Analysis of algorithm :

There are three asymptotic notation there to represent the order of growth.

Big O : Exact or upper

Theta : Exact ( we are using this till now)

Big Omega : Exact or lower

Asymptotic notation:

Whenever we are talking about asymptotic notation, the graphs are always drawn in first quadrant as both n (input) and time taken will always be positive.

Big O notation – exact or upper bound ( commonly used to refer worst case)

Theta notation – Exact bound (commonly used to refer average case)

Big Omega – Exact or lower bound (commonly used to refer best case)

Simply,

best case ==> Bogus

average case ==> inpractical in many scenarios

worst case ==> will be used in time complexity

These asymptotic notations are used in order to represent the order of growth in different types of input. There may be algorithm specific best, worst and average cases and depend on these cases, the order of growth will change.

We may that the algorithm will take n times in worst scenario. This is why we explain about the order of growth in one line. As asymptotic notation are the mathematical way of expressing this case specific algorithm to express the order of growth.

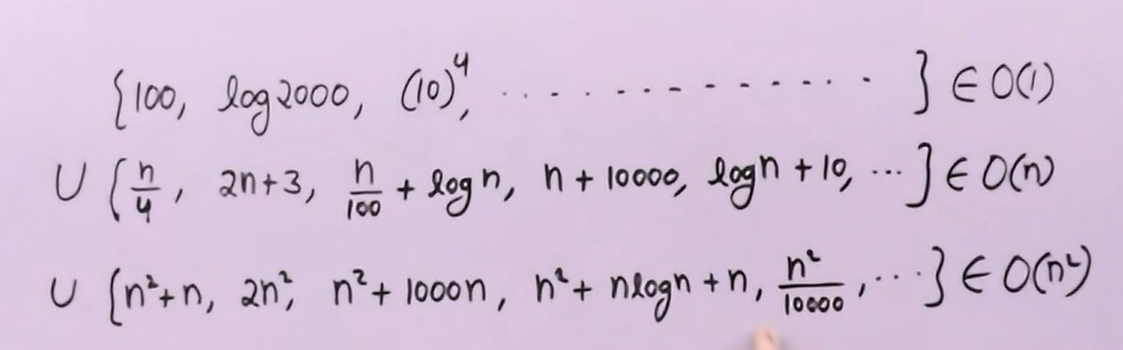
O(n) is the order of growth (or) in the terms of theta.

theta(n) in worst case are both the same.

Big O Notation :

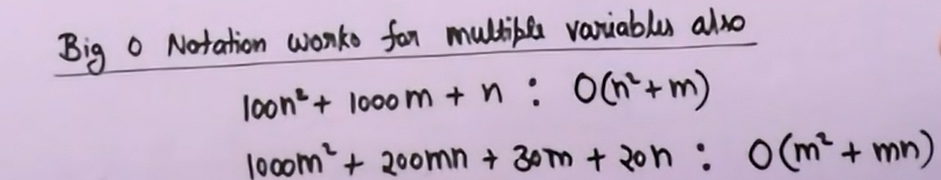
Ignoring lower order terms

Ignoring leading constant.



The first series may be written as O(n) or O(n2) but if we know the exact order of growth, we may use the theta notation.

Big O notation for multiple variables



If we are finding time complexity of array traversal, instead of saying it in upper bound, we will say it in theta bound as theta(n) as we know also the input for time complexity. But when the logics are complex, we may get the innermost statement and after finding certain thing, we may write the worst case based on how many times that innermost statement going to run.

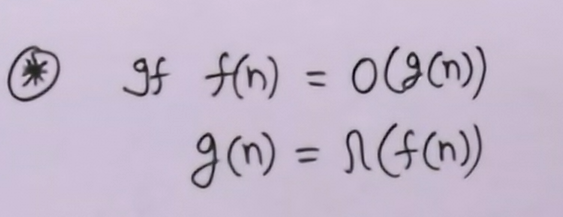
When we didn’t know the exact bound and with something, we found the upper bound, then we may use the same. Or if we found the worst case, we may use the same for denoting in Upper bound.

Big Omega :

WE can use Big Omega or Big O or Theta when we know the exact bound. On the direct method, using those two principles, we may found the exact bound and in this case, we can use any of three notation whereas the Theta notation is preferred.

The omega notation will be used when there is a game program where the program runs until the user exit the game. It’s a infinite game so that there won’t be any upper bound or exact bound. We only have the option to found the lower bound of the program. This program have to initialise all the n players score ( previous score). Then we may say that it’s order of growth which in turn determines the time complexity will be omega(n)

From the above things, when it comes to the logic, literally for all the algorithm, we may right it as omega(1).

