1. Worst-Case Complexity

The worst-case complexity of an algorithm is the function defined by the maximum number of steps taken on any instance of size n.

1. Best-Case and Average-Case Complexity

The best-case complexity of an algorithm is the function defined by the minimum number of steps taken on any instance of size n.

The average-case complexity of the algorithm is the function defined by an average number of steps taken on any instance of size n.

1. Big-O Notation

Big-O notation gives and upper bound of the complexity in the worst case, helping to quantify performance as the input size becomes arbitrarily large. This tells you the number of operations an algorithm will make.

Mathematically,

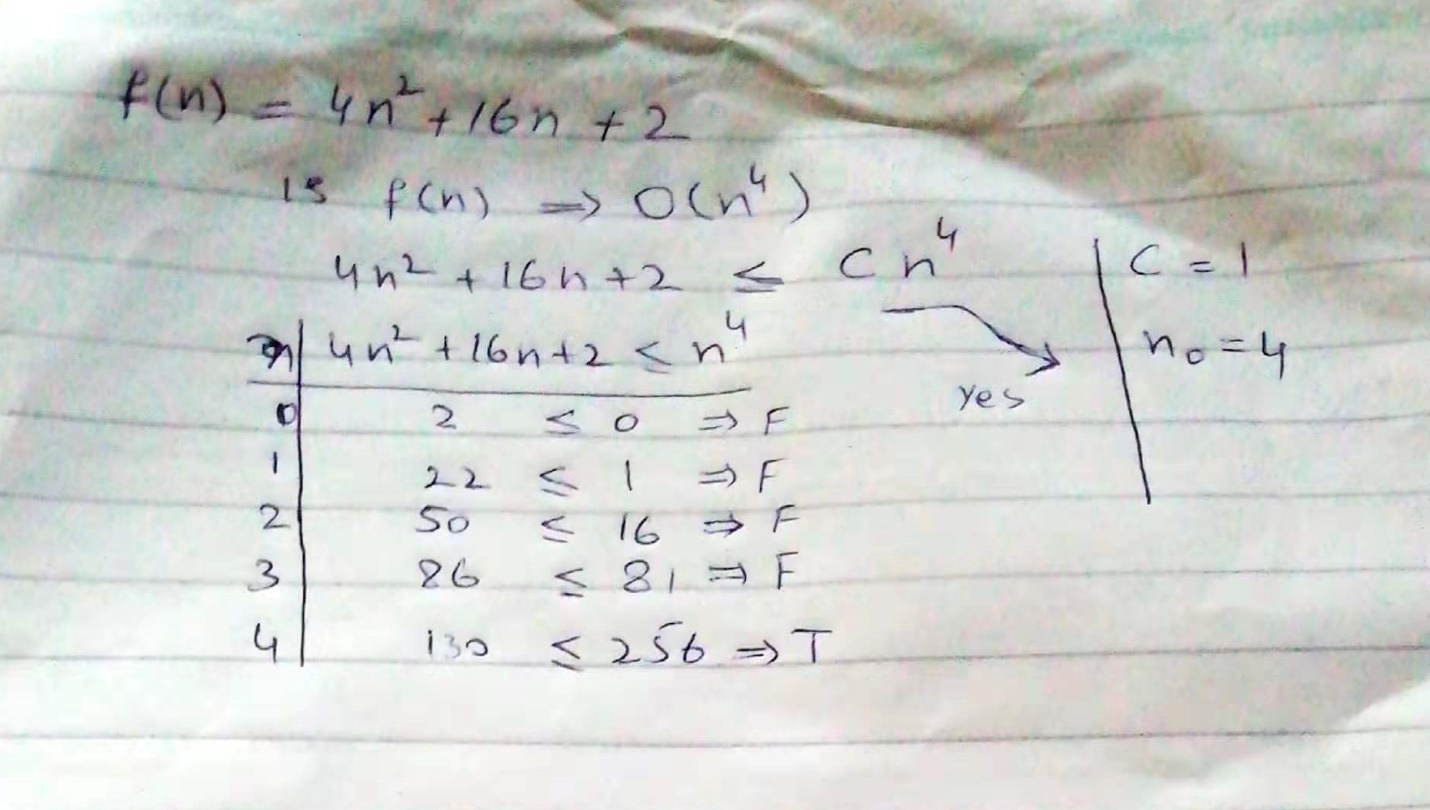
f(n) IS O(g(n)) if C and n0

such that,



Where n0 is some particular value for n

For Example,



Constant Time: O(1)

Logarithmic Time: O(log(n))

Linear Time: O(n)

Linearithmic Time: O(n log (n))

Quadric Time: O(n2)

Cubic Time: O(n3)

Exponential Time: O(bn), b > 1

Factorial Time: O(n!)

