* 1. Big Oh
  2. Little Oh

***Formal Definition*** (*c and n0 are fixed positive numbers*):

∀

*Definition* *2*:

***Example***: Unlike Big Oh, o describes an algorithm that is *not* asymptotically “tight”. At some point the growth of g(x) dwarfs f(x) to such an extent that f(x) becomes insignificant. Thus, the only “real” difference (it is a major one) is that Big allows f(x) and g(x) to grow at the same rate.

For the function , its

***Proof by Definition***: using Def 2, where f(x) = 2x + 1 and g(x) = x2

Because the limit is 0, we have demonstrated

1. Big Omega

***Formal Definition*** (*c and n0 are fixed positive numbers*):

***Example***: is a description of a functions asymptotic lower bound, i.e. the best-case running time for an algorithm.

For the function , its best case is

***Proof by Definition***:

We will show for some constant c, and

First divide each side of the inequality by :

=

Simplify

For and c = 1, holds true:

=

1. Little Omega

***Formal Definition***:

***Example***: is the inverse to Little Oh, and like Little Oh, it complements Big Omega in that it describes a lower bound that is not asymptotically tight.

For the function , its

***Proof by Definition***: is

Because the limit is , we have shown

1. Big Theta

***Formal Definition*** (*c1, c2 and n0 are fixed positive numbers*):

***Example***: Describes the worst-case scenario for an algorithm. For the function , its worst case is .

***Proof by Definition***: is

We know the above to be true (using the definition of Big Theta which states:

We can also prove it by the first definition:

Find constants *c1, c2* using f(x) and g(x), where g(x) = x2

Simplify by dividing by across the inequality.

Simplify

For n0 = 0 and settingx = 1, c2 = 4 and c1 = 1 we see that our definition holds.

a = 8

b = 2

f(n) =

nlog28 = n3

n3 > 3n2. f(n) is asymptotically larger than nlogba so case 1 applies.

**Answer:**

a = 3

b = 3

f(n) =

nlog33 = n

is asymptotically larger than n, but not polynomially larger. \*Probably need to take this further. Currently can only say it falls between case 2 & 3.

\*NOTE: U1 stands for unwind 1, U2 unwind 2, etc.

**U1)** Let and substitute in T(n)

Simplify

Repeat using from **U1** (unwind 1) *.*

**U2)** Begin by letting and substituting into T(n)

Simplify

Pattern

Repeat until (in which T becomes the constant T(1)), which makes k = log3n

= 3log3n +log3n

= 3log3n +log3n

Ignore T(1). It is a constant and we do not care about its value at substantially large values of n.

= 3log3n +nlog3n

**Answer** = O(nlogn)