1. Let $A$ be the set $\{a, e, i, o, u\}$, and consider the relation $R$ on $A$ whose graph is given by the following adjacency matrix:

$$
\begin{array}{ccccc}
& a & e & i & o & u \\
a & 0 & 1 & 1 & 0 & 1 \\
e & 0 & 0 & 1 & 0 & 1 \\
i & 0 & 0 & 0 & 0 & 0 \\
o & 0 & 1 & 1 & 0 & 1 \\
u & 0 & 0 & 1 & 0 & 0 \\
\end{array}
$$

(Recall that the convention is that the cell at row $x$, column $y$ is 1 if $x R y$.)

(a) Draw the graph of $R$:

(b) Either identify a cycle in the graph of $R$, or give a topological ordering of the elements of $A$ according to $R$:

(c) Which of the following properties apply to $R$: reflexive, symmetric, transitive, antisymmetric?
2. Here is an adjacency list representation of a directed graph:

<table>
<thead>
<tr>
<th>Node</th>
<th>Successors</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>(none)</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td>A, B</td>
</tr>
<tr>
<td>E</td>
<td>B, D</td>
</tr>
<tr>
<td>F</td>
<td>D, G</td>
</tr>
<tr>
<td>G</td>
<td>C</td>
</tr>
</tbody>
</table>

(a) Draw the graph.

(b) If the graph is acyclic, give a topological ordering of its nodes; otherwise, identify a cycle.

(c) What is the longest path in the graph that never revisits a node?

by inspection: \( F \rightarrow G \rightarrow C \rightarrow A \rightarrow B \)

(d) Define the distance between two vertices as the length of the shortest path between them, or \( \infty \) if there is no such path. What is the greatest non-infinite distance in this graph?

Can run Floyd’s algorithm, or

by inspection: \( G \) is 3 steps from \( B \)

\( (G \rightarrow C \rightarrow A \rightarrow B) \)

and nothing is further

\( (F \rightarrow B \text{ via } C \) is 4 steps in (c), but there is a 2-step path \( F \rightarrow D \rightarrow B) \)
3. Here is an adjacency matrix representation of an undirected weighted graph (a weight of $\infty$ means there is no edge between those vertices):

\[
\begin{array}{ccccccc}
 & A & B & C & D & E & F & G \\
A & 0 & 5 & 7 & 4 & 6 & 5 & 2 \\
B & 5 & 0 & 1 & \infty & \infty & \infty & \infty \\
C & 7 & 1 & 0 & 8 & \infty & 2 & \infty \\
D & 4 & \infty & 8 & 0 & 1 & \infty & \infty \\
E & 6 & \infty & \infty & 1 & 0 & 1 & \infty \\
F & 5 & \infty & 2 & \infty & 1 & 0 & 1 \\
G & 2 & \infty & \infty & \infty & \infty & 1 & 0 \\
\end{array}
\]

(a) Draw the graph.

(b) Find the (weighted) shortest path from A to E.

Using Dijkstra’s Algorithm:

\[
A \rightarrow G \rightarrow F \rightarrow E, \text{ length } 4
\]

(c) Would the answer to the previous question change if the weight of the edge between B and C were changed to $-1$? Why or why not?

No, it would not change distance from A to E. The shortest distances A to B and A to C would become 4, but that is not enough to improve the A to E cost.

(d) Find a minimum spanning tree for the graph.

Using Prim’s Algorithm:
4. (5 points) Suppose the running time $T(N)$ of some algorithm is given by the following recurrence:

$$
\begin{align*}
T(1) &= 1 \\
T(N) &= T(N - 1) + 2N - 1, \quad (N > 1)
\end{align*}
$$

(a) Fill in the following table of values. For the last entry, give a closed-form expression for $T(N)$, either by solving the recurrence or by guessing:

<table>
<thead>
<tr>
<th>$T(1)$</th>
<th>$T(2)$</th>
<th>$T(3)$</th>
<th>$T(4)$</th>
<th>$T(N)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>9</td>
<td>16</td>
<td>$N^2$</td>
</tr>
</tbody>
</table>

Solving:

$$
T(N) = T(N-1) + 2N - 1
= T(N-2) + 2(N-1) + 2N - 2
= \ldots = T(N-k) + 2((N-k+1) + \ldots + (N-2) + (N-1)) - k
= T(1) + 2(2 + 3 + \ldots + (N-1) + N) - (N-1)
= 2\left(1 + 2 + 3 + \ldots + N\right) - N = (N^2 + N) - N = N^2
$$

(b) Prove by induction that your closed-form expression for $T(N)$ is correct.

Base Case: $T(1) = 1 = 1^2 \checkmark$

Ind. Step: Suppose $T(N) = N^2$ for some $N \geq 1$

Then $T(N+1) = T(N) + 2(N+1) - 1$

$= N^2 + 2N + 2 - 1$

$= (N+1)^2 \checkmark$

So true for all $N \geq 1$.  


5. (12 points) Here is our Scala code for inserting a value in a binary search tree:

```scala
trait Tree
case object Empty extends Tree
case class Node(left: Tree, value: Int, right: Tree) extends Tree

def insert(t: Tree, n: Int): Tree = t match {
  case Empty => Node(Empty, n, Empty)
  case Node(l, v, r) =>
    if (n == v) // No change -- already in tree
      t
    else if (n < v)
      Node(insert(l, n), v, r)
    else // n > v
      Node(l, v, insert(r, n))
}
```

(a) Complete the following skeleton to define a function `insertAll` which takes a tree and a list of numbers and returns a new tree with all of the numbers inserted into the original tree:

```scala
def insertAll(t: Tree, nums: List[Int]): Tree = nums match {
  case Nil =>
  case head :: tail =>
    case head :: tail =>
      insert(insertAll(t, tail), head)
    }

(b) Show the tree which results from evaluating `insertAll(Empty, List(3, 1, 4, 1, 5))`:

```
```

(continued)
(c) Give a tight big-oh upper bound on the average running time of `insertAll` in terms of the size of the list, $N$ (assume that the initial tree is empty, and that the resulting tree is “balanced”):

$$\text{insert into tree with $k$ nodes is } O(\log k) \text{ on average, so insertAll is } O(\log 1 + \log 2 + \ldots + \log (N-1)) = O(N \log N)$$

(d) Here is a version of inorder traversal which returns the visited items in a list (the `:::` operator concatenates two lists; assume for this problem that this can be done in constant time):

```scala
def inorder(t: Tree): List[Int] = t match {
  case Empty => Nil
  case Node(l, v, r) => inorder(l) ::: List(v) ::: inorder(r)
}
```

Now we may define the following function:

```scala
def doSomething(nums: List[Int]): List[Int] = inorder(insertAll(Empty, nums))
```

What is the result of `doSomething(List(3, 1, 4, 1, 5))`?

In order traversal of a BST lists all values in order:

```
List(1, 3, 4, 5)
```

(e) Describe the effect of `doSomething(nums)` on an arbitrary list `nums` of type `List[Int]`:

```
Sorts the list nums, removing duplicates
```

(f) Give a tight big-oh upper bound on the average running time of `doSomething` in terms of the size of its argument, $N$:

$$O\left(\frac{N \log N + N}{\text{insertAll}} \right) = O(N \log N)$$

This is called “tree sort”