1 Midsquare Method

The midsquare method algorithm is as follows:

1. Choose an integer seed X_0 that has four digits.
2. Calculate X_0^2.
3. Extract the middle four digits of X_0^2 as the next random number.
4. Repeat the process from step 2.

The following is the implementation of the midsquare method algorithm in Python:

Code 1: Midsquare Method Algorithm in Python

```python
def midsquare(seed, n):
    # Initialize an empty list
    random_numbers = []

    # Loop n times to generate n random numbers
    for i in range(n):
        # Square the current seed
        seed = str(seed ** 2)

        # Add leading zero's if the result has odd length
        if len(seed) % 2 == 1:
            seed = '0' + seed

        # Extract the middle 4 digits of the squared result
        seed = int(seed[2:6])

        # Append the new seed (random number) to the list
        random_numbers.append(seed)

    # Return the list of generated random numbers
    return random_numbers
```

2 Linear Congruential Method

The linear congruential method algorithm is as follows:

1. Choose four integers a, c, m, and X_0. The integer must satisfy the following conditions: $m > 0$ and a, c, $X_0 < m$.
2. Calculate $X_{i+1} \equiv (aX_i + c) \mod m$.
3. Convert X_{i+1} to a random number by dividing it by m.
4. Repeat the process from step 2.

The following is the implementation of the linear congruential method algorithm in Python:

Code 2: Linear Congruential Method Algorithm in Python

```python
def linear_congruential(a, c, m, seed, n):
    # Initialize an empty list
    random_numbers = []

    # Loop n times to generate n random numbers
    for i in range(n):
        # Calculate the next random number
        seed = (a * seed + c) % m

        # Append the new seed (random number) to the list
        random_numbers.append(seed / m)

    # Return the list of generated random numbers
    return random_numbers
```

Theorem 1. *The LCM has a full period if and only if:*

1. *c and m are relatively prime,*
2. *if q is a prime number that divides m, then q divides $a-1$,*
3. *$a-1$ is divisible by 4 if m is divisible by 4.*

Proof. The full period means that the sequence must not be repeated until the m-th number. Consider the recurrence relation:

$$X_{i+1} \equiv (aX_i + c) \mod m \tag{1}$$

The sequence will be repeated if $X_{i+1} = X_i$. This means that:

$$(aX_i + c) \mod m = X_i \tag{2}$$

Rearranging equation (2) gives:

$$aX_i \equiv X_i - c \mod m \tag{3}$$

Since $X_i < m$, then $X_i - c < m$. This means that $X_i - c$ is a non-negative number. Therefore, $X_i - c \mod m = X_i - c$. Substituting this into equation (3) gives:

$$aX_i \equiv X_i - c \mod m \tag{4}$$

Rearranging equation (4) gives:

$$(a-1)X_i \equiv -c \mod m \tag{5}$$

Since $c < m$, then $-c < m$. This means that $-c$ is a non-negative number. Therefore, $-c \mod m = -c$. Substituting this into equation (5) gives:

$$(a-1)X_i \equiv -c \mod m \tag{6}$$

Since $X_i < m$, then X_i is relatively prime to m. This means that X_i has a multiplicative inverse modulo m. Multiplying both sides of equation (6) by X_i^{-1} gives:

$$a - 1 \equiv -cX_i^{-1} \mod m \tag{7}$$

Since X_i is relatively prime to m, then X_i^{-1} exists. This means that equation (7) has a solution if and only if c and m are relatively prime. This proves the first condition. The second condition can be proven similarly. The third condition can be proven by considering the case when m is divisible by 4. This completes the proof. □

3 Combined Linear Congruential Generators

The combined linear congruential generator (CLCG) is a method to generate random numbers by combining multiple LCGs. The algorithm is as follows:

1. Choose k LCGs with parameters a_i, c_i, m_i, and X_{0i} for $i = 1, 2, \ldots, k$.
2. Calculate $X_{i+1} \equiv \left(\sum_{j=1}^{k} a_j X_{ij} + c_j\right) \mod m_j$ for $i = 1, 2, \ldots$.
3. Convert X_{i+1} to a random number by dividing it by m_j.
4. Repeat the process from step 2.

The following is the implementation of the CLCG algorithm in Python:

Code 3: Combined Linear Congruential Generator Algorithm in Python

```python
def clcg(a, c, m, seed, n):
    # Initialize an empty list
    random_numbers = []

    # Loop n times to generate n random numbers
    for i in range(n):
        # Calculate the next random number
        seed = sum([ai * xi for ai, xi in zip(a, seed)]) + c
        seed = [si % mi for si, mi in zip(seed, m)]

        # Append the new seed (random number) to the list
        random_numbers.append(seed)

    # Return the list of generated random numbers
    return random_numbers
```