Problem: Generate N random number using geometric distribution function given that probability is 0.5. Show the curve of probability mass function (pmf) vs x and frequency / N vs x.

Code:

```
/*
Method:
--------
1. For values of x we keep generating pdf and cdf. At some point cdf stop changing.
   This last value of "x" is stored in "idx" variable. Both pdf and cdf stored in
   "theoretical_pdf" and "theorectical_cdf vector".

2. For every n trials, we generate a random number and a total of n random numbers.
   We store those value in "random_prob" vector.

3. Next we find upper_bound of every "random_prob" index from "theoretical_cdf".
   Find the corresponding "x" for the upper_bound value and update the Frequency of
   "x" in "freq" array.

4. Print the theoretical pmf value and plot the graph in excel sheet.

5. Print the freq / n values and plot the graph in excel sheet.
*/

#include <bits/stdc++.h>

using namespace std;
const double eps = 0.000001;

int n; // how many random numbers we have generate
int idx; // store the value where pmf becomes zero
int freq[111]; // store the frequencey of x in generated random number
vector <double> random_prob; // store n generted random probability
vector <int> random_number; // store the n generated random number
vector <double> theoretical_cdf; // store the cdf of geometric distribution
vector <double> theoretical_pmf; // store the pmf of geometric distribution

// this function calculate geometric cdf using P(x) = (1 - (1 - p) ^ (x + 1))
// here p = 0.5 so P(x) = (1 - 0.5 ^ (x + 1))
double geo_cdf(double x)
{
    return (1.0 - pow(0.5, x + 1.0));
}

// this function calculate geometric pmf using p(x) = ((1 - p) ^ x) * p
// here p = 0.5 so p(x) = (0.5 ^ x) * 0.5
double geo_pmf(double x)
{
    return (pow(.5, x) * .5);
}

// this function generates where pmf of a particular x becomes zero
// and return that x
int converged_index()
{
    double x = 0.0;
```

```
    while ((1.0 - geo_cdf(x)) >= eps)
    {
        theoretical_cdf.push_back(geo_cdf(x));
        theoretical_pmf.push_back(geo_pmf(x));
        x = x + 1.0;
    }

    return (int)x;
}

// this function generates n random probability
void generate_random_prob()
{
    srand(time(NULL));

    for (int i = 0; i < n; ++i)
        random_prob.push_back((double) rand() / RAND_MAX);
}

// this function generates the n random number
// using the the upper bound from cdf we generated earlier
void generate_random_number()
{
    //we go through every n random probability
    for (int i = 0; i < random_prob.size(); ++i)
    {
        int val = 0;

        //find the upper bound of a random probability
        for (int j = 0; j < theoretical_cdf.size(); ++j)
        {
            //if we find the upper bound then break
            if (theoretical_cdf[j] - random_prob[i] >= eps)
            {
                val = j;
                break;
            }
        }

        // increase the frequency by one
        ++freq[val];
        random_number.push_back(val);
    }
}

int main()
{
    cin >> n;

    idx = converged_index();
    generate_random_prob();
    generate_random_number();

    //print pmf of every x

    for (int i = 0; i < idx; ++i)
        cout << theoretical_pmf[i] << endl;
```

```
    //print freq / n for evey x

    for (int i = 0; i < idx; ++i)
        cout << (double) (freq[i] / (double) n) << endl;

    return 0;
}
```

Plot:

