Q1. Print all the permutations of a string.

public class Recursion3 {

   public static void printPermutation(String str, int idx, String perm) {

       if(str.length() == 0) {

           System.out.println(perm);

           return;

       }

       for(int i=0; i<str.length(); i++) {

           char currChar = str.charAt(i);

           String newStr = str.substring(0, i) + str.substring(i+1);

           printPermutation(newStr, idx+1, perm+currChar);

       }

   }

   public static void main(String args[]) {

       String str = "abc";

       printPermutation(str, 0, "");

   }

}

Time complexity - O(n\*n!)

Q2. CountPathMaze

public class Recursion3 {

   public static int countPaths(int i, int j, int m, int n) {

       if(i == m-1 || j == n-1) {

           return 1;

       }

       return countPaths(i+1, j, m, n) + countPaths(i, j+1, m, n);

   }

   public static void main(String args[]) {

       int m = 4, n = 5;

       System.out.println(countPaths(0, 0, m, n));

   }

}

Time complexity - O(2^(m+n))

Q3. Tiling problem

public class Recursion3 {

   public static int placeTiles(int n, int m) {

       if(n < m) {

           return 1;

       } else if(n == m) {

           return 2;

       }

       return placeTiles(n-1, m) + placeTiles(n-m, m);

   }

   public static void main(String args[]) {

       int n = 4, m = 4;

       System.out.println(placeTiles(n, m));

   }

}

Q4. Friends pairing problem

public class Recursion3 {

   public static int pairFriends(int n) {

      if(n <= 1) {

          return 1;

      }

       return pairFriends(n-1) + (n-1) \* pairFriends(n-2);

   }

   public static void main(String args[]) {

       int n = 3;

       System.out.println(pairFriends(n));

   }

}