Question 2

2 a.)

We start out our algorithm by finding the meta graph of G through the SCC algorithm. We know that the SCC algorithm is linear time with O(n + m). n represents the amount of nodes in G and m represents the amount of edges in G. Lets call this metagraph G'. For every v' in V', weight(v') = the sum of all strongly connected components' weights in its set. This is because if there is a strongly connected component, one could collect all of the eggs wihin this node. Therefore adding all the weights is appropriate.

From here, we run the max weighted path of a DAG algorithm on the node in G' that contains s. From there, we return the max path

Runtime Analysis:

```
The runtime of creating a meta graph is O(n + m) where n = |V| and m =
|E|. Additionally, the longest path of a DAG algorithm is O(n + m) as
well. Therefore, the overall time complexity is linear which is O(n + m)
```

2 b)

We have a very similar approach to 2a. We start our algorithm by finding the meta graph of G. through the SCC algorithm. We know that the SCC algorithm is linear time with O(n + m). n represents the amount of nodes in G and m represents the amount of edges in G. We will call this metagraph G'. Now we have a list data structure (array) that stores all sources in G'. This would take O(n) time. We find the sources of G' by reversing it and finding the sinks. We find the sinks by seeing if each node has no neighbors. Reversing a graph is O(n + m) and traversing each node is O(n). Therefore, this entire process of finding each source is O(n + m). From here, we create a new s' vertex that has a weight of 0 and has an edge to all sources in G'.

Therefore,

```
V' = G(V') U {s'}
```

In addition to the edges G' currently has, G' has the following edges

```
s' -> v' given that v' is a source in the metagraph G'
```

Therefore the edges of G' is E' = G(E') U {{s',v'}} given that v' are sources in G'

From here, we run the longest path algorithm of a DAG which is O(n + m) time where n is the number of nodes in G' and m is the number of edges in G'.

Runtime Analysis:

```
The runtime of creating a meta graph is O(n + m) where n = |V| and m =
|E|. Adding a new node s' to all sources is linear time. Additionally, the
longest path of a DAG algorithm is O(n + m) as well. Therefore, the
overall time complexity is linear which is O(n + m)
```

2 c)

Let EasterEggs(G, k) return the maximum number of eggs in the graph G by traversing at most k locations. We will store the maximum number of eggs in a 2D-array named MaxEggs[v][n], where v is the starting node and n is upper bound on the locations traversed. We are populating the array MaxEggs[v][n] by calling EasterEggs(G, k), where we traverse all the nodes and check all of the possible k locations to from the traversed node's neighbors. This will give us all the possibilities of the number of eggs collected using at most k locations.

EasterEggs(G, k): MaxEggs[v from 0 to n-1][n from 1 to k] = 0 For i from n-1 to 0: For j from 2 to k: For all z neighbors of i: MaxEggs[i][j] = max(MaxEggs[i][j], MaxEggs[z][j-1] + weight(i)) return max(MaxEggs[i][k] for i from 0 to n-1)

```
Since we traverse through all n nodes and all of the neighbors of that
node, and each of these has a runtime of O(n), the runtime of those loops
is O(n^2). Since we are also traversing through all k locations, the total
runtime of the algorithm is O(k*n^2).
```