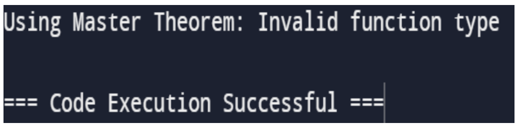
5. Write C programs for solving recurrence relations using the Master Theorem, Substitution Method, and Iteration Method will demonstrate how to calculate the time complexity of an example recurrence relation using the specified technique.

```
def master_theorem(a, b, k, f):

    if isinstance(a, int) and isinstance(b, int) and isinstance(k, int):

        if a < 1 or b < 1 or k < 0:

            return "Invalid input: a, b should be >= 1 and k should be >= 0"

    else:

        return "Invalid input: a, b, and k should be integers"



    if f == "n^k":

        if a == b ** k:

            return "Theta(n^k log n)"

        elif a < b ** k:

            return "Theta(n^k)"

        else:

            return "Theta(n^k log_b a)"

    elif f == "n^k log n":

        if a == b ** k:

            return "Theta(n^k log n log n)"

        elif a < b ** k:

            return "Theta(n^k log n)"

        else:

            return "Theta(n^k log_b a)"

    else:

        return "Invalid function type"
```

```
a = 2

b = 2

k = 1

f = "n"

print("Using Master Theorem:", master_theorem(a, b, k, f))
```

Output:



```
Using Master Theorem: Invalid function type

=== Code Execution Successful ===
```

TIME COMPLEXITY:-O(nlogn)