1. (4 points) For the directed graph below, find all the strongly connected components and draw the DAG of strongly connected components. Label each strongly connected component with all the nodes it contains.

![Directed Graph](image)

Draw the DAG in the box below:

2. (8 points) Execute DFS on the same graph (reproduced here for convenience) starting at node A and breaking ties alphabetically. Draw the DFS tree/forest. Mark the pre and post values of the nodes with numbering starting from 1.
3. (4 points) In the DFS execution from above, mark the following edges as as T for Tree, F for Forward, B for Back and C for Cross. (No justification necessary)
4. (a) **(4 points)** Draw a strongly connected graph with 6 vertices with the smallest possible number of edges in the box below.

<table>
<thead>
<tr>
<th>Edge</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G \to C$</td>
<td></td>
</tr>
<tr>
<td>$A \to J$</td>
<td></td>
</tr>
<tr>
<td>$B \to A$</td>
<td></td>
</tr>
<tr>
<td>$B \to D$</td>
<td></td>
</tr>
</tbody>
</table>

(b) **(2 points)** In general, the minimum number of edges in a strongly connected directed graph with $n$ vertices is $n$. (no justification necessary)

5. **(6 points)** Suppose $G = (V, E)$ is an undirected graph with positive integer edge weights $\{w_e| e \in E\}$. We would like to find the shortest path between two vertices $s$ and $t$ with an additional requirement: if there are multiple shortest paths, we would like to find one that has the minimum number of edges.

We would like to define new weights $\{w'_e| e \in E\}$ for the edges so that, a single execution of Dijkstra’s algorithm on the graph $G$ with new weights $\{w'_e\}$, starting from $s$ finds the shortest path to $t$ with this additional requirement.

How should we set the new weights $w'_e$?

$$w'_e = \frac{w_e + 1}{|V|}$$

No justification necessary.

6. **(8 points)** Here is an implementation of Bellman-Ford algorithm:

**Input:** Directed Graph $G = (V, E)$, with edge lengths $\{\ell_e| e \in E\}$.

**Output:** Compute distances $\text{dist}(u)$ to each vertex $u$ from a start vertex $s$. 
for $i = 1$ to $n$ do
  $dist(u) \leftarrow \infty$
  $prev(u) \leftarrow \text{nil}$
end for

$k \leftarrow 0$
repeat
  for $i = 1$ to $n$ do
    for each directed edge $(i, j)$ do
      update$(i, j)$
    end for
  end for
  $k \leftarrow k + 1$
until all $dist$ values stop changing OR ($k = n$)

Algorithm 1: Bellman-Ford Algorithm

It turns out that the runtime of the above algorithm can be very sensitive to the way in which vertices in a graph are numbered. In other words, the runtime of the algorithm on the same graph can widely vary, if we change the numbering of the vertices.

Give one graph $G$ on 11 vertices and two ways to label the vertices of $G$, such that in one labelling the algorithm makes 20 calls to update, while in the other labelling the algorithm terminates in $10^2$ calls to update.

7. (6 points) We computed the minimum spanning tree $T$ on a graph $G$ with costs $\{c_e\}_{e \in E}$. Unfortunately, after computing the minimum spanning tree, we discover that the costs of all the edges in the graph have changed as follows: the new cost $w_e$ are given by,

$$w_e = \begin{cases} 
2 \cdot c_e & \text{if } c_e > 100 \\
0 & \text{if } c_e \leq 100 
\end{cases}$$

Is the tree $T$ that we computed earlier, still a minimum spanning tree of the graph?

Write “yes” or “no”:

If yes, prove; if no, disprove with a counterexample.
8. (6 points) In this graph, some of the edge weights are known, while the rest are unknown.

\[ cost(A, D) = 2, cost(B, D) = 1, cost(C, D) = 4, cost(B, E) = 3 \]

List all edges that must belong to a minimum spanning tree, regardless of what the unknown edge weights turn out to be. Justify each of your edges briefly (a sentence or less is enough).

<table>
<thead>
<tr>
<th>Edges that must belong to every MST</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>G - H</td>
<td>This is the only edge that can connect H to any other vertex, so it must be included in any MST. Remember that any MST must span all of the vertices.</td>
</tr>
<tr>
<td>B - D</td>
<td>The cut property. More details: let ( S_1 = D ), and ( S_2 = V \backslash S_1 = ABCEF ). Then three edges from the original graph will connect these two forests, with edge weights: 1, 2, 4. So any MST must contain the B - D edge of weight 1.</td>
</tr>
<tr>
<td>B - E</td>
<td>Also by cut property. Consider ( S_1 = ABD ), and ( S_2 = V \backslash S_1 = CEFGH ). Then the 3 edge is the shortest edge connecting both forests.</td>
</tr>
</tbody>
</table>


9. Design an efficient algorithm for the following problem

**Input:**  $n$ numbers $\{a_1, \ldots, a_n\}$

**Goal:** Compute the polynomial with $a_1, \ldots, a_n$ as its roots. In other words, compute coefficients $b_0, \ldots, b_n$ so that $(x - a_1) \cdot (x - a_2) \cdots (x - a_n) = b_0 + b_1 x + \ldots b_n x^n$.

(Hint: Try divide and conquer & use $O(n \log n)$ time polynomial multiplication algorithm as a blackbox)

(a) **(10 points)** Pseudocode:

(b) **(3 points)** Write the recurrence for the running time of the algorithm in the box.

$$T(n) =$$

(c) **(6 points)** Solve the recurrence to compute the running time and put your answer in the box. Show your work below the box.

$$T(n) =$$
10. (13 points) You are given the road network $G = (V, E)$ of a country, and the lengths $\{\ell_e | e \in E\}$ of each road in the network.

Some of the cities have airports, while others don’t. Let $F$ be the subset of cities that have an airport in them.

Devise an algorithm to compute the distance from each city to the nearest airport. (Assume that the graph is directed and that all edge lengths are non-negative).

Remember every correct algorithm will receive a score depending on its runtime. (can you do it with the same run-time as Dijkstra’s?).

(a) **Main Idea:** (try less than 6 sentences if you can, but don’t fret if you go over)

(b) **Runtime of the algorithm**

(c) **Proof of Correctness** (try less than 4 sentences if you can, but don’t fret if you go over)
11. (10 points) Suppose you are given an array $A[1 \ldots n]$ of sorted integers that has been circularly shifted $k$ positions to the right for some $k$. For example, $[35, 42, -5, 15, 27, 29]$ is a sorted array that has been circularly shifted $k = 2$ positions, while $[27, 29, 35, 42, -5, 15]$ has been shifted $k = 4$ positions. We can obviously find the largest element in $A$ in $O(n)$ time.

Assuming all the integers in the array are distinct, describe an $O(\log n)$ algorithm to find the largest element in $A$.

**Brief but precise description of the algorithm:** (try less than 6 sentences if you can, but don’t fret if you go over)