Algorithm expErdosRenyi(integer n, double p) {  
  
1 double expectation=0.0;  
2  
3 double[] factorials = new double[n+1];  
4 factorials[0] = 1;  
5 for (int i=1; i<n+1; i++) {  
6 factorials[i] = i\*factorials[i-1];  
7 }  
8  
9  
10 double expectCoeff = Math.*pow*((1-p),(factorials[n]/(factorials[n-2]\*factorials[2])));

11  
12 double[] robStan = new double[n+1];  
13   
14 double[] powers = new double[n\*n+1];  
15 for (int pwr=0; pwr<n\*n+1; pwr++) {  
16 powers[pwr] = Math.*pow*(1/(1-p),pwr);  
17 }  
18  
19 double[] powersMinus1 = new double[n+2];  
20 for (int minus=0; minus<n+2; minus++) {  
21 powersMinus1[minus] = Math.*pow*(-1,minus);  
22 }  
23  
24 robStan[0] = 1;  
25 robStan[1] = 1;  
26  
27 for (int k=2; k<n+1; k++) {  
28 double temp=0.0;  
29 double thing=0.0;  
30 for (int i=1; i<k+1; i++) {  
31   
32 temp = powersMinus1[i+1]

33 \*(factorials[k]/(factorials[k-i]

34 \*factorials[i]))\*powers[i\*k]/powers[i\*i]\*robStan[k-i];  
35 thing = thing + temp;  
36 }  
37 robStan[k]=thing;  
38 }  
39  
40 expectation=expectCoeff\*robStan[n];  
41 return expectation;

Task 5:

**Design**

Computing binomial coefficients can take upto O() time if implemented naively, (such an example is included in acyclic.java as binom. When placed inside a loop would give us very bad run time and make the time complexity multinomial. To compute binomial coefficients we dynamically compute factorials and then use the factorials later inside the loop by the formula

Similarly for calculating powers of we pre-compute powers upto because is always less than or equal to and then access it in the nested loop later on to improve on performance.

**Asymptotic Time complexity**

Line 1-4 – Takes constant time.

Line 4-7 – Takes O() time. (presence of a loop which on each iteration does a multiplication)

Line 10 – Takes O() time as its using the Math.pow() function.

Line 11-14 – Takes constant time again.

Line 15-17 – Takes O() time. (presence of a loop which on each iteration takes a power which takes lgn time, hence O().

Line 19 – Takes constant time.

Line 20-22 – Takes O() time. (presence of a loop which on each iteration takes a power which takes lgn time, hence O().

Line 23-25 – Takes constant time.

Line 27-38 – Presence of a nested for loop makes the running time of this segment O().

Adding all of the above (excluding the constant time since we only need the asymptotic time complexity)

We get runtime = O() + O() + O() + O() + O(),

Which is equal to O().