15. Write program 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.

Code:

```
import math

def master_theorem():
    a = 2
    b = 2
    n = 8
    log_result = math.log(a, b)

    print("For T(n) = 2T(n/2) + n, using Master Theorem:")
    print(f"a = {a}, b = {b}, f(n) = n, log_b(a) = {log_result}")

    if log_result == 1:
        print("Case 2: T(n) = O(n log n)")
    elif log_result > 1:
        print("Case 1: T(n) = O(n^log_b(a))")
    else:
        print("Case 3: T(n) = O(f(n))")

def substitution_method(n):
    if n <= 1:
        return 1
    else:
        return 2 * substitution_method(n // 2) + n

def substitution_demo():
    n = 8
    result = substitution_method(n)
    print(f"Using Substitution Method for T(n) = 2T(n/2) + n, T({n}) = {result}")
```

```
def iteration_method(n):
    sum = 0
    k = 0
    while n > 1:
        sum += n
        n //= 2
        k += 1
    sum += n
    return sum

def iteration_demo():
    n = 8
    result = iteration_method(n)
    print(f"Using Iteration Method, total sum for T(n) = 2T(n/2) + n is {result}")


def main():
    master_theorem()
    substitution_demo()
    iteration_demo()

if __name__ == "__main__":
    main()
```

Output:

```
For T(n) = 2T(n/2) + n, using Master Theorem:
a = 2, b = 2, f(n) = n, log_b(a) = 1.0
Case 2: T(n) = O(n log n)
Using Substitution Method for T(n) = 2T(n/2) + n, T(8) = 32
Using Iteration Method, total sum for T(n) = 2T(n/2) + n is 15
```

Time Complexity:

- T(n)= O(log-b(n))