3. Data Structures and Algorithms :: Asymptotic Time Complexity

The approach we use in computer science to analyze the time complexity of an algorithm while avoiding the confounding variables is:

1. We write the algorithm in pseudocode.
2. Since the algorithm is expressed in pseudocode we do not have to be concerned with compiling it.
3. The skill of the programmer is avoided because we are not implementing the algorithm in a HLL.
4. We analyze the efficiency without actually running the algorithm on a real machine,

The key to this **abstract view** of the time complexity of the algorithm is to focus not on how many seconds the algorithm takes to complete, but rather on **operation counts** (i.e., we count how many times certain operations are performed), and in particular, on operation counts relative to the size of the input.

Of course, this analysis will not tell us how much physical time the algorithm will take to solve the problem, but as we discussed, measuring physical time is fraught with problems.

Rather, what we obtain is an **estimate** of the time and this estimate is very useful in comparing the efficiencies of related algorithms, e.g., when comparing the times for sorting algorithms to sort a list.

3. Data Structures and Algorithms :: Asymptotic Time Complexity (continued)

For example, suppose we have two sorting algorithms: A and B.

We decide to measure the time complexity of both algorithms by counting the number of times we move an element from one location in the list to another location.

Suppose for A the number of moves is roughly proportional to n^2 where n is the size of the list being sorted. For B we determine that the number of moves is roughly proportional to n.

Which algorithm is more efficient?

3. Data Structures and Algorithms :: Asymptotic Time Complexity (continued)

If we assume that the problem size n goes to infinity then we can derive a function which counts how many times key operations are performed relative to n. This measures what is called the **asymptotic time complexity** of the algorithm.

The most common asymptotic time complexity measurement is to determine the **worst case performance** of the algorithm (the others are best case and average case), which gives us an **upper bound** on the time, i.e., the algorithm will never take more time to execute than that predicted by the upper bound.

The upper bound is denoted as $O(g(n))$—pronounced "big oh of g of n"—where $g(n)$ is a function and O refers to the "order" of the growth of the function.

This representation is known as **Big O notation**.