**O(1) time**

* Accessing Array Index (int a = ARR[5];)
* Inserting a node in Linked List
* Pushing and Poping on Stack
* Insertion and Removal from Queue
* Finding out the parent or left/right child of a node in a tree stored in Array
* Jumping to Next/Previous element in Doubly Linked List

**O(n) time**

In a nutshell, all Brute Force Algorithms, or Noob ones which require linearity, are based on O(n) time complexity

* Traversing an array
* Traversing a linked list
* Linear Search
* Deletion of a specific element in a Linked List (Not sorted)
* Comparing two strings
* Checking for Palindrome
* Counting/Bucket Sort and here too you can find a million more such examples....

**O(log n) time**

* Binary Search
* Finding largest/smallest number in a binary search tree
* Certain Divide and Conquer Algorithms based on Linear functionality
* Calculating Fibonacci Numbers - Best Method The basic premise here is NOT using the complete data, and reducing the problem size with every iteration

**O(n log n) time**

The factor of 'log n' is introduced by bringing into consideration Divide and Conquer. Some of these algorithms are the best optimized ones and used frequently.

* Merge Sort
* Heap Sort
* Quick Sort
* Certain Divide and Conquer Algorithms based on optimizing O(n^2) algorithms

**O(n^2) time**

These ones are supposed to be the less efficient algorithms if their O(nlogn) counterparts are present. The general application may be Brute Force here.

* Bubble Sort
* Insertion Sort
* Selection Sort
* Traversing a simple 2D array

**O(n!) time**

* Solving the travelling salesman problem via brute-force search
* generating all unrestricted permutations of a partially ordered set;
* finding the determinant with Laplace expansion
* enumerating all partitions of a set