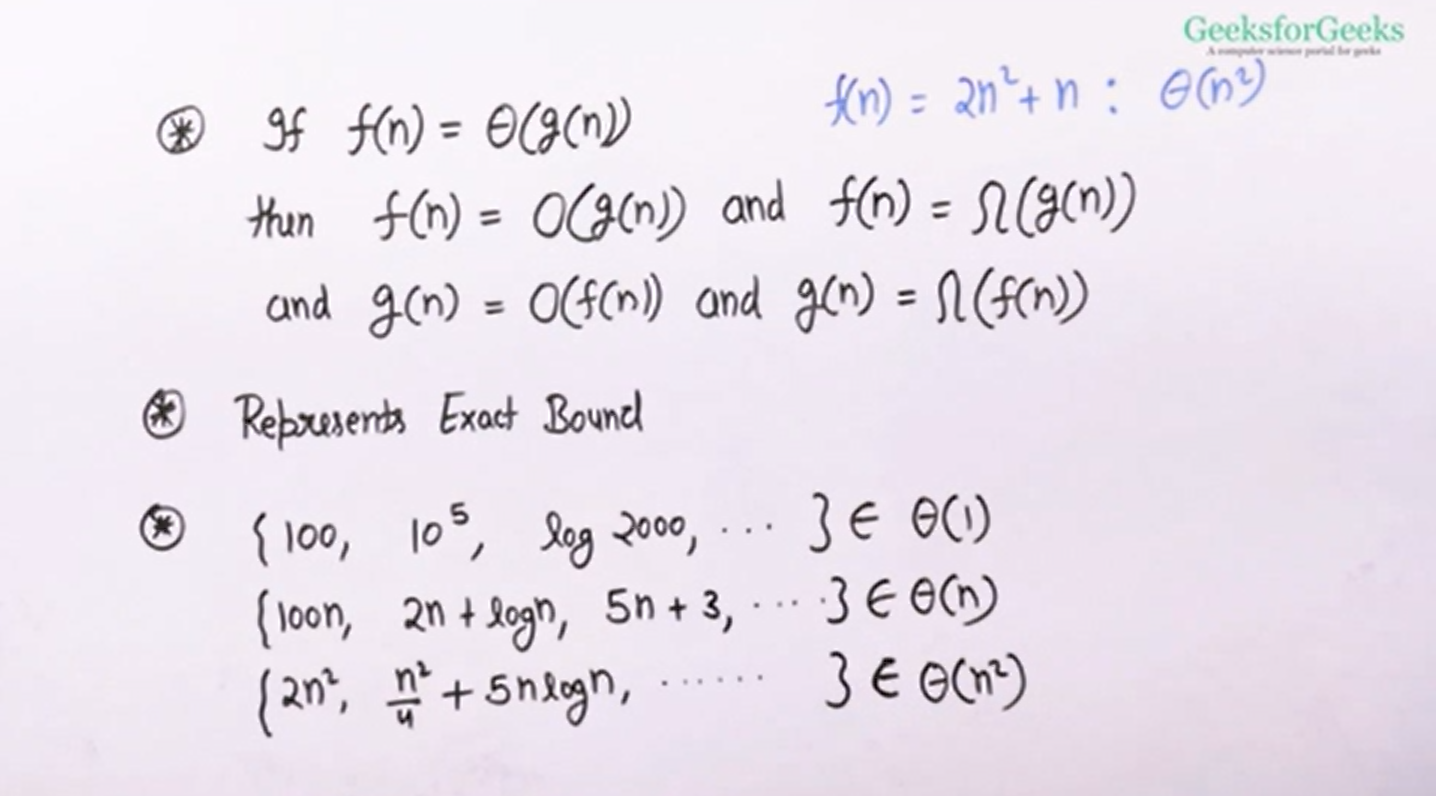


Theta notation :

Whenever we know the exact bound, prefer to use the theta notation.

When the functions are expressed mathematically as equations without any if condition, then we may find the exact bound and with this exact bound we may right all the three types of notation.

