Homework 8

The exercises do *not* require long writings. Try to be precise and to the point. Your answers should be short.

1. Suppose we have proved the following fact: Let A, B, C be $n \times n$ matrices, and $AB \neq C$. If we choose a vector $\boldsymbol{r} \in \{0,1\}^n$ uniformly at random, then the probability

$$\Pr(\{AB\boldsymbol{r} = C\boldsymbol{r}\}) \leq 1/2.$$

Using this fact, design a Monte Carlo algorithm that verifies for any given $n \times n$ matrices A, B, C whether $AB = C$. Analyse the running time and failure probability of the algorithm.

2. Design an algorithm that computes $a^n \bmod p$ for given integers $a > 0$, $n \geq 0$ and $p > 1$. Your algorithm should run in time $O(\log n)$ assuming that each multiplication of integers takes constant time.

3. For integer $x > 0$, let $\pi(x)$ denote the number of prime numbers less than or equal to x. The prime number theorem, proved independently by Jacques Hadamard and Charles Jean de la Vallée-Poussin in 1896, states that the $\pi(x) \sim \frac{x}{\ln x}$ as x increases.
Design an efficient randomised algorithm that generates a random prime number of a given length n. Analyse the running time of the algorithm using the prime number theorem.

4. Imagine an online social network which consists of 7 users A, B, C, D, E, F, G as shown below. The edges between the users represent their "friendship" links. Only users that are connected by edges can see each other's messages. Suppose A posts a message. Suppose that if any person sees this message, the probability that the person re-posts it is 0.92. What is the probability that G will eventually see and re-post the message.

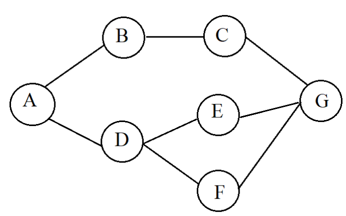

Fig. 1. Calculate the probability that G will re-post a message posted by A.

5. Many real-world complex networks (e.g. Facebook social network, the Internet, protein-protein interaction networks, interbank payment networks) exhibit the so-called *scale-free*

property[1]. This property means that the degree distribution $P_D(x)$ is asymptotically following the distribution x^{-b} for some positive real number $b \in [2, 3]$, i.e., $P_D(x) \in O(x^{-b})$. Suppose G is a random network with scale-free property and the parameter $b = 2$. What is the expected degree of nodes in G? You may express the result as an asymptotic expression.

[1] You may be interested to explore the fascinating field of social network analysis: `https://en.wikipedia.org/wiki/Social_network_analysis` offers a start.

Answers:

1. The algorithm runs as follows:
   (a) For $i = 1, 2, \ldots, n$, generate $r_i \in \{0, 1\}$.
   (b) Let the vector $\boldsymbol{r}$ be $(r_1, \ldots, r_n)$.
   (c) Compute $\boldsymbol{x} \leftarrow B\boldsymbol{r}$, and then $\boldsymbol{x} \leftarrow A\boldsymbol{x}$.
   (d) Compute $\boldsymbol{y} \leftarrow C\boldsymbol{r}$
   (e) Check if $\boldsymbol{x} = \boldsymbol{y}$. If so, output that $AB = C$; otherwise, output $AB \neq C$.

   Correctness analysis:
   - The algorithm has one-sided error: If $AB = C$, then the algorithm is always correct. If $AB \neq C$, then the algorithm is corrected with probability $\geq 1/2$.
   - Thus by repeating the procedure k times, and outputing $AB = C$ only if the vectors $\boldsymbol{x} = \boldsymbol{y}$ in all these times, we can reduce the failure probability to $< \frac{1}{2^k}$.

   Running time analysis:
   - Generating a random n-bit integer takes $O(n)$.
   - Computing $\boldsymbol{x}$ and $\boldsymbol{y}$ takes $O(n^2)$.
   - Comparing $\boldsymbol{x}$ with $\boldsymbol{y}$ takes $O(n)$.
   - Thus each time the algorithm is run, it takes $O(n^2)$.
   - Over k trials, the algorithm takes $O(kn^2)$.

Algorithm 1 ModExponentiation(a, n, p)

2. INPUT Integer $a > 0$, $n \geq 0$ and $p > 1$.
OUTPUT $a^n \bmod p$.

```
Express n in its binary form where n = n[ℓ]n[ℓ−1]...n[1] where each n_i ∈ {0,1}, and ℓ is the number
of bits in n.
x ← a
y ← 1
for i = 1,...,ℓ do
    if n[i] = 1 then
        y ← yx mod p
    end if
    x ← x^2 mod p
end for
return y.
```

3. The procedure runs as follows:
   (a) Generate n 0 or 1 bits $x[1]x[2]\ldots x[n]$ and let the number x be

   $$x[1]x[2]\ldots x[n]$$

   (b) Run primality test to check if x is prime.
   (c) if x is a prime, stop and output x.
   (d) if x is not a prime, repeat.

   Since $\pi(x) \sim \frac{x}{\ln x}$ and since $n \sim \ln x$, among all n-bit integers $\sim 1/n$ are prime. We assume that the primality test is 100% correct and efficient. Treating the procedure above as a geometric random variable X, where X denotes the number of trials of the algorithm until the first prime number being generated.
   The number of trials we expect to make is $\mathcal{E}[X]$ which is roughly $O(1/(1/n)) = O(n)$.

4. $1 - (1 - 0.92^2) \times (1 - 0.92 \times (1 - (1 - 0.92)^2)) = 0.9868$

5. $O(\ln n)$ using Harmonic number