***DIFFERENCE BETWEEN BIG O AND BIG OMEGA***:

The difference between  **Big** O **notation** and  **Big** Ω **notation** is that **Big** O is **used** to describe the worst case running time for an algorithm. But, **Big** Ω **notation**, on the other hand, is **used** to describe the best case running time for a given algorithm.

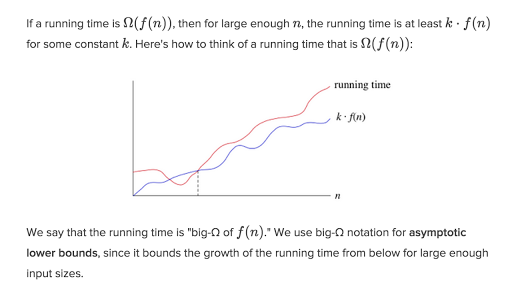
***OMEGA NOTATION:***

**Omega Notation** (Ω-**notation**) **Omega notation** represents the lower bound of the running time of an algorithm. Thus, it provides the best case complexity of an algorithm.

The Lower and Upper Bound Theory provides a way to find the lowest complexity algorithm to solve a problem.

***Lower Bound***   
 Let L(n) be the running time of an algorithm A(say), then g(n) is the **Lower Bound** of A if there exist two constants C and N such that L(n) >= C\*g(n) for n > N. Lower bound of an algorithm is shown by the asymptotic notation called [Big Omega](https://www.geeksforgeeks.org/analysis-of-algorithms-set-3asymptotic-notations/) (or just Omega).

***Upper Bound***    
 Let U(n) be the running time of an algorithm A(say), then g(n) is the **Upper Bound** of A if there exist two constants C and N such that U(n) <= C\*g(n) for n > N. Upper bound of an algorithm is shown by the asymptotic notation called [Big Oh(O)](https://www.geeksforgeeks.org/analysis-algorithms-big-o-analysis/) (or just Oh).

******

# Take, for example, a function that searches an array for the value 0:

def containsZero(arr): #assume normal array of length n with no edge cases

for num x in arr:

if x == 0:

return true

return false

What’s the best case?

Well, if the array we give it has 0 as the first value, it will take constant time: Ω (1)

What’s the worst case?

If the array doesn’t contain 0, we will have iterated through the whole array: O(n)

Let’s change our code a bit.

def printNums(arr): #assume normal array of length n with no edge cases

for num x in arr:

print(x)

Can you think of a best case and worst case??

I can’t! No matter what array we give it, we have to iterate through every value in the array.

So the function will take AT LEAST n time (Ω(n)), but we also know it won’t take any longer than n time (O(n)).

What does this mean?

 Our function will take **exactly** n time: Θ(n).

|  |  |  |
| --- | --- | --- |
| **Big Oh** | **Big Omega** | **Big Theta** |
| It is like <=  rate of growth of an algorithm is less than or equal to a specific value | It is like >=  rate of growth is greater than or equal to a specified value | It is like ==  meaning the rate of growth is equal to a specified value |
| The upper bound of algorithm is represented by Big O notation. Only the above function is bounded by Big O. asymptotic upper bond is it given by Big O notation. | The algorithm’s lower bound is represented by Omega notation. The asymptotic lower bond is given by Omega notation | The bounding of function from above and below is represented by theta notation. The exact asymptotic behavior is done by this theta notation. |
| Big oh (O) – Worst case | Big Omega (Ω) – Best case | Big Theta (Θ) – Average case |
| Big-O is a measure of the longest amount of time it could possibly take for the algorithm to complete | Big- Ω is take a small amount of time as compare to Big-O it could possibly take for the algorithm to complete. | Big- Θ is take very short amount of time as compare to Big-O and Big-? it could possibly take for the algorithm to complete. |
| Mathematically – Big Oh is 0 <=f(n) <= c g(n) for all n>=n0 | Mathematically – Big Omega is O<= C g(n) <= f(n) for all n>=n 0 | Mathematically – Big Theta is O<=C 2 g(n)<=f(n)<=C 1 g(n) for n>=n 0 |
|  |  |  |
|  |  |  |