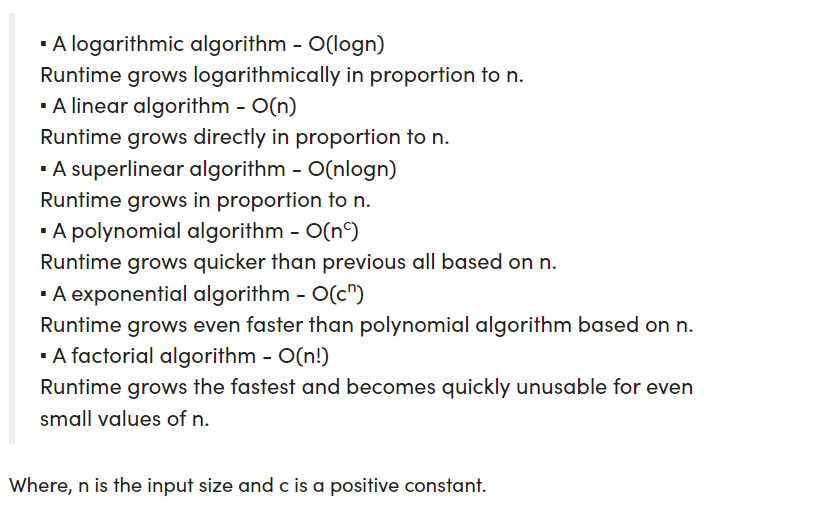
Big O and Big Omega notation comes into picture in the case where we are unable to find out the exact bound of the algorithm. It’s only be essential for the complex algorithm where we are unable to find the exact bound.

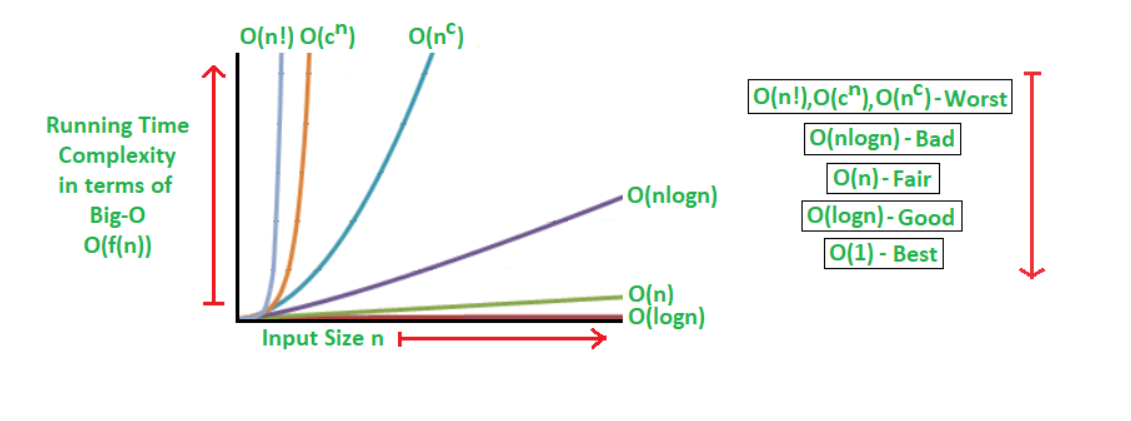


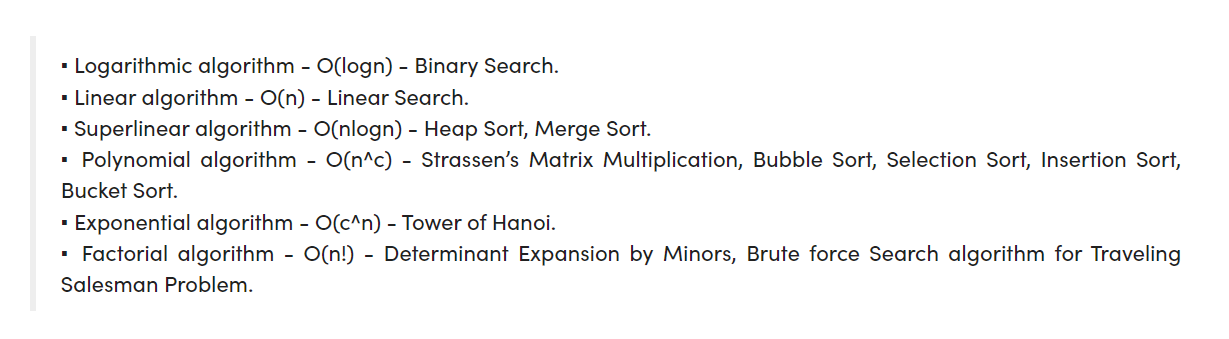
On insertion sort, it will take n times when the array is sorted and takes n^2 when the array is not sorted. Instead of saying it as Big O(n^2). We may say that it’s theta(n) in best case and it’ll take theta(n^2) in the worst case. When we are saying Big O(n^2), it covers the best case and also covers the worst case.

Analysis of Recursion (Introduction):

Inorder to analyse the recursive function, first we have to write a recurrence relation for it.







Memory footprint Analysis of Algorithms (Space Complexity)

