Answer the questions below. For the questions involving both programming and analysis, please submit your code through Udacity and your analysis through T-square.

1. In a list of distinct numbers $a_1, \dots a_n$, we say that the elements a_i and a_j are *inverted* if $i < j$ but $a_i > a_j$. Suppose that all orderings of $a_1, \dots a_n$are equally probable under some probability distribution (In other words we shuffled a set of numbers perfectly to obtain the order $a_1 \dots a_n$.) What is the expected number of inversions?

Let E_{ij} be the event that $i < j$ and $a_i > a_j$. Let random variable X_{ij} have the following definition:

$$X_{ij} = \begin{cases} 1 & \text{if (i,j)is an inversion} \\ 0 & \text{if (i,j)is not and inversion} \end{cases}$$

Then let X equal the number of inversions. In equation form:

$$X = \sum_{i=1}^{n} \sum_{j=1}^{n} X_{ij}$$

The expected number of inversions then is

$$E[X] = E\left[\sum_{i=1}^{n} \sum_{j=1}^{n} X_{ij}\right]$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{n} E[X_{ij}]$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{n} \Pr(E_{ij})$$

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Pr(E_{ij})$$

because i < j

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

The last line demands some explanation. The probability is 0.5 because we have n choose 2 combinations and there are n*(n-1) permutations. The probability that there is an inversion is the ratio of these two quantities. By reducing ratio of these two quantities we arrive at 0.5. The inner sum is reduced by summing 0.5 n-i times. We arrive at the final answer by applying the fact that the sum of the first n natural numbers is n*(n+1)/2. In this case we are summing the integers from 1 to n-1 with the ½ on the outside of the sum, and so we arrive at the final result.

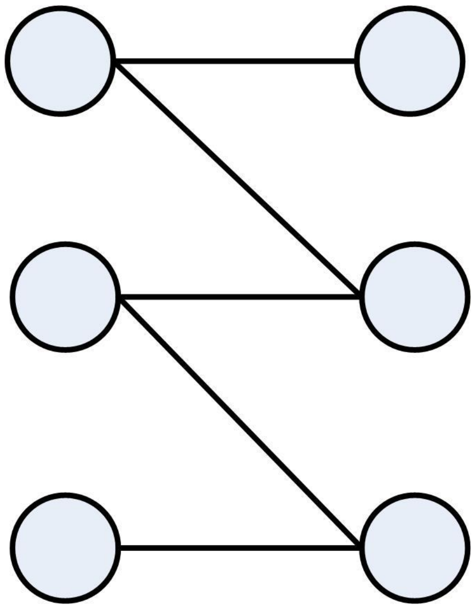
2. Consider the following greedy algorithm for finding a matching.
   - Initialize S to an empty set of edges
   - While there is an edge where *both* vertices are unmatched by S
     - Add the above edge to S
   - Return S

   a. What is the running time of this algorithm?

This is very similar to the approximate vertex cover algorithm covered in lecture. Assuming that the graph is defined as an adjacency list, this algorithm would be order O(|E| |V|). The outer loop would go through each of the edges, so |E| phases. Each phase would go through the set of vertices to determined which are matched or unmatched.

   b. Give an example of a graph where the algorithm is not guaranteed to find the maximum matching.

See the graph below. In this case, there is one vertex that has only one match. That match however can be paired with other vertices. In this case, the algorithm matches the vertices such that the vertex with the lone match goes unpaired. In other words, it matches the vertices using the diagonal edges.



   c. Show that the matching found by the algorithm always has at least half as many edges as a maximum matching.

This is similar to the proof below for greedy weighted matching. Let M* be the maximum matching and M be the maximal matching returned by the algorithm. For any edge e ∊ M*-M (the edges in M* that are not in M), there is a reason that the e was not included in a maximal matching. Namely, a previously edge, call it f(e), was selected to be in the matching and it shared a vertex with e. For any edge f in the maximal matching, this conflict can happen at most two times, one time for each edge. Thus we have

$$|M| \leq |M^*| \leq 2|M|$$

d. (Bonus) Now consider a similar algorithm for finding a maximum weight matching in an edge-weighted graph: Greedily add the heaviest edge possible to the current matching; stop when no further edge can be added. Show that this algorithm finds a matching whose weight is at least half the optimum.

Let M* be the maximum weighted matching and M be the maximal weighted matching produced by the greedy algorithm. For any edge e ∊ M* - M (the edges that are in M* and not in M) there is a reason e did not make it into the greedy matching. Namely, a previously considered edge, call it f(e), had a higher weight and shared a vertex with e. For any edge f in the maximal matching, there can be f(e) for at most two edges e, because they can conflict with edge f at either end. So we have:

$$\sum_{e \in M^*} w_e \geq \sum_{e \in M^*} w_{f(e)} \geq \sum_{f \in M} 2w_f$$

3. Consider the following factor 2 approximation algorithm for the minimum cardinality vertex cover problem. Find a depth first search tree in the given graph, G, and output the set, say S, of all the non-leaf vertices of this tree. Show that S is indeed a vertex cover for G and $|S| \leq 2 \cdot OPT$.

