1. Find the complexity of the following set loops, where n is given as input:

i = n;

while (i > 1) {

j = i;

while (j < n) {

k = 0;

while (k < n) {

k = k + 2;

}

j = j \* 2;

}

i = i / 2;

}

Explanation:

The inner loop (k) runs until k reaches n, going up by 2 each time, so it takes about n/2 steps → O(n).

The middle loop (j) doubles j each time until it becomes n, so it runs about log(n/i) times.

The outer loop (i) keeps cutting i in half each time, so it runs about log n times.

Total time = n × (log n) × (log n)

Final complexity: O(n × (log n)²)

2) Discussion - The Hypothesis

Sometimes we can’t find two functions f(n) and g(n) where one perfectly fits as the upper bound and the other as the lower bound .

This happens when the function doesn’t grow steadily, it goes up and down, like f(n) = n × sin(n).

In those cases, the function doesn’t have a simple “growth pattern,” so we can’t describe it easily using Big-O and Big-Ω.

In short:

Big-O and Big-Ω work best for functions that grow smoothly.

3) Find time complexity of the following recurrence T(n) = 3T(n/4) + n

Using the Master Theorem:

a = 3

b = 4

f(n) = n

We compare n with n^log₄3 ≈ n^0.792.

Since n grows faster, f(n) dominates.

Answer: T(n) = O(n)

4) Find time complexity of the following recurrence T(n) = 8T(n/6) + √n

Here:

a = 8

b = 6

f(n) = √n = n^0.5

Compute n^log₆8 ≈ n^1.086.

That grows faster than √n, so the recursive part dominates.

Answer: T(n) = Θ(n^1.086)

5) Find time complexity of the following recurrence  T(n) = 3T(3n/5) + n

Here:

a = 3

b = 5/3

f(n) = n

We get n^log₍₅⁄₃₎3 ≈ n^2.15, which is larger than n, so again the recursion dominates.

Answer: T(n) = (n^2.15)