Assignment 3
John Abo
30088517

1. First, we will let $X_{i,j}$ be the number of times $A[i] > A[j]$ in $i < j$, we can get

$$E\left[\sum_{i<j} X_{i,j}\right] = \sum_{i<j} E\left[X_{i,j}\right]$$

We can simplify it into

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr(A[i] > A[j])$$

Where the inner sum includes all element in $A[2..n]$ as it'll be starting with the element to the right of $A[i]$, where the other loop starts at 1. Thus, the outer sum goes to $n-1$, as there is no element beyond the position n. The probability that $A[i] > A[j]$ gives us

$$= \frac{1}{2}\sum_{i=1}^{n-1} n - i$$
$$= \frac{1}{2}\left(\sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i\right)$$
$$= \frac{1}{2}\left(n(n-1) - \frac{n(n-1)}{2}\right)$$
$$= \frac{n(n-1)}{2} - \frac{n(n-1)}{4}$$
$$= \frac{n(n-1)}{4}$$

2.

a. $Path(m,n)$:
Input: m and n are the dimensions of the environment that will be traversed

Create array $A[1..m+n]$, that will hold the path that will be taken. It will contain either a move $west$ or $south$.

//These insert all the possible moves into the path array
$i = 1;$
$j = 1;$

While $i < m$ do
    $A[j] = south;$
    $j = j + 1;$
    $i = i + 1;$
$i = 1;$
While $i < n$ do
    $A[j] = west;$
    $j = j + 1;$
    $i = i + 1;$

//randomly permutes the new array
$Permute - in - place(A)$;

Return A;

The first while loop runs in $O(m)$ while the other runs in $O(n)$, then $Permute - in - place(A)$ runs in $O(m+n)$ (as seen in lecture 7, Probabilistic Analysis). This means the algorithm will run in $O(m+n)$ time.

b. $TargettedPath(m,n,i,j)$:
Input: m and n are the dimensions of the environment, i and j are a point in the environment such that $1 < i < m$ and $1 < j < n$

//Create array $A[1..m+n]$ that will hold the path generated

//Creates array B that will hold the path from start to (i,j)
$B = Path(i,j)$;

//Creates array C that will hold the path from (i,j) to end of total path
$C = Path(m-i, n-j)$;

Append B and C into array A

Return A

This algorithm works by generating 2 sets of paths, such that they share the point (i,j). The first path can be any possible path from start to (i,j), as it effectively creates a sub-environment that it will generate a path for. The second call creates a second sub-environment that is the remaining distance from (i,j) to the end.

The running time of this algorithm is based on the running time of path. As it is called twice in succession, it will be the sum of both running times:

$O(m-i+n-j)$ and $O(i+j)$

Which gives us that the total running time is $O(m+n)$ again.

3. Deterministic quicksort's (where the last element of each subarray is chosen as the pivot) worse case is a fully sorted array. Performance will only get worse the more sorted the input becomes. As the last element is chosen each time, the more likely it will be that the last element is the largest element in the subarray, leading to the pivot step moving all elements to one side. Formally, the inputs will be *k-sorted*, where k is the number of positions each element is from their sorted position. With quicksort's partition creating partitions of size $n-k$ and k, such that the partitions are unbalanced, leading to $O\big(n(n-k)\big)$ or $O(n^2)$ performance. While insertion sort's insertion loop will only need to run k times for a *k-sorted* array, as the key is only k away from where it will be inserted, leading to $O(nk)$ or $O(n)$ performance.

4. $Letters(letter, magazine)$:

   Create hash table $table$ that is implemented using chaining and accounts for k unique hashes, such that k is the number of possible characters in $letter$ and $magazine$.

   For each $character$ in $letter$ do:
       $table.insert(character)$
   For each $character$ in $magazine$ do:
       If $table.search(character)$ then
           $table.delete(character)$
       Else
           Return false
   Return true

   The pseudocode inserts all characters from $letter$ into a hash table what uses chaining, this allows the characters and number of characters to be kept track of as magazine is searched. Every time a character is found, the linked list that makes up a chain in the table are reduced by one element, thus reducing the counter. $search$ and $insert$ run in constant time on average (as seen in lectures). This gives us the following runtime analysis:

   First for loop runs in $O(n)$, second for loop runs in $O(m)$, this leads to a total running time of $O(n + m)$.

5. Code is included with submission