   Hint: Use the tree to construct a matching that includes all the vertices in S.

First to show that S is a vertex cover. A vertex cover is a set of vertices such that all the edges in the graph are incident to at least one of the vertices in the cover. In this case, the vertex cover includes all the vertices in the graph except for those at the leaves of the depth-first search tree. Thus, all the edges in the graph are incident to at least one vertex. The edges connecting the leaves to the rest of the tree have a vertex in the leaf-parents.

Now to show that |S| ≤ 2|OPT|. If we use the depth first search tree to construct a matching M, we can construct this matching to include all the vertices of S. (The matching should be done in a DFS manner as well to ensure that all the vertices of S are included in the matching.) Thus we have

$$|S| \leq 2|M|$$

because each matching has two vertices and to include all the vertices in S the matching will extend beyond S into the leaves of the DFS tree. We also have that the size of the matching M must be less than or equal to the size of the minimum vertex cover. As stated in lecture, the edges of M are vertex disjoint, so any vertex cover must include at least one vertex from each matching. So we have

$$|M| \leq |OPT|$$

as well as

$$2|M| \leq 2|OPT|$$

Putting this all together gives

$$|S| \leq 2|M| \leq 2|OPT|$$

4. The following is known as the maximum acyclic subgraph problem. Given a directed graph $G = (V, E)$, pick a maximum cardinality set of edges from E so that the resulting subgraph is acyclic. Implement an algorithm that gives a $1/2$ factor approximation here.

   https://www.udacity.com/course/viewer#!/c-ud557/l-1209378918/m-3519618629

   Hint: Rather than thinking about picking$1/2$ the optimum, think about picking $1/2$ of the edges.

5. In the lesson, we gave a randomized algorithm for finding a minimum cut set in graph. Now, we consider the problem of finding a *maximum* cut set (all weights on edges are all one).
   a. Consider an algorithm that partitions the vertices into two sets A and $B$by placing each vertex uniformly and independently at random into one of the two sets. What is the expected value of $C(A, B)$, i.e. the number of edges crossing the cut?

Let event $X_i = 1$ if edge i is in the cut set C(A,B) and $X_i = 0$ otherwise. Then

$$E[X] = \sum v\Pr(X = v)$$

By linearity, we have

$$= E\left[\sum_{i=1}^{|E|} X_i\right] = \sum_{i=1}^{|E|} E[X_i] = \sum_{i=1}^{|E|} 1 * \Pr(X_i)$$

Since the vertices are selected uniformly and independently at random, the probability that an edge is in the cut is 1/|E|. So we have

$$= \frac{1}{|E|}\sum_{i=1}^{|E|} 1 = \frac{1}{|E|}\frac{|E|(|E|+1)}{2}$$

$$= \frac{|E|+1}{2}$$

I know that I am off by ½ here (it should just be |E|/2), but I have looked at my proof and I don't know what I am doing wrong.

b. Use the method of conditional expectation to derandomize the above algorithm so that it always achieves this expected value. Implement your algorithm here.

https://www.udacity.com/course/viewer#!/c-ud557/l-1209378918/m-3481888754

c. Explain why your algorithm is correct. Use the notation $x_k = A$ to describe the event that vertex k is assigned to the set A in the partition. Use Y to denote the number of cut edges, and let $P: \{1, \ldots n\} \to \{A, B\}$ represent the partition returned by the derandomized algorithm.

We want to show that the invariant of E[Y] ≥ |E|/2 is maintained. Assume that the first t vertices have been colored. We will call this set S_t. For those edges where one vertices is in set A and one vertex is in set B, that edge is definitely in the cut. We will call this $c(S_t)$. For those edges where both vertices are both in one set, they are definitely not in the cut. For the remaining vertices, the algorithm determines which assignment would maximize the conditional expectation of Y. We will call the set of undetermined vertices set $u(S_t)$.

To select which set to put the t+1 vertex in, we look at the expectation if we assign the vertex to either set:

$$E[Y|S_{t+1}] = c(S_{t+1}) + \frac{u(S_{t+1})}{2}$$

The last term is divided by two because the undetermined edges have a 0.5 probability of belonging to either set. As such, the edges that are incident to them have a 0.5 probability of being in the cut set. One of the choices (putting vertex t+1 in A or B) will result in a conditional expectation that is no less than the previous conditional expectation:

$$E[Y|S_{t+1}] \geq E[Y|S_t]$$

Since the invariant proved in part (a) is maintained (just like it was with the 3 CNF example in lecture), we have that E[Y] is at least half the number of edges, or

$$P: \{1, \ldots n\} \to \{A, B\} \geq \frac{|E|}{2}$$