113. Complexity Analysis techniques: Master theorem, Substitution Method, Iteration Method

AIM: To find Complexity Analysis techniques: Master theorem, Substitution Method, Iteration Method

PROGRAM:

```
import math

def master_theorem_analysis(a, b, f_n):
    """ Master Theorem Analysis """
    # Determine log_b a
    log_b_a = math.log(a, b)

    if f_n == 'n':
        return f'T(n) = Theta(n^{log_b_a} log^k n) where k = 0'
    elif f_n == 'n^2':
        return f'T(n) = Theta(n^{log_b_a} log^k n) where k = 1'
    elif f_n == 'n^3':
        return f'T(n) = Theta(n^{log_b_a} log^k n) where k = 2'
    elif f_n == 'n^k':
        k = int(f_n[-1])
        return f'T(n) = Theta(n^{log_b_a} log^k n)'
    else:
        return 'Invalid input'

def substitution_method_analysis():
    """ Substitution Method Analysis """
    return 'T(n) = O(n log n)'

def iteration_method_analysis():
    """ Iteration Method Analysis """
    return 'T(n) = O(n log n)'

# Example usage:
a = 2
b = 2
f_n = 'n'

# Applying Master Theorem
print("Master Theorem Analysis:")
print(master_theorem_analysis(a, b, f_n))
```

```
# Applying Substitution Method
print("\nSubstitution Method Analysis:")
print(substitution_method_analysis())

# Applying Iteration Method
print("\nIteration Method Analysis:")
print(iteration_method_analysis())
```

OUTPUT:

```
Master Theorem Analysis:
T(n) = Theta(n^1.0 log^k n) where k = 0

Substitution Method Analysis:
T(n) = O(n log n)

Iteration Method Analysis:
T(n) = O(n log n)
```

TIME COMPLEXITY: O(n log n)