**Lower Bound**

The Lower Bound Theory provides a way to find the lowest complexity algorithm to solve a problem. Before understanding the theory, first let’s have a brief look on what actually Lower bounds are.

* **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 (or just Omega).

**Lower Bound Theory:**  
According to the lower bound theory, for a lower bound L(n) of an algorithm, it is not possible to have any other algorithm (for a common problem) whose time complexity is less than L(n) for random input. Also every algorithm must take at least L(n) time in worst case. **Note** that L(n) here is the minimum of all the possible algorithm, of maximum complexity.

The Lower Bound is a very important for any algorithm. Once we calculated it, then we can compare it with the actual complexity of the algorithm and if their order are same then we can declare our algorithm as optimal. So in this section we will be discussing about techniques for finding the lower bound of an algorithm.

**Note** that our main motive is to get an optimal algorithm, which is the one having its Upper Bound Same as its Lower Bound (U(n)=L(n)). Merge Sort is a common example of an optimal algorithm.

**Trivial Lower Bound –**  
It is the easiest method to find the lower bound. The Lower bounds which can be easily observed on the basis of the number of input taken and the number of output produces are called Trivial Lower Bound.

**Example:** Multiplication of n x n matrix, where,

*Input*: For 2 matrix we will have 2n2 inputs

*Output*: 1 matrix of order n x n, i.e., n2 outputs

In the above example it’s easily predictable that the lower bound is O (n2).