Assignment 4

Question 1

a)

Let S_i denote the number of swaps performed by the algorithm in the i-th iteration of the for-loop. Then,

$S_1 = \{0 , 1\}$,

$S_2 = \{0, 1, 2\}$,

$S_i = \{0, 1, \dots , i\}$,

$S_{n-1} = \{0, 1, \dots , n - 1\}$.

For any given S_i, there are i + 1 possible number of swaps. Therefore, the probability that one of these number of swaps occurs is 1 / (i + 1). In order to find the expected value of S_i, the sum of the number of possible swaps must be multiplied by the probability that a given number of swaps occurs, which is 1 / (i + 1).

$E[S_1] = (0 + 1) * 1 / (1 + 1) = 1 * 1 / 2 = 1 / 2 = 0.5$

$E[S_2] = (0 + 1 + 2) * 1 / (2 + 1) = 3 * 1 / 3 = 1$

$E[S_i] = (0 + 1 + \dots + i) * 1 / (i + 1)$

$E[S_i] = 1 / (i + 1) * \sum k$, for $k = 0$ to $k = i$

$E[S_i] = 1 / (i + 1) * i(i + 1) / 2$ # ($\sum k$, for $k = 0$ to $k = i$) $= i(i + 1) / 2$

$E[S_i] = i(i + 1) / 2(i + 1)$

$E[S_i] = i / 2$

Therefore, $E[S_i] = i / 2$ and $E[S_{n-1}] = (n - 1) / 2$

b)

S_1 S_2, …, S_{n-1} are all independent variables which means that their expected values are independent.

$S = S_1 + S_2 + \dots + S_{n-1}$

$E[S] = E[S_1 + S_2 + \dots + S_{n-1}]$

$E[S] = E[S_1] + E[S_2] + \dots + E[S_{n-1}]$

$E[S] = 1 / 2 + 2 / 2 + \dots + (n - 1) / 2$

$E[S] = 1 / 2 * (1 + 2 + \dots + n - 1)$

$E[S] = 1 / 2 * \sum k$, for $k = 1$ to $k = n - 1$

$E[S] = 1 / 2 * n(n - 1) / 2$ # ($\sum k$, for $k = 1$ to $k = n - 1$) $= n(n - 1) / 2$

$E[S] = n(n - 1) / 4$

$E[S] = (n^2 - n) / 4$

Therefore, $E[S] = (n^2 - n) / 4$

Question 2

Cycle-Detection(A, n)

```
i = 0
for j = 1 to n
    Make-Set(j)
e = 0
for each edge (u, v) in A
    e++
minimum_edges = e - n + 1
if minimum_edges < 0
    minimum_edges = 0
k = 0
for each edge edge (u, v) in A[minimum_edges + 1 … e]
    if Find-Set(u) != Find-Set(v)
        Union(u, v)
    else
        i = minimum_edges + k
        return i
    k++
i = minimum_edges + 1
return i
```

There are exactly n iterations of the for loop on lines 2-3 so there are exactly n calls to Make-Set on line 3 and Make-Set runs in O(1) time complexity, which means the time complexity of this for loop is O(n). There are exactly m — m being the number of edges in A — iterations of the for loop on lines 5-6 so *e* is incremented by 1 on line 6 m times and each increment has time complexity O(1), which means the time complexity of this loop is O(m). The for loop on lines 11-17 calls Find-Set a maximum of n times and calls Union a maximum of n times. Find-Set runs in O(1) time complexity and Union runs in O(lgn) time complexity when using the weighted union heuristic and the linked list representation. This means this for loop runs in O(n + nlgn) time complexity. Lines 1, 4, 7, 8, 9, 10, 18 and 19 all run in O(1) time complexity. Bringing all of this information together, the entire algorithm runs in O(m + n + nlgn) time complexity, which is asymptotically better than O(mn).

The first for loop creates a set for every vertex in G. The second for loop counts the total number of edges in G. The variable *minimum_edges* represents the minimum number of edges that need to be removed in order for all cycles to be removed from G. This is because a connected graph with n vertices that has more than n - 1 edges has at least one cycle, therefore the minimum number of edges to be removed in order to remove all cycles in a graph is the **total number of edges - the total number of vertices + 1**. If there are no cycles in G to begin with or if e - n +1 < 0 (the graph is not connected), then *minimum_edges* = 0 which means A[*minimum_edges* + 1 … *e*] = A. If there are no cycles in G then *i* will be returned as 1. *e* represents the total number of edges in G, and to remove all possible cycles from G, the algorithm checks the remaining *e* - *minimum_edges* edges to check if there are any cycles remaining. There is a counter (*k*) used to keep track of how many edges have been visited in the third for loop, and once a cycle is found the value of *i* is set to *minimum_edges* + *k* and returned. Overall what this means is that the first *i* - 1 edges are to be removed from G in order ensure there are no cycles remaining. If no cycles are found, then *i* = *minimum_edges* + 1, because *minimum_edges* is the number of edges to be removed, so *i* must be one greater than that number. Therefore, this algorithm clearly returns the smallest integer *i* such that the graph $G_i = (V, E - \{e_1, e_2, \dots, e_{i-1}\})$ has no cycles.