1(a) Written by: Anna Yun Yeen Lieu, Read by: Xingyu Wang

S_i = number of swaps in i^{th} iteration of for loop

While loop: Integer A[j + 1] has (1/i) probability of belonging to any one specific index, where $1 \leq index \leq j+1$ in array A that results in number of swaps $S_i \in \{0, 1, ...i-1\}$. This is because the while loop exits once A[j + 1] is placed into the right index through adjacent swaps. The equal probability of being in any one index is because A is uniformly chosen random permutation of 1...n

For example: If A[j + 1] belongs to index j + 1, then S_i = 0; If A[j + 1] belongs to index j, then S_i = 1; If A[j + 1] belongs to index 1, then S_i = i - 1;

$$\begin{aligned}
E[S_i] =& \sum_{S_i=0}^{i-1} S_i \frac{1}{i} \\
=& \frac{1}{i} \sum_{S_i=0}^{i-1} S_i \\
=& \frac{1}{i} \frac{i(i-1)}{2} \\
=& \frac{(i-1)}{2}
\end{aligned}$$

1(b) $S = S_1 + S_2 + .. + S_n$ = total number of swaps

Expected number of swaps of one iteration i = $E[S_i] = \frac{(i-1)}{2}$ so take the sum of expected iterations over all values of $i \in \{1, 2, ..., n-1, n\}$ to get Expected total number of swaps.

$$\begin{aligned}
E[S] =& E[\sum_{i=1}^{n} S_i] \\
=& \sum_{i=1}^{n} E[S_i] \\
=& \sum_{i=1}^{n} \frac{(i-1)}{2} \\
=& \frac{1}{2} \sum_{i=1}^{n} (i-1) \\
=& \frac{1}{2} [\frac{n(n+1)}{2} - n] \\
=& \frac{n(n+1)}{4} - \frac{n}{2} \\
=& \frac{(n+1)n - 2n}{4} \\
=& \frac{n(n+1-2)}{4} \\
=& \frac{n(n-1)}{4}
\end{aligned}$$

Q.2

Written by Crystal Tung Yip, Read by: Xingyu Wang

```
remove_cycles(A, V):
1     i <- 1
2     for each v ∈ V:
3         MAKE_SET(v)
4     R = {e_m,...,e_1}// A in reverse order
5     for each (u,v) ∈ R:
6         i ++
7         x = FIND_SET(u)
8         y = FIND_SET(v)
9         if x == y: break
10        else: UNION(x, y)
11    return total number of edges - i - 1
```

This algorithm uses the FIND_COMPONENTS pseudocode covered in lecture. To detect a cycle in the process of making sets and joining vertices we do the following: we know that while unionizing two vertices of a graph, if x == y holds true on line 9, i.e. the two vertices are already in the same set, then we know we have a cycle. Since we executed this process in the reverse order of A, we know that the first cycle we detect in this loop is the last edge we want to remove such that the graph has no more cycles. This algorithm uses a sequence of n unions and m > n finds. As proved in lecture, this costs O(m + nlogn).

Correctness:

Loop Invariant - After each iteration of the loop, there are no cycles in the current graph.

Base Case:

On entering the loop, the graph G has no nodes. I.e. Thus, there are no cycles.

Inductive Step:

Consider the ith iteration of the loop.

2 Cases:

1. x == y and line 9 holds
   The current graph, G_{i-1} has no graphs [IH]
   Since there are no unions, after executing the condition in line 9, the graph after the ith iteration has no cycles
2. x != y and line 10 holds
   Since x and y are not in the same set, unionizing x and y will not create a cycle in graph G_i The loop in FIND_COMPONENTS exits under 2 conditions: a cycle is

detected or all edges are unionized in the reverse order of A edges. We know that if a loop exits because a cycle is detected, it should be the last edge to be removed in the order of A.