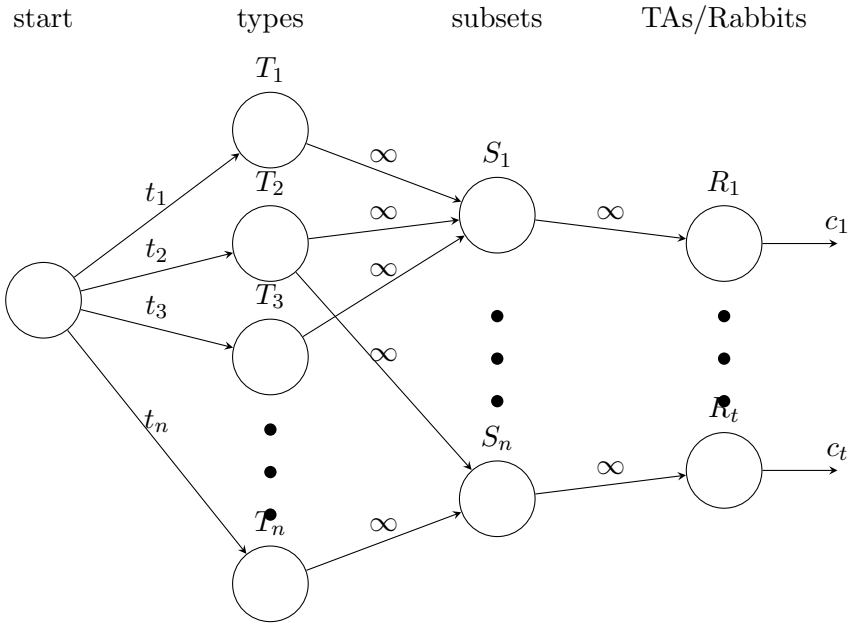
We can use a binary search like idea to reduce our array size by a factor of 2 at each recursive call and do a constant work at each step. That gives us a divide and conquer solution which runs in time $O(log\ n)$. ■

Problem 8 (Network Flow Reduction) 10

A professor is trying to assign TAs to questions to grade. There are n types of questions to grade, with t_i questions of type i, for $1 \leq i \leq n$. There are m TAs, where TA j has the time to grade at most c_j questions, and is capable of only grading questions of types drawn from a subset $S_j \subseteq \{1, 2, \ldots, n\}$ of types.

Give a polynomial-time algorithm to determine if the professor can assign TAs to the questions so that all the questions are graded. If it is possible to do so, your algorithm should also return an assignment that indicates how many questions of each type are graded by each TA. There is no need to prove the correctness of the algorithm or state a running time for your algorithm.

Solution:



We create a flow graph as follows and then connect all the TAs to a sink and find the max-flow. If the max-flow is equal to $\sum_i t_i$ then we know that all the questions have been graded. The assignment is simply whichever of the inner edges have non-zero capacities. These will tell us which subsets were used to assign questions to corresponding TAs.

■

Problem 9 (Reducing to Integer Linear Programming) 10

Reduce the following variant of the Knapsack problem to Integer Linear Programming (ILP). That is, write an ILP whose solution will give the solution to the Knapsack variant. If you use indicator variables, briefly describe what they represent. Also briefly describe what each constraint is enforcing.

Problem (Knapsack)**.** *You want to choose a subset of a set of n items, with item i having a value v_i and weight w_i. If the total weight of the items you choose in your subset is at most a bound B, what is the maximum total value you can achieve?*

Solution:

Note that we need to choose which items to pick and which to not pick for our Knapsack. The notion of picking/not picking can be encoded using $0-1$ indicator variables x_i for each item, with 0 indicating that we have not picked the item for our subset and 1 indicating that we have picked the item for our subset. Now if we take $\sum_i x_i v_i$ then only the values in our picked subset get added up, everything else is zeroed out. Similarly for $\sum_i x_i w_i$. Thus the ILP will be:

$$\begin{aligned} \max \quad & \sum_i x_i v_i && \text{Maximize value of items picked up in our subset} \\ \text{s.t.} \quad & \sum_i x_i w_i \leq B && \text{Enforcing the bound B on maximum weight} \\ & x_i \in \{0,1\} \end{aligned}$$

■

Problem 10 (LIS: Longest Increasing Subsequence) 3+10

You are given a sequence of integers $A = [a_1, a_2, \ldots, a_m]$ and you need to find the length of the longest increasing subsequence (LIS) of A. We say a sequence B is increasing if $B_i \leq B_{i+1}$ for all indices $i, i+1$.

For example in $A = [7, 2, 3, 1, 4, 3, 5]$, the longest increasing subsequence is $B = (2, 3, 4, 5)$ and the length outputted by LIS is 4.

a If you had black-box access to an algorithm that outputted the length of the Longest Common Subsequence (LCS) of two sequences A and B, show how to use that to solve the Longest Increasing Subsequence (LIS) problem.

b Assuming you did not have access to LCS, solve LIS in poly-time using dynamic programming. You only need to write the English description of your DP and the Bellman Equation (with base cases).

Solution:

1. Let M be the sorted ascending L.
   Then we query LCS with $A = L, B = M$ and get the answer.

2. There's more than one way to solve this. One possible solution is as follows.

   Let $OPT[i]$ denote the longest increasing subsequence in $[a_1, a_2, \ldots a_i]$ **that has a_i as its last element**. Again, a_i has to be a part of the subsequence. To find $OPT[i]$, we can check the largest $OPT[j]$ for $1 \leq j < i$ such that $a_j \leq a_i$ and add 1. Since if this condition is met, we know we can append a_i to the subsequence that ends at a_j. And we take the max over all these earlier subsequences, so that gives us the largest possible value for $OPT[i]$.

   To make the base case convenient, we add an a_0 element with the value -∞.

   Bellman Equation:

$$OPT[i] = \begin{cases} 0 \text{ if } i = 0 \\ \max\limits_{\substack{j \text{ s.t} \\ 0 \leq j < i \\ a_j \leq a_i}} (OPT[j]) + 1 \end{cases}$$

   However note that the solution won't necessarily be at $OPT[n]$. We need to loop through OPT and return the maximum value.

■