Problem 1

Question:

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 $\vec{r} \in \{0, 1\}^n$ uniformly at random, then the probability

$$Pr(\{AB\vec{r} = C\vec{r}\}) \leq \frac{1}{2}$$

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.

Answer:

We can state the algorithm as follows:

Algorithm 1 Monte Carlo Algorithm for checking $AB = C$

```
1: procedure MonteCarloAlgorithm($A, B, C, k$)                    ▷ Size: $A, B, C = n \times n$, and $k \in Z$
2:     count ← 0
3:     Generate $n \times n$ Zero Matrix $= Z$
4:     while count $\neq k$ do                                    ▷ $\mathcal{O}(k)$
5:         Generate a random $\vec{r} \in \{0, 1\}$ of size $[n, 1]$  ▷ $\mathcal{O}(1)$
6:         Find $D = B \times \vec{r}$                            ▷ $\mathcal{O}(n^2)$
7:         Find $E = A \times D$                                  ▷ $\mathcal{O}(n^2)$
8:         Find $F = C \times \vec{r}$                            ▷ $\mathcal{O}(n^2)$
9:         Compute $G = E - F$                                    ▷ $\mathcal{O}(n^2)$
10:        if $G$ equals $Z$ then                                 ▷ $\mathcal{O}(n^2)$
11:            return $True$                  ▷ Return true if the two matrices are equal
12:        end if
13:        count ← count + 1                                      ▷ $\mathcal{O}(1)$
14:    end while
15:    return $False$                         ▷ Return false, if no equivalence is found
16: end procedure
```

Since this is a Monte Carlo algorithm we need to ask the user for the amount of iterations k and obviously the required inputs are the matrices A, B, C.

The running time for the above algorithm takes the following into account: Matrix Operations (Multiplication and Subtraction) are $\mathcal{O}(n^2)$ and generating a random vector is constant time. From the comments in the algorithm above we get that

$$\boxed{\text{Running Time } = \mathcal{O}(k(1 + n^2 + n^2 + n^2 + n^2 + n^2)) + \mathcal{O}(1) = \mathcal{O}(kn^2)}$$

We can analyze the failure of probability as such: Since we have that the failure of probability is $\frac{1}{2}$, and we have k iterations of the loop we will have that:

$$\boxed{\text{Failure of Probability } = \frac{1}{2^k}} \quad \text{Where } k \text{ is the number of iterations of the loop}$$

Problem 2

Question:

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

Answer:

First we shall consider the expression $a^n \mod p$, since we know taking the modulus of a number is a constant time operation our question reduces to finding a^n in $\mathcal{O}(\log(n))$ time. Now consider this divide and conquer algorithm for computing the power:

Algorithm 2 Algorithm for finding a^n

```
1: procedure POWER($a, n$)                                  ▷ $a > 0$ and $n \geq 0$
2:     if $n$ equals 0 then                                 ▷ Base Case since $a^0 = 1$
3:         Return 1;
4:     end if
5:     if $n$ is even then                                  ▷ $n$ is in the form $2k$ where $k \in Z$
6:         Return Power $\left(a, \frac{n}{2}\right) \times$ Power $\left(a, \frac{n}{2}\right)$
7:     else if $n$ is odd then                              ▷ $n$ is in the form $2k + 1$ where $k \in Z$
8:         Return $a \times$ Power $\left(a, \frac{n-1}{2}\right) \times$ Power $\left(a, \frac{n-1}{2}\right)$
9:     end if
10: end procedure
```

Algorithm 3 Algorithm for computing $a^n \mod p$

```
1: procedure POWERMOD($a, n, p$)                            ▷ $a > 0$ and $n \geq 0$, $p > 1$
2:     Return Power($a, n$) mod $p$
3: end procedure
```

The runtime of $PowerMod$ is $\mathcal{O}(\log(n))$ since finding the power takes $\mathcal{O}(\log(n))$ and taking the modulus of that number and multiplication of integers is $\mathcal{O}(1)$.

We can justify the runtime of the power function as such, consider the recursive step:

$$a^n = \begin{cases} 1 & \text{if } n = 0 \\ a^{\frac{n}{2}} \cdot a^{\frac{n}{2}} & \text{if } n \text{ is even} \\ a \cdot a^{\frac{n-1}{2}} \cdot a^{\frac{n-1}{2}} & \text{if } n \text{ is odd} \end{cases}$$

From this we can form a recurrence relation that approximates the run time of the recursive algorithm (The last step is done by using master theorem) :

$$T(n) = T\left(\frac{n}{2}\right) + \mathcal{O}(1) \implies \boxed{\text{Running Time of Power Algorithm} = \mathcal{O}(\log(n))}$$

Intuitively we can think of a^n being **dividing** up into smaller values (by a factor of atleast a half) in a binary tree and that tree will have a height of $\log(n)$. And conquering up the tree and multiplying the numbers will take constant time time therefore total running time would be $\mathcal{O}(\log(n))$.

$$\boxed{\text{Running Time of PowerMod Algorithm} = \mathcal{O}(\log(n) \cdot 1 \cdot 1) = \mathcal{O}(\log(n))}$$

Problem 3

Question:

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 Vallèe-Poussin in 1896, states that the $\pi(x) \sim x$ as $\frac{x}{\ln x}$ 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.

