ANALYSIS OF ALGORITHM (BACKGROUND)

FUNCTION TO RETURN THE SUM OF n NATURAL NUMBERS:

def sum(n):

FORMULAE:

n\*(n+1)/2

total = 0

for i in range(1,n+1):

total +=i

print(total)

sum(5)

We use asymptotic or theoretical analysis to analyze the optimal code for a single problem.

This is not dependent on the machine, programming language, etc.

We do not have to implement all the ideas/algorithms.

The asymptotic analysis measures the order of growth in terms of input size.

ORDER OF GROWTH:

A function f(n) is said to be growing faster than g(n) if

Lim g(n)/f(n) = 0 OR Lim f(n)/g(n) = ∞

n->∞ n->∞

where f(n) and g(n) represents the time taken .

n>=0

f(n),g(n) >= 0

The direct way to find, to find and compare growths:

Step 1: ignore lower-order terms.

Step 2: ignore the leading term constant.

EXAMPLE:

F(n) = 2n²+n+6 order of growth: n²(quadratic)

G(n) = 100n+3 order of growth: n(linear)

How do we compare?

C< log log n < log n < n^(1/3) < n^(1/2) < n < n² < n³ < n⁴ < 2^n < n^n

BIG O NOTATION: (Upper bound on order of growth)

Definition:

We say f(n) = O(g(n)) if and only if there exists constant c and n0 such that

f(n) <= c g(n) for all n >= n0.

EXAMPLE: f(n) = 2n + 3 can be written as O(n)

SOLUTION:

An easy way to find the constant c:

Take the constant with the highest growing term (here it is 2n, which is 2) and add 1 to it (here it is 2+1 = 3).

c g(n) = n since the order of growth is

f(n) <= c g(n) for all n >= n0

(2n+3) <= c n

(2n+3) <= 3n

3 <= 3n-2n

3 <= n

Therefore, n0 = 3

OMEGA NOTATION: (lower bound)

Opposite of big O notation, i.e., used to provide the lower bound.

Definition:

f(n) = Ω (g(n)) if and only if there exist positive constants c and n0 such that

0<=c g(n) <= f(n) for all n>=n0.

EXAMPLE: f(n) = 2n + 3

The constant c value in omega notation is less than the constant of the largest value in the equation. Here the constant of n is 2, so a value less than 2 is considered a c, so we take c=1.

c g(n) = n since the order of growth is

SOLUTION:

C g(n) <= f(n)

n <= 2n+3

n0 = 0

All values greater than or equal to 0 will give a higher value than n.

THETA NOTATION:

Combination of both big O and theta Notation.

Definition:

F(n) = Θ (g(n)) if and only if there exists constants c1,c2 (where c1 > 0 and

c2 > 0 ) and n0 (where n0 >= 0 ) such that

c1 g(n) <= f(n) <= c2 g(n) for all n>= n0

EXAMPLE: f(n) = 2n + 3

SOLUTION: constant c1 is based on theta, and c2 on big O. C1 = 1, c2 = 3

c1 g(n) <= f(n) <= c2 g(n)

1n <= (2n+3)

n0 = 0

(2n+3) < = 3n

n0 = 3

1n <= (2n+3) < = 3n

From both we conclude n0 = 3 for both to be true