# 4. Data Structures and Algorithms :: Orders of Growth and Big O Notation

Commonly encountered orders of growth include:

| **Order of Growth** | **Common Name** |
|---|---|
| $O(1)$ | Constant complexity |
| $O(lg\ n)$ | Logarithmic complexity |
| $O(n)$ | Linear complexity |
| $O(n\ lg\ n)$ | Linearithmic complexity (<u>line</u>ar + loga<u>rithmic</u>) |
| $O(n^2)$ | Quadratic complexity |
| $O(n^3)$ | Cubic complexity |
| $O(2^n)$ | Exponential (power of 2) |

These are ranked from slowest to fastest growth.

## 4. Data Structures and Algorithms :: Linear Search Time Complexity

Let's analyze the asymptotic time complexity of the linear search algorithm (written in pseudocode):

```
Begin linearSearch(In: List<T> list; In: T key) → Int
  For i := 0 to list.size() - 1 Do
    If list_i = key Then
      Return i
    End If
  End For
  Return NOT_FOUND
End linearSearch
```

The first step in analyzing the **worst case time complexity** of any algorithm is to determine what the **key operation** is (or key operation*s*) and then to **determine a function** which counts the **maximal number** of times the key operation is performed as a **function of the problem size**.

For *linearSearch* the **key operation** is the comparison of $list_i$ to *key* and in the worst case we will have to walk through the entire list before determining that *key* is not in the list. Thus:

| List Size | Max Comparisons |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| ... | ... |
| $n$ | $n$ |

# 4. Data Structures and Algorithms :: Linear Search Time Complexity (continued)

Let $c(n)$ be the maximum number of comparisons that we make for a list of size $n$. It is clear that $c(n) = n$.

The next step in analyzing *linearSearch* is to determine the order of growth of $c(n)$, that is, to determine if *linearSearch* is $O(1)$, $O(n)$, $O(n^2)$, etc. In other words, what would $g(n)$ need to be such that $c(n) \leq g(n)$?

Clearly $g(n) = n$ would suffice, so we would say that the asymptotic time complexity of *linearSearch* is $O(n)$.

## 4. Data Structures and Algorithms :: Orders of Growth and Big O Notation

You may notice that if we chose $g(n) = n^2$ then we would also have $c(n) \leq g(n)$ in which case we could say that the asymptotic time complexity of *linearSearch* is $O(n^2)$.

Although this is technically true—*linearSearch* will never require more than $n^2$ comparisons—it would be semantically incorrect. In analysis of algorithms when determining the worst case behavior of an algorithm, we are interested in determining the **smallest** or **slowest growing** function $g(n)$ that is equal to or exceeds $c(n)$.

If we did not make that rule, then analyzing almost every algorithm would be easy because we could claim with almost absolute certainty that any particular algorithm $A$ is $O(n!$ to the $n!$ to the $n!$ ... to the $n!)$ where "to the $n!$" is repeated $n!$ times!