Answer:

Assume decimal number system for both cases.

We shall consider 2 Cases:

1st Case - Padding Numbers: Suppose you have a number x with length 2 we can say the maximum number is 99 and the minimum number is 00, if length is 4 we can say the maximum number is 9999 and the minimum number is 0000. (So essentially padding bits to the start)

After addressing this first case we can now produce an algorithm:

Algorithm 4 Algorithm for computing a random prime number with length n

```
1:  procedure GeneratePrimePadding($n$)                                      ▷ $n > 0$
2:      Produce Max Number = $10^n - 1$                  ▷ Produces max number with $n$ digits: $\mathcal{O}(1)$ time
3:      Produce Min Number = $(00...0)_n$                   ▷ Produces zero with $n$ digits: $\mathcal{O}(1)$ time
4:      while A prime number has not been found do                  ▷ $\mathcal{O}(\log(\text{Max Number}))$
5:          Produce random number in interval : $r = (\text{Min Number}, \text{Max Number}]$       ▷ $\mathcal{O}(1)$ time
6:          if $r$ is prime using Fermats Primality Test then                 ▷ $\mathcal{O}(\log(10^n))$ time
7:              Return $r$
8:          end if
9:      end while
10: end procedure
```

We can analyse this randomized algorithm as follows:

Producing Max Number, Min Number and a random number between that range takes constant time. Also Fermat's primality test takes $\mathcal{O}(\log(10^n))$ time.

Since we know by the Prime Number Theorem that $\pi(x) \approx \frac{x}{\log(x)}$ (which means there are that many prime numbers less than x), therefore we know that in our algorithm after finding the maximum possible number (Max) with length n, we can find that $\pi(\text{Max}) \approx \frac{\text{Max}}{\log(\text{Max})}$ which gives us the amount of primes less than max.

We can then find the probability that a randomly chosen value in the range is prime which will be approximately: $\frac{\frac{\text{Max}}{\log(\text{Max})}}{\text{Max}} = \frac{1}{\log(\text{Max})} = p$

After this we can produce a random geometric variable Y such that it represents the number of trials we need before we get a successful prime number. Since we know that the expectation of a geometric random variable is $\frac{1}{p}$ (from lectures) which implies that we would *expect* approximately $\log(\text{Max})$ trials before we correctly generate a prime number.

Also from looking at the algorithm above we can conclude that the running time is:

$$\boxed{T(n) = \mathcal{O}(\log(10^n) \cdot \log(10^n - 1)) \approx n^2}$$

Problem 3 continued on next page...

2nd Case - Not-Padding Numbers: Suppose you have a number x with length 2 we can say the maximum number is 99 and the minimum number is 10, if length is 4 we can say the maximum number is 9999 and the minimum number is 1000. (So not padding any bits like the previous case)

After addressing this first case we can now produce an algorithm:

Algorithm 5 Algorithm for computing a random prime number with length n

```
procedure GeneratePrimeNoPadding($n$)                                  ▷ $n > 0$
    Produce Max Number = $10^n - 1$          ▷ Produces max number with $n$ digits: $\mathcal{O}(1)$ time
    Produce Min Number = $10^{n-1}$          ▷ Produces zero with $n$ digits: $\mathcal{O}(1)$ time
    while A prime number has not been found do                   ▷ Analysis done below
        Produce random number in interval : $r = [\text{Min Number}, \text{Max Number}]$      ▷ $\mathcal{O}(1)$ time
        if $r$ is prime using Fermats Primality Test then        ▷ $\mathcal{O}(\log(10^n))$ time
            Return $r$
        end if
    end while
end procedure
```

We can analyse this randomized algorithm as follows:

Producing Max Number, Min Number and a random number between that range takes constant time. Also Fermat's primality test takes $\mathcal{O}(\log(10^n))$ time.

By the prime number theorem we can say that there cannot be more than $\pi(\text{Max}) - \pi(\text{Min})$ where max is the maximum number of length n and the min is the minimum number of length n. More mathematically there cannot be more than $\frac{\text{Max}}{\log \text{Max}} - \frac{\text{Min}}{\log \text{Min}}$ prime numbers.

Therefore the probability that a randomly chosen value in the range is prime is (For the sake of simplification, let max $= x$ and $min = y$:

$$p = \frac{\frac{\text{Max}}{\log \text{Max}} - \frac{\text{Min}}{\log \text{Min}}}{\text{Max} - \text{Min}} = \frac{\frac{x}{\log(x)} - \frac{y}{\log(y)}}{x - y} = \frac{\log(y^x) - \log(x^y)}{(x-y)\log(x)\log(y)} = \frac{\log\left(\frac{y^x}{x^y}\right)}{(x-y)\log(x)\log(y)}$$

After this we can produce a random geometric variable Y such that it represents the number of trials we need before we get a successful prime number. Since we know that the expectation of a geometric random variable is $\frac{1}{p}$ (from lectures).

