5. Write 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.

PROGRAM:

USING MASTER'S THEOREM:

```
import math


def master_theorem(a, b, k, p):
    if a > (b ** k):
        return "T(n) = Θ(n^" + str(math.log(a, b)) + ")"
    elif a == (b ** k):
        return "T(n) = Θ(n^" + str(k) + " log n)"
    else:
        return "T(n) = Θ(n^" + str(k) + ")"
a = 2
b = 2
k = 1
p = 0
print("Using Master Theorem:", master_theorem(a, b, k, p))
```

OUTPUT:

```
Using Master Theorem: T(n) = Θ(n^1 log n)
```

TIME COMPLEXITY: Theeta (n log n)

USING SUBSTITUTION METHOD :

PROGRAM:

```
def substitution_method(n):
    if n == 1:
        return 1
    else:
        return 2 * substitution_method(n//2) + n
# Example usage for the given recurrence relation
n = 8
```

```
result = substitution_method(n)
print("Using Substitution Method: T(n) =", result)
```

OUTPUT:

```
Using Substitution Method: T(n) = 32
```

TIME COMPLEXITY: Theeta(n log n)

USING ITERTION METHOD :

PROGRAM:

```
def iteration_method(n):
    total = 0
    while n > 0:
        total += n
        n //= 2
    return total
n = 8
result = iteration_method(n)
print("Using Iteration Method: T(n) =", result)
```

OUTPUT:

```
Using Iteration Method: T(n) = 15
```

TIME COMPLEXITY: Theeta(n)