1. Big Omega

Big Omega is used to give a lower bound for the growth of a function. It’s defined in the same way as Big O.

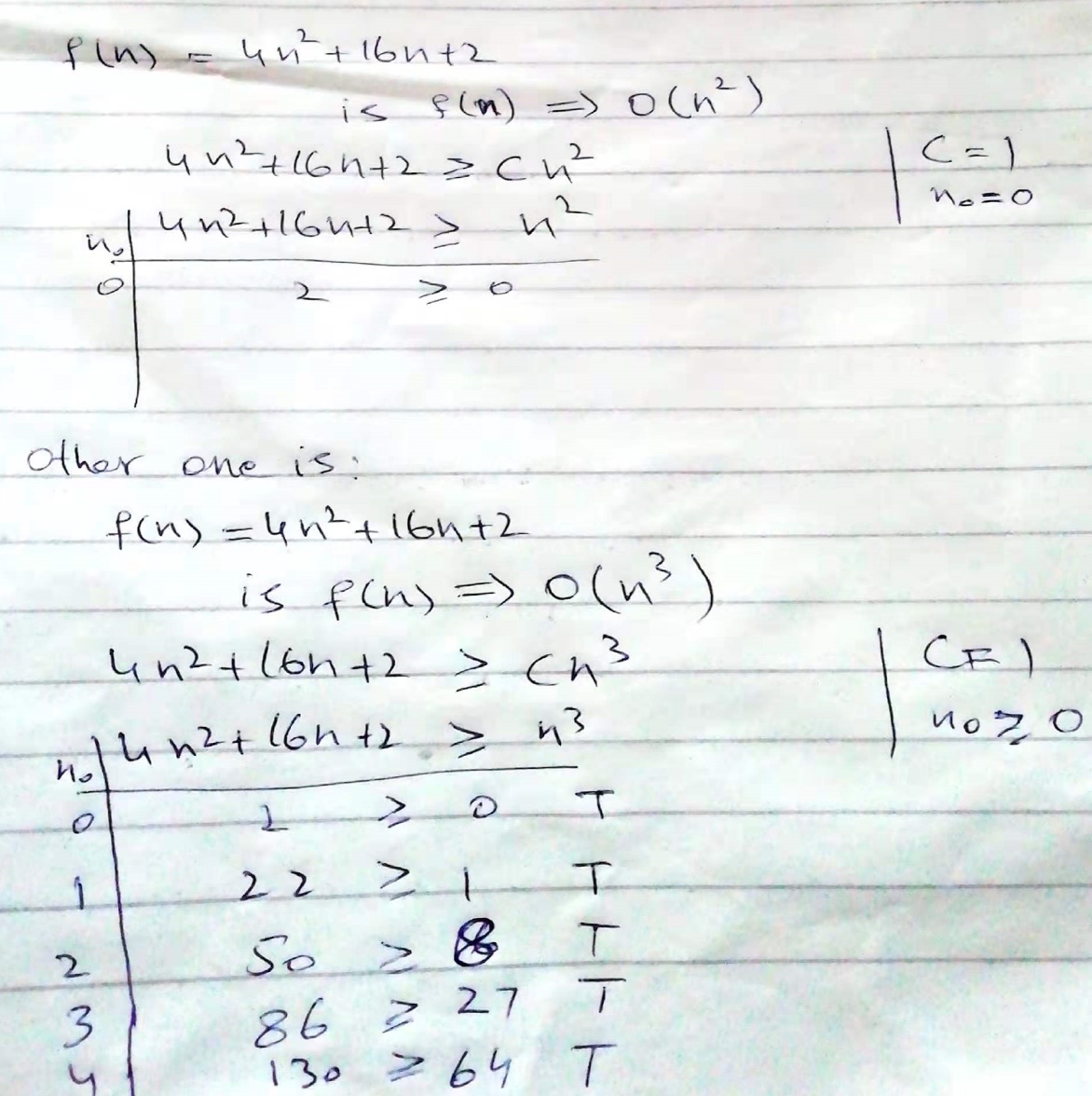
f(n) IS (g(n)) if C and n0

such that,



Where n0 is some particular value for n

For Example,



1. Big Theta

Big Theta is used to indicate that a function is bounded both from above and below.

For big theta both big O and big omega condition should be true for any function.

mathematically:

f(n) IS theta(g(n)) iff

1. f(n) IS O (g(n))
2. f(n) IS omega(g(n))