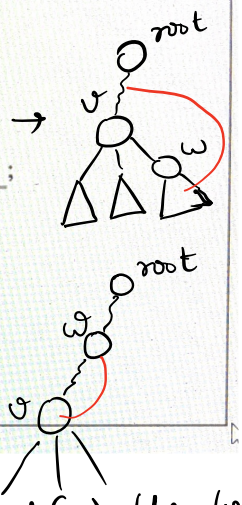
Q1.

```
1  VISITED[v] ← True;
2  HIGH-POINT[v] ← LEVEL[v];
3  foreach w ∈ N(v) do
4      if (VISITED[w] = False) then
5          Set PARENT[w] ← v and LEVEL[w] = 1 + LEVEL[v];
6          Invoke DFS(w) ;
7          HIGH-POINT[v] ← min{HIGH-POINT[w], HIGH-POINT[v]} ;
8          if HIGH-POINT[w] ≥ LEVEL[v] then
9              IS-CUT-VERTEX[v] ← True ;
10         end
11     else if (w ≠ PARENT[v]) then
12         HIGH-POINT[v] ← min{LEVEL[w], HIGH-POINT[v]} ;
13     end
14 end
```

Procedure DFS(v)



If w is parent(v) then (v,w) would be a tree edge.
Else, (v,w) is a back edge & thus imp. for high pt. calculation

```
1  Let (v_1, ..., v_n) be any ordering of vertices of G;
2  for i = 1 to n do
3      VISITED[v_i] ← False and IS-CUT-VERTEX[v_i] ← False ;
4  end
5  for i = 1 to n do
6      if (VISITED[v_i] = False) then
7          LEVEL[v_i] ← 0;
8          Invoke DFS(v_i);
9          if (v_i has one child) then IS-CUT-VERTEX[v_i] ← False ;
10     end
11 end
```

Procedure Compute-Cut-vertices(G)

This will cover all root vertices, and ensure that we set Is-cut-vertex to false if a root has only one child.

Q2.

At most $n-1$ bridge edges:

Recall: Bridge edges are those whose removal increases the connected components of the graph.

Consider a spanning forest F of G.

If an edge $e \notin F$, it cannot be a bridge edge (because its removal does not affect connectivity of the graph)

$\therefore$ Each bridge edge must belong to a spanning forest.

$\Rightarrow$ At most $n-1$ bridge edges.

At most $n-2$ cut-vertices:

Claim: At least 2 non cut-vertices.

Consider a DFS tree T rooted at r, and say L = set of leaf nodes at T.

Fact 1: Leaf nodes are not cut vertices.
Fact 2: Root is not a cut vertex if it has a single child.

If $|L| \geq 2$: L is a set of at least 2 non cut vertices.

If $|L| = 1$: DFS tree T is a path!
$\therefore$ Root has only one child.
$\Rightarrow$ $L \cup \{r\}$ is a set of two non-cut vertices.

□

Q3.

Topological ordering: Linear order of vertices s.t. for every directed edge (x,y) in G, x appears before y in the ordering.

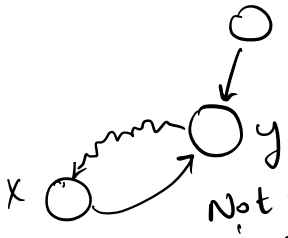
a) (i)

To prove: $FT(y) < FT(x)$

Case 1: y is visited before DFS(x) is invoked:

$\therefore ST(y) < ST(x)$

if $FT(y) > FT(x)$ then y is an ancestor of x. $\Rightarrow$ There is a path from y to x. This is not possible in the DFS tree of a DAG.



$\therefore ST(y) < FT(y) < ST(x) < FT(x)$

Case 2: y is visited after DFS(x) is invoked:

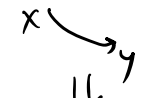
(x,y) must be a forward / tree edge.

