15. 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.

Program:

```
import math

def master_theorem(a, b, k, n):
    log_b_a = math.log(a, b)
    if k < log_b_a:
        return n ** log_b_a
    elif k == log_b_a:
        return n ** k * math.log(n)
    else:
        return n ** k

a, b, k, n = 2, 2, 1, 8  # T(n) = 2T(n/2) + n
result = master_theorem(a, b, k, n)
print(f"Using Master Theorem, the time complexity is
O(n^log_b_a) = O(n^{math.log(a, b):.2f}) = O({result})")
def substitution_method(n):
    if n == 1:
        return 1
    else:
        return 2 * substitution_method(n // 2) + n

n = 8  # T(n) = 2T(n/2) + n
result = substitution_method(n)
print(f"Using Substitution Method, the time complexity
is O(n*log(n)) = O({result})")
```

```
def iteration_method(n):
    total = 0
    while n > 0:
        total += n
        n //= 2
    return total

n = 8  # T(n) = 2T(n/2) + n
result = iteration_method(n)
print(f"Using Iteration Method, the time complexity is O(n*log(n)) = O({result})")
```

Output:

```
"C:\Program Files\Python312\python.exe" "C:\Work Space\DAA COADS.PYTHON\program 15.py"
Using Master Theorem, the time complexity is O(n^log_b_a) = O(n^1.00) = O(16.635532333438686)
Using Substitution Method, the time complexity is O(n*log(n)) = O(32)
Using Iteration Method, the time complexity is O(n*log(n)) = O(15)

Process finished with exit code 0
```

Time complexity:

O(n log n)