O(1) time

1. Accessing Array Index (int a = ARR[5];)

2. Inserting a node in Linked List

3. Pushing and Poping on Stack

4. Insertion and Removal from Queue

5. Finding out the parent or left/right child of a node in a tree stored in Array

6. Jumping to Next/Previous element in Doubly Linked List

and you can find a million more such examples...

O(n) time

1. Traversing an array

2. Traversing a linked list

3. Linear Search

4. Deletion of a specific element in a Linked List (Not sorted)

5. Comparing two strings

6. Checking for Palindrome

7. Counting/Bucket Sort

and here too you can find a million more such examples....

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

O(log n) time

1. Binary Search

2. Finding largest/smallest number in a binary search tree

3. Certain Divide and Conquer Algorithms based on Linear functionality

4. Calculating Fibonacci Numbers - Best Method

The basic premise here is NOT using the complete data, and reducing the problem size with every iteration

O(nlogn) time

1. Merge Sort

2. Heap Sort

3. Quick Sort

4. Certain Divide and Conquer Algorithms based on optimizing O(n^2) algorithms

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.

O(n^2) time

1. Bubble Sort

2. Insertion Sort

3. Selection Sort

4. Traversing a simple 2D array

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.