CS200 - Final Review

1. Let $T(n)$ be the number of strings in $\{0,1,2\}^n$ that do *not* contain two consecutive zeros. Write a recurrence relation for $T(n)$

2. Let $[n] = \{1,2,3,\ldots,n\}$. Given a permutation of the elements of $[n]$, an inversion is an ordered pair (i,j) with $i,j \in [n]$, such that $i < j$, but j precedes i in the permutation. For instance, consider the set $[5]$, and the permutation $(3,5,1,4,2)$ - then there are six inversions in this permutation:

$$(1,3),(1,5),(2,3),(2,4),(2,5),(4,5). \tag{1}$$

If a permutation is chosen uniformly at random from among all permutations, what is the expected number of inversions? Use our 5 step process:

   (a) What is the sample space and what is the random variable that we care about?

   (b) Break up the main random variable into a weighted sum of indicator random variables.

   (c) Use linearity of expectation

   (d) Use property of expected value of indicator random variables.

   (e) Add up the terms in the sum to get the final answer

(Hint - given, for example, $\{2,6\} \in [8]$, is it more likely to get a permuation where 2 is before 6, or 6 is before 2?)

3. In the following, you may assume that the graph $G = (V,E)$ is undirected and does not have self loops or multi-edges. Let $deg(v)$ be the degree of a vertex v.

   (a) **[3 points]** $D(v,u,(V,E)) \equiv$ In the graph (V,E) there is a path of length 2 from vertex v to vertex u.

   (b) **[3 points]** $R(v,(V,E)) \equiv v$ is the vertex with the smallest degree in the graph (V,E)

   (c) **[3 points]** $W(V,E) \equiv$ There is a vertex in the graph (V,E) that is not connected to any other vertices.

   (d) **[3 points]** $M(V,E) \equiv$ There is a vertex in the graph (V,E) that is connected to all other vertices.

   (e) **[3 points]** $T(V,E) \equiv$ All vertices in the graph (V,E) have the same degree.

   (f) **[3 points]** $K(V,E) \equiv$ All vertices in the graph (V,E) have even degree.

4. What is a recurrence relation for this algorithm? Evaluate the recurrence relation using iterative method, and if possible, master method.

Algorithm 1: `MergeSort`(C, n)

Input : Array of C of length n (where n is a power of 2)
Output: Sorted array containing all elements of C

```
if n==1 then
    return C;
end
A=MergeSort(C[1 : n/2], n/2);
B=MergeSort(C[n/2 + 1 : n/2], n/2);
p_A = 1;
p_B = 1;
Increase length of A and B by 1 each, and set final element of each array to ∞;
for k = 1 to n do
    if A[p_A] < B[p_B] then
        C[k] = A[p_A];
        p_A += 1;
    else
        C[k] = B[p_B];
        p_B += 1;
    end
end
```