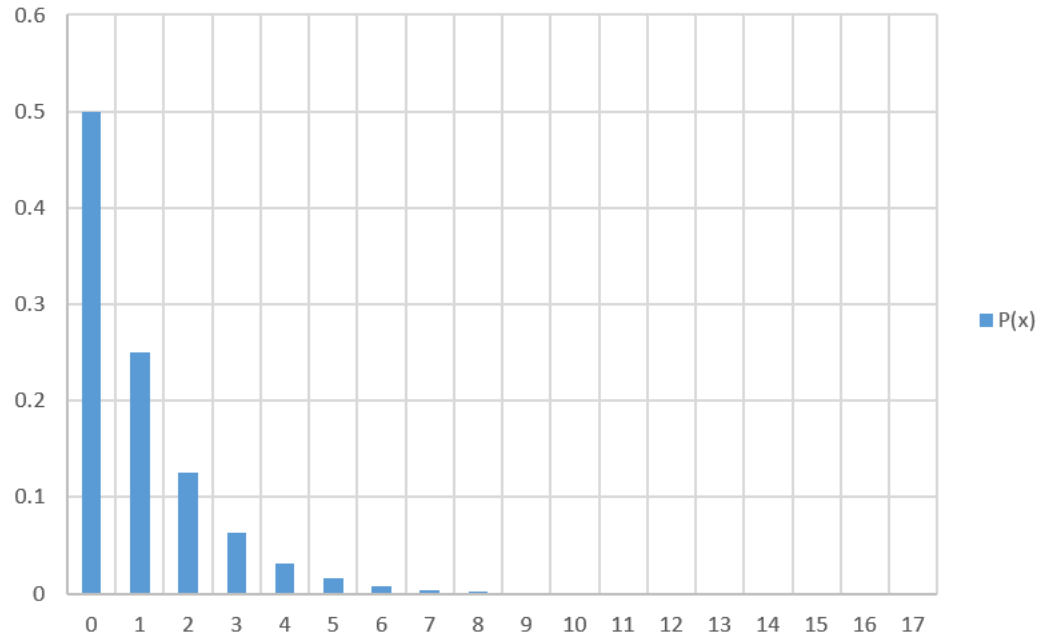

Fig: Plot of p(x) vs x

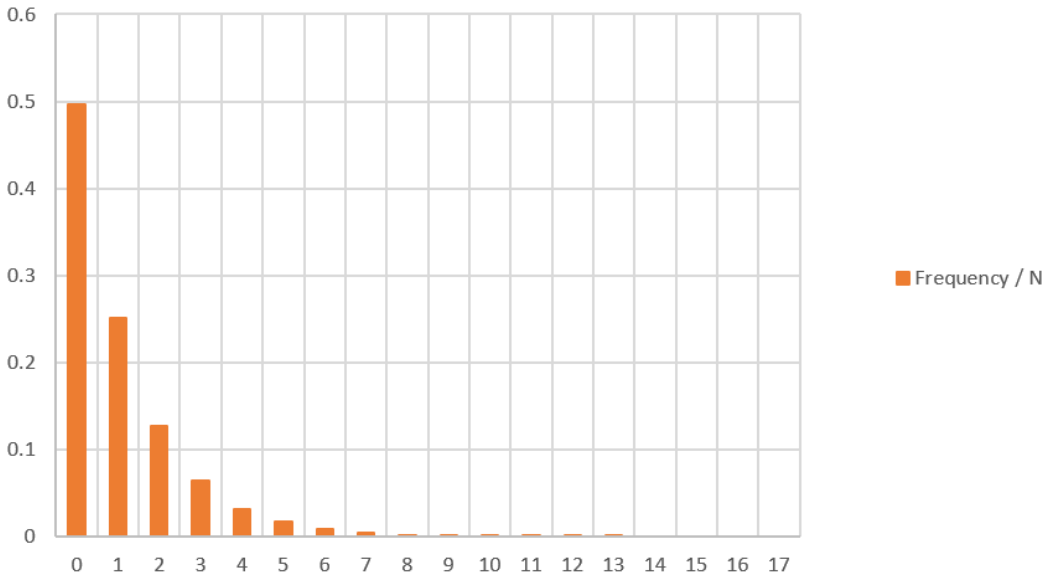

Fig: Plot of Freq / N vs x

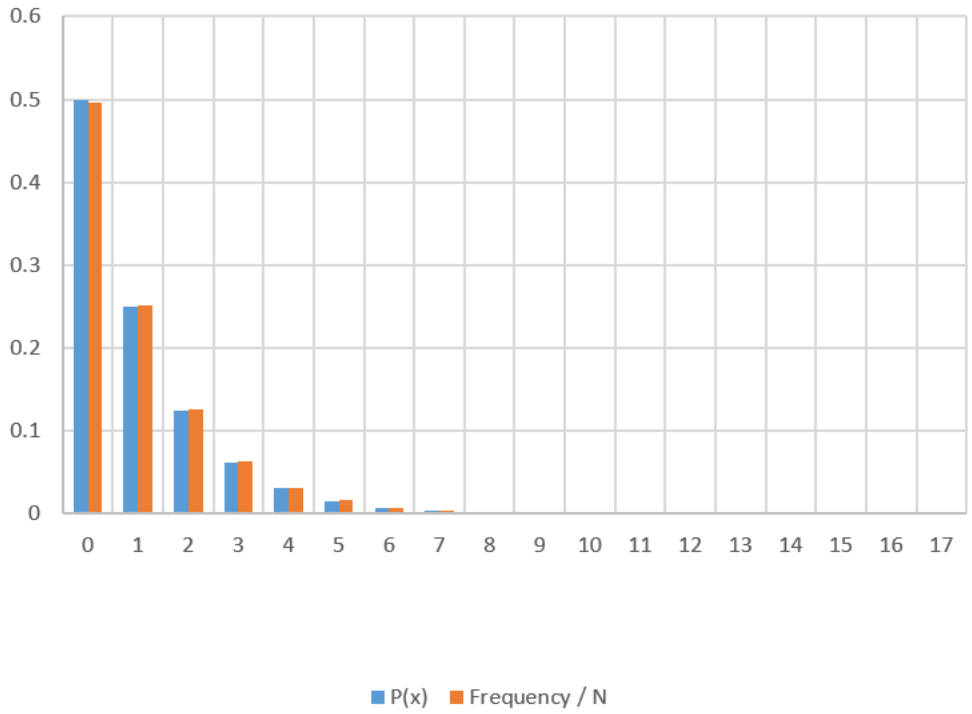

Fig: p(x) and frequency / n in same graph