Problem A

Consider a traffic monitoring system that tracks the number of IP packets transferred for each traffic flow. A flow $x$ is identified by the pair of IP source and destination addresses. We assume that the data structure to store the number of packets for each flow is a hash table with 65,536 buckets, specifically designed to exploit the “power of 2 random choices” result for random load balancing.

1. What is the claim of the “power of 2 random choice” result? Why is it relevant for hash tables?

2. Describe in details two hash functions $h_1(x)$ and $h_2(x)$ specifically designed for the flow identifier considered in the problem and for the given size of the hash table.

3. Describe in pseudocode the operation to update the hash table when a new packet of flow $x$ is processed.
Problem B

Consider an $N \times N$ rearrangeable Clos network, with $N = 3^h$ for some integer $h > 1$, factorized recursively with factor 3, and using only $3 \times 3$ modules. Let $C_3$ be a $3 \times 3$ module.

1. Evaluate formally the complexity in terms of $C_3$ for any $h$.

Now consider the specific case $N = 9$.

1. Draw the whole network.

2. Compute the complexity in terms of $C_3$.

3. Connect the following input-output couples: $1 \rightarrow 9, 2 \rightarrow 2, 3 \rightarrow 4, 4 \rightarrow 5, 5 \rightarrow 3, 6 \rightarrow 7, 7 \rightarrow 6, 8 \rightarrow 8, 9 \rightarrow 1$.

4. Show just the final Pauli matrix at the end of the above connections.
Problem C

Consider a slotted bufferless switch of size $N \times M$, with any $N$ and $M$, comprising all the possible three cases: $N = M$, $N > M$ and $N < M$. When an output contention occurs among different packets, one packet at random is transferred across the switch. Assume that the arrival process is Bernoulli i.i.d. being $\rho \in [0, 1]$ the normalized average load at an input. The traffic is uniformly distributed across all the inputs and outputs.

1. Compute the admissibility conditions for $\rho$

2. Compute the throughput achievable in function of $\rho$, describing in details all the steps in the derivation.

3. Assume $\alpha = N/M$ fixed and $N \to \infty$. What is the maximum throughput achievable? If needed, recall that

$$\lim_{x \to \infty} \left(1 + \frac{\alpha}{x}\right)^x = e^\alpha$$
Hints for the solution

Problem A

1. See the class notes

2. Each hash function must map a sequence of 64 bits (i.e. 32 bits for the IP source and 32 bits for the IP destination), denoted as $x$ to a sequence of 16 bits (i.e. log$_2$ 65536, to index each bucket of the hash table). Let $x_1$ the integer value ($\in [0, 2^{32} - 1]$) corresponding to the IP source address and the $x_2$ the one corresponding to the IP destination address. Thus, one possible option would be $h_1(x) = x_1 \mod 2^{16}$ and $h_2(x) = x_2 \mod 2^{16}$, i.e. $h_1(x)$ uses the 16 least significant bit of the source IP address and $h_2(x)$ the 16 least significant bit of true source IP address. Notably, using instead the most significant bits would perform poorly (i.e. higher hash collision) due to the particular structure of the IP addresses.

3. Let $T$ be the hash table and $T[i]$ is the $i$th bucket, implementing a linked list to store elements in the format $(x, v)$, where $x$ is the flow identifier and $v$ is the corresponding number of packets. The operation to update $T$ whenever a new packet of $x$ is received is described as follows:

```python
function UPDATE FLOW(x)
    if (x, v) ∈ T[h_1(x)] then  ▷ x found in bucket $h_1(x)$
        Insert (x, v + 1) in T[h_1(x)] ▷ Update counter and store it again
    else if (x, v) ∈ T[h_2(x)] then  ▷ x found in bucket $h_2(x)$
        Insert (x, v + 1) in T[h_2(x)] ▷ Update counter and store it again
    else
        $s_1$ = sizeof(T[h_1(x)])  ▷ Find the size of bucket $h_1(x)$
        $s_2$ = sizeof(T[h_2(x)])  ▷ Find the size of bucket $h_2(x)$
        if $s_1 < s_2$ then
            Insert (x, 1) in T[h_1(x)]  ▷ Store $x$ into the smallest bucket
            $s_1$ is the smallest
        else
            Insert (x, 1) in T[h_2(x)]  ▷ $s_2$ is the smallest
        end if
    end if
end function
```

Problem B

Refer to exercises 16 and 21.

Problem C

Refer to exercises 60.