Written Part

1. How many ways are there to arrange the letters in the word "**`probabilistic`**"?
2. With reference to the previous problem, how many ways if all the **`b`**'s have to precede all the **`i`**'s?
3. How many ways are there to put 100 (unlabelled) balls into 50 labelled boxes?
4. We computed the "mean time to failure" to be $1/p$ if the probability of failure is p. What is the mean time to the second failure? In a sense, the answer is obvious, but prove it from first principles. That is, write down an expression for q_k, the probability that the second failure occurs on the k-th trial, and then compute and simplify the weighted sum $\sum kq_k$. HINT: Recall that in class we computed the mean time to failure by determining p_k, the probability that the first failure occurs on the k-th trial to be the quantity in Definition 19.4.6 of the MIT notes. We then computed simplified $\sum kp_k$, using the formula $(1-x)^{-2} = 1 + 2x + 3x^2 + \cdots$ which we obtained by differentiating $(1-x)^{-1} = 1 + x + x^2 + \cdots$.
5. What number do you get if you subtract the binomial coefficients $\binom{n}{k}$ with an even k from those with an odd k, where n is fixed?
6. What difference of binomial coefficients is equal to the sum

$$\binom{12}{5} + \binom{11}{5} + \binom{10}{5} + \binom{9}{5} + \binom{8}{5}\ ?$$

7. Imagine a maze created in a m by n grid. Assume that there is a *unique* path from any cell to any other cell. What is the total length of the walls in the maze as a function of m and n? For example, below is a 2 by 2 grid with the required path property and the total length of walls is 9. Explain your answer.

```
+---+---+
|       |
+---+   +
|       |
+---+---+
```

Programming Part

Your task here is to determine minimum spanning trees that do and do not contain specified edges. The program is to be loosely based on the algs4.jar classes `Edge`, `EdgeWeightedGraph`, and `KruskalMST`. You should write corresponding classes `MyEdge`, `MyEdgeWeightedGraph`, and `MyKruskal`.

For a graph G containing edge e, we use $mst(G+e)$ to denote the minimum spanning tree that contains edge e and $mst(G-e)$ to denote the minimum spanning tree that does not contain the edge e. We further define

$$include(G) = \sum_{e \in G} mst(G+e) \text{ and } exclude(G) = \sum_{e \in G} mst(G-e).$$

Your program is to compute $include(G)$ and $exclude(G)$ (returning -99 if the graph is or becomes disconnected).

Instead of doubles, the weights in the graphs will be longs. You may assume that all weights are in the range $0 \leq weight < 1000$ and that there are at most 10,000 vertices in the graph. You may also assume that no edge is input twice. It is possible that there are ties in the weights and such ties *must* be resolved by taking lexicographic order of edges (v,w) (that is (v,w) is less than (v',w') if $v < v'$, or if $v = v'$ and $w < w'$.

```
class MyEdge implements Comparable<MyEdge>{
   private final int v; // NOTE: ensure v < w.
   private final int w;
   private long weight;

   public int minv() { return v; }
   public int maxv() { return w; }
   public long weight() { return weight; }
   public void changeWeight( long weight ) { this.weight = weight; }

   MyEdge ( int v, int w, long weight ) {
      this.v = v < w ? v : w;
      this.w = v < w ? w : v;
      this.weight = weight;
   }

   public String toString() {
      return String.format("%d-%d %d", v, w, weight);
   }

   public int compareTo(MyEdge that) {
      // FOR YOU TO FILL IN
   }

}
```

Since we are using Kruskal's algorithm, the graphs will be represented as a Bag of edges. The constructor reads the graph from a file using the same format as in algs4.jar (but with the weight as a long).

```
class MyEdgeWeightedGraph {
    private final int V;
    private final int E;
    private Bag<MyEdge> edges;

    public int V() { return V; }
    public int E() { return E; }

    public Iterable<MyEdge> edges() { return edges; }

    public MyEdgeWeightedGraph( In in ) {
        // FOR YOU TO FILL IN
    }
}
```

An outline of `MyKruskal` is found below. Like `KruskalMST` you must use the class `MinPQ` and `UF`. Feel free to use code directly from `KruskalMST` but you are encouraged to initialize the priority queue with an array instead of repeated inserts so that the priority queue is built in time $O(E)$ time instead of $O(E \log E)$ (see the constructors for `MinPQ`).

```
class MyKruskal {

   private long weight;                              // weight of MST
   private Queue<MyEdge> mst = new Queue<MyEdge>();  // edges in MST

   public long weight() { return weight; }

   public Iterable<MyEdge> edges() { return mst; }

   public MyKruskal( MyEdgeWeightedGraph G ) {
      // FOR YOU TO FILL IN
   }

   public static long include( MyEdgeWeightedGraph G ) {
      // FOR YOU TO FILL IN
   }

   public static long exclude( MyEdgeWeightedGraph G ) {
      // FOR YOU TO FILL IN
   }


}
```

For example an input file containing

```
4
5
0 1 999
1 2 10
2 3 99
3 0 11
0 2 3
```

represent a weighted graph for which $include(G) = 1197$ and $exclude(G) = 1293$. Those should be the two numbers output by the following main.

```
    public static void main( String[] args ) {

        In in = new In( args[0] );
        MyEdgeWeightedGraph G = new MyEdgeWeightedGraph( in );

        MyKruskal mst = new MyKruskal( G );
        System.out.println( MyKruskal.include( G ) );
        System.out.println( MyKruskal.exclude( G ) );
    }
```