Therefore the expected number of trials before we get a prime is:

$$\frac{1}{p} = \frac{1}{\frac{\log\left(\frac{y^x}{x^y}\right)}{(x-y)\log(x)\log(y)}} = \frac{(x-y)\log(x)\log(y)}{\log\left(\frac{y^x}{x^y}\right)} = \frac{(\text{Max} - \text{Min})\log(\text{Max})\log(\text{Min})}{\log\left(\frac{\text{Min}^{\text{Max}}}{\text{Max}^{\text{Min}}}\right)}$$

This means the while loop will run at most $\frac{(\text{Max}-\text{Min})\log(\text{Max})\log(\text{Min})}{\log\left(\frac{\text{Min}^{\text{Max}}}{\text{Max}^{\text{Min}}}\right)}$ times before termination.

Also from looking at the algorithm above we can conclude that the running time (after simplification) is:

$$\boxed{T(n) = \mathcal{O}\left(\log(10^n) \cdot \frac{(\text{Max} - \text{Min})\log(\text{Max})\log(\text{Min})}{\log\left(\frac{\text{Min}^{\text{Max}}}{\text{Max}^{\text{Min}}}\right)}\right) = \mathcal{O}\left(n \cdot \frac{(\text{Max} - \text{Min})\log(\text{Max})\log(\text{Min})}{\log\left(\frac{\text{Min}^{\text{Max}}}{\text{Max}^{\text{Min}}}\right)}\right)}$$

Problem 4

Question:

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.

Answer:

First we shall define some events:

- $B = B$ reposting the message
- $C = C$ reposting the message
- $D = D$ reposting the message
- $E = E$ reposting the message
- $F = F$ reposting the message
- $G_s = G$ seeing the message
- $G_r = G$ reposting the message

Also note that G will not see the post if

- B and D don't repost
- B, E and F don't repost
- C, E and F don't repost
- C and D don't repost

And lastly note that the probability of any node reposting the message is 0.92 and any node not reposting posting is 0.08

To find the answer to the question we need to find out $P(G_s \cap G_r)$ and since seeing the message doesn't determine if repost occurs (i.e. independence) we can now find the probability:

$$
\begin{aligned}
&P(G_s \cap G_r) = P(G_s) \cdot P(G_r) \\
&P(G_s \cap G_r) = P(G'_s) \cdot P(G_r) \\
&P(G_s \cap G_r) = (1 - (P(B' \cap D') + P(B' \cap E' \cap F') + P(C' \cap E' \cap F') + P(C' \cap D'))) \cdot P(G_r) \\
&P(G_s \cap G_r) = (1 - (0.08)^2 - 0.08 \cdot (0.92 \cdot 0.08)^2 - (0.92 \cdot 0.08)^3 - 0.08 \cdot (0.92 \cdot 0.08)) \cdot 0.92 \\
&P(G_s \cap G_r) = 0.9868799549 \cdot 0.92 \\
&P(G_s \cap G_r) \approx 0.9079
\end{aligned}
$$

$$\boxed{\therefore P(G_s \cap G_r) \approx 0.9079}$$

Problem 5

Question:

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. 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. $P_D(x) \in \mathcal{O}(x^{-b})$. 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.

Answer:

Consider a random graph $G = (V, E)$ and degree distribution $P_D(x) \in \mathcal{O}(x^{-2})$ we wish to find the expected degree of nodes in G.

Denote the degree of some node $n \in V$ by $d(v)$.

Note that the max degree is when the vertex has $|V| - 1$ edges (to all vertices but itself). We set an upper bound of $|V| - 1$ so our expression doesn't diverge *but* converges.

Hence we can calculate the expected degree of nodes in G as such:

$$\begin{aligned}
\mathbb{E}[D(n)] &= \sum_{x \in \mathbb{R}: P_D(x) > 0} x \cdot P_D(x) \\
\mathbb{E}[D(n)] &= \sum_{x=1}^{|V|-1} x \cdot P(d(v) = x) \\
\mathbb{E}[D(n)] &= \sum_{x=1}^{|V|-1} x \cdot \mathcal{O}(x^{-2}) \\
\mathbb{E}[D(n)] &\leq C \sum_{x=1}^{|V|-1} x \cdot (x^{-2}) \\
\mathbb{E}[D(n)] &= C \sum_{x=1}^{|V|-1} \frac{1}{x} \\
\mathbb{E}[D(n)] &= C \cdot H_{|V|-1} \\
\mathbb{E}[D(n)] &\approx C \cdot \log(|V| - 1) + \Theta(1) \\
\mathbb{E}[D(n)] &\in \mathcal{O}(\log(|V| - 1))
\end{aligned}$$

We can say that $\mathbb{E}[D(n)] = H_{|V|-1} \to (\log(|V| - 1))$ and $\mathbb{E}[D(n)]$ grows with $\log(|V|)$ so we conclude that

$$\boxed{\text{Expected Degree of Nodes in G approaches : } \log(|V| - 1)}$$