* Some useful Rules for Algorithm Analysis.
* To analyze a function analyze each statement in the function. The statement that has the greatest complexity determines the order of the function. Some helpful rules are being given for your reference:
  1. **Addition Rule** "You can combine sequences of statements by using the addition rule, which states that the running time of a sequence of statements is just the maximum of the running times of each individual statement.
  2. **Multiplication Rule** This is used to determine the running time for a loop. "You need to determine the running time for the loop's body and the number of times the loop iterates. Frequently you can just multiply the running time of the body by the number of iterations.”
* An algorithm is a sufficiently precise method or procedure for accomplishing a specific task, which can be programmed on computer.
* Complexity refers to the measure of the performance of an algorithm.
* Complexity can be related to time (temporal complexity) or to space (space complexity).
* There are five guidelines for finding out the time complexity of a piece of code are:
  1. Loops
  2. Nested loops
  3. Consecutive statements
  4. If-then-else statements
  5. Logarithmic complexity
* **Big-O** notation is used to depict an algorithm's growth rate i.e., change in algorithm performance when its input size grows.  
  Some Growth rates of algorithms are as shown in following table:

|  |  |  |
| --- | --- | --- |
| Time complexity | | Example |
| O(1) | constant | Adding to the front of a list. |
| O(log N) | log | Finding an entry in a sorted array |
| O(N) | linear | Finding an entry in an unsorted array |
| O(N log N) | n log n | Sorting n items by 'divide-and-conquer' |
| O(N2) | quadratic | Shortest path between two nodes in a graph |
| O(N3) | cubic | Simultaneous linear equations |
| O(2N) | exponential | The Towers of Hanoi problem |

* Dominant term is the one which affects the most, an algorithm's performance.
* Only the dominant term is included in big-O notation.
* The **Worst-case** complexity provides an upper-bound on running time.
* **Average-Case** complexity provides expected running time.
* **Best-Case** complexity provides the time of optimal performance.