Since y is visited inside the call to DFS(x), $ST(x) < ST(y) < FT(y) < FT(x)$ □

(ii) $L = (v_1, v_2, \dots v_n)$

$FT(v_1) > FT(v_2) > \dots > FT(v_n)$

$\Rightarrow$ L is a topological ordering.



$\Downarrow$

$FT(x) > FT(y)$

By part (i) above, each edge can be directed from vertex of lower index to vertex of higher index.

$\Rightarrow$ Topological ordering.

b) A DAG always contains a vertex of in-degree 0.

Take the first vertex v_1 in L. There cannot be any v s.t. (v, v_1) is an edge.

Q4. $A = (a_1, \ldots, a_n)$

Longest increasing subsequence: Each item greater than the previous items in the sequence.

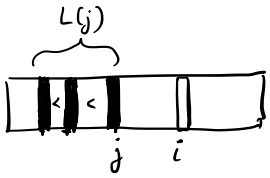
e.g. 10, 22, 9, 33, 21, 50, 41, 60, 80

↓

10, 22, 33, 50, 60, 80

Define $L[i]$ = length of LIS ending at index i (s.t. $A[i]$ is the last element of the LIS).

$$L[i] = \begin{cases} 1 + \max(L[j]) & \forall\, j \in [0, i) \text{ and } A[j] < A[i] \\ 1 & \text{if no such } j \text{ exists} \end{cases}$$



$$\text{LIS of array } A = \max_{0<i<n} (L[i])$$

Notice: overlap when implementing in naive recursive fashion. Exponential complexity.

Dynamic programming - based approach:

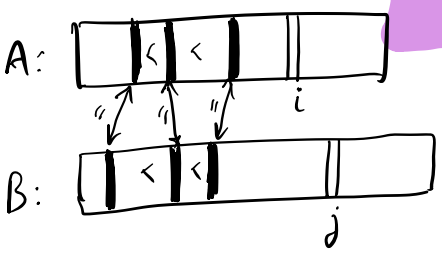
```
for i = 0 to n-1:
    L[i] = 1
    for j = 0 to i-1:
        if (A[i] > A[j] ∧
            L[i] < L[j]+1):
            L[i] = L[j]+1.

Return max(L)
```

$O(n^2)$ complexity.

Q5. Longest common increasing subsequence:

Longest increasing subsequence that is also common to both A and B.

Define $Q[i,j]$:= LCIS of $A[1,\dots,i]$ and $B[1,\dots,j]$, terminating at $B[j]$.

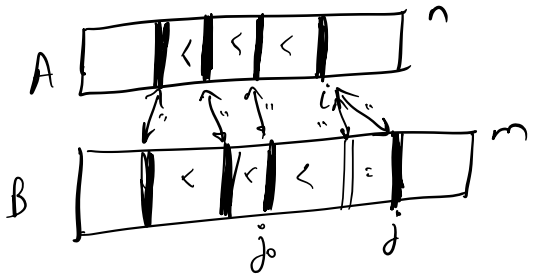


Case I: $Q[i,j]$ doesn't include $A[i]$

In this case $Q[i,j] = Q[i-1,j]$

Case II: $Q[i,j]$ includes $A[i]$

Note: In this case, $B[j] = A[i]$.



$$Q[i,j] = \left(\max_{j_0 : j_0 < j \text{ and } B[j_0] < B[j]} (Q[i-1, j_0]) \right) \text{ concat } A[i]$$

Define $L[i,j]$ = length of $Q[i,j]$

```
Init : for i = 1 to n , j = 1 to n
           set L[i,j] = 0.

For i = 1 to n :
    set j0 = -1
    For j = 1 to m
        set L[i,j] = L[i-1,j]
        if ( B[j0] < B[j] < A[i] or
             (j0 = -1 and L[i-1,j0] < L[i-1,j]) ):
                 j0 = j
        if ( B[j] = A[i] ):
```

```
        set L[i,j] =
              max(L[i,j],
                  1 + L[i-1, j-1])
Output max(L[n,j])
```