Problem 1

We observe that $R(t)$ depends on the value of $Z(t)$, which in turn depends upon previous values of $X_n(t)$ for all n. The specific dependency is a function of the sampling function $I(t)$. Thus, we may observe that while $X(t)$ is not a markov process, $Y(t) = (X(t), X(Z(t)))$.

If we let $X(t) = (x_1, x_2, \ldots, x_n)$ and $M(t) = (M_1(t), M_2(t), \ldots M_n(t))$, we can similarly let $A(t) = (A_1(t), A_2(t) \ldots A_n(t))$. With this, $A(t)$ is a matrix with all zeros except for a one at index $R_n(t)$ which is the index of the smallest queue, which is also $\argmin_n(X_n(Z^n(t))$

$$Y(t+1) = ((X(t) - M(t) + A_n(t), X(Z(t+1))) = ((X(t) - M(t) + A_n(t), X(Z(t+1)))$$

In scenario A, we may compute $X(Z(t+1)) = X_1(Z^1(t)), X_2(Z^3(t)) \ldots X_{(t \pmod N)+1)}(t) \ldots X_N(Z^n(t))$, which is to say that we can only update queue $t \pmod N + 1$ with the actual queue length. Thus,

$$R(t+1) = \begin{cases} R(t) & X_{R(t)}(t) < X_{(t \pmod N)+1)}(t) \\ t \pmod N + 1 & X_{R(t)}(t) \geq X_{(t \pmod N)+1)}(t) \end{cases}$$

Let queue n spend k_n time steps being the shortest sampled queue.Thus, on average, it is receiving reqeusts for k_n time steps, so will expect to append ak_n new elements every N time steps and then expect to process Nm_n of them every N time steps. The system will be stable if $aE[k] < Nm_n$ for every n.

$R(t)$ will only equal a certain queue for a finite period of time, before the next sampled queue has had sufficiently long enough to If m_n is higher, intuitively, k_n will also be larger, but only to a certain point, because $\sum_{n=1}^{N} E[k_n] = N$ while m_n can be unbounded. Suppose that the distribution of M is very skewed, and one m_i dominates every other m. While m_i may be larger, it will thus process it's queue faster, but as a consequence, will also receive correspondingly more packets to process, meaning that on net, it should equilibriate.

While I don't have a good way to solve for k, I do believe that it will even out to where the criterion is $a < \sum_{i=1}^{n} m_n$.

We can show that this is sufficient by using the Lyapunov function on Y of

$$V(Y(t)) = \sum_{i=1}^{n} X_i(t)$$

We want to compute

$$\begin{aligned} E[V(Y(t+1)) - V(Y(t)) | Y(t) = y_0] &< 0 \\ E[[\sum_{n=1}^{N} X_n(t) - \sum_{n=1}^{N} M_n(t)]^+ + \sum_{n=1}^{N} A_n(t)] - \sum_{n=1}^{N} X_n(t) &< 0 \\ E[\sum_{n=1}^{N} A_n(t) - \sum_{n=1}^{N} M_n(t)] &< 0 \\ a_n - \sum_{n=1}^{N} m_n(t) &< 0 \\ a_n &< \sum_{n=1}^{N} m_n(t) \end{aligned}$$

as desired

In [ ]:
```
import pandas as pd
import numpy as np
```

In [ ]:
```
N = 5
M = np.array([0.2, 0.2, 0.1, 0.15, 0.05])
delta_list = [0.5, 0.7, 0.9]
# chosen arbitrarily because none included
p_array = np.array([0.1, 0.2, 0.3, 0.2, 0.2])


class ServerScheme:
    def __init__(self, name, N, p_array, sample_func=None):
        self.name = name
        self.N = N
        self.p_array = p_array
        self.sample_func = sample_func

    def sample(self, t, X, Z):
        return self.sample_func(self.N, t, self.p_array, X, Z)

    def __str__(self) -> str:
        return f"Sampling Scheme: {self.name}"


def A(N=None, t=None, p_array=None, X=None, Z=None):
    update_i = t % N
    Z[update_i] = X[update_i]
    return Z, np.argmin(Z)


def B(N=None, t=None, p_array=None, X=None, Z=None):
    update_i = np.random.choice(np.arange(0, N), p=p_array)
    Z[update_i] = X[update_i]
    return Z, np.argmin(Z)


def JSQ(N=None, t=None, p_array=None, X=None, Z=None):
    return Z, np.argmin(X)


def simulate(N, M, M_N_map, T, delta, sampling_scheme):
    X = np.zeros(N)
    Z = np.zeros(N)
    a = delta * np.sum(M)
    avg_x = []
    for t in range(T):
        Z, sample_i = sampling_scheme.sample(t, X, Z)
        a_t = np.random.binomial(1, a, 1)[0]
        X[sample_i] += a_t
        for m in range(len(M)):
            X[M_N_map[m]] = max(0, X[M_N_map[m]] -
                                np.random.binomial(1, M[m], 1)[0])
        avg_x.append(np.mean(X))
    return np.mean(np.array(avg_x))

A_scheme = ServerScheme("A", N, p_array, A)
B_scheme = ServerScheme("B", N, p_array, B)
JSQ_scheme = ServerScheme("JSQ", N, p_array, JSQ)

schemes = [A_scheme, B_scheme, JSQ_scheme]
print("Average X(t): 5 separate queue model")
res_df = pd.DataFrame()
```

```
for delta in delta_list:
    for scheme in schemes:
        res_df.loc[delta, scheme] = simulate(N, M, {k: k for k in range(N)}, 1000, delta, s
display(res_df)


A_single_scheme = ServerScheme("A", 1, np.ones(1), A)
B_single_scheme = ServerScheme("B", 1, np.ones(1), B)
JSQ_single_scheme = ServerScheme("JSQ",1, np.ones(1), JSQ)
single_schemes = [A_single_scheme, B_single_scheme, JSQ_single_scheme]
res_df = pd.DataFrame()
print("Average X(t): Single queue model")
for delta in delta_list:
    for scheme in single_schemes:
        res_df.loc[delta, scheme] = simulate(1, M, {k: 0 for k in range(len(M))}, 100000, c
display(res_df)
```

```
Average X(t): 5 separate queue model
```

	Sampling Scheme: A	Sampling Scheme: B	Sampling Scheme: JSQ
0.5	0.6216	1.0122	0.4014
0.7	0.9796	1.6350	0.6866
0.9	1.5150	2.9628	1.0340

```
Average X(t): Single queue model
```

	Sampling Scheme: A	Sampling Scheme: B	Sampling Scheme: JSQ
0.5	0.60641	0.59019	0.59558
0.7	1.33557	1.43487	1.33798
0.9	5.24985	4.87221	5.02690