Comparing Algorithms

Algorithms (including search algorithms) can be compared on

- the time taken,
- the space used, and
- the quality or accuracy of the results.

The time taken, space used, and accuracy of an algorithm are a function of the inputs to the algorithm. Computer scientists talk about the **asymptotic complexity** of algorithms, which specifies how the time or space grows with the input size of the algorithm. An algorithm has time (or space) complexity $O(f(n))$– read "big-oh of $f(n)$)" – for input size n, where $f(n)$ is some function of n, if there exist constants n_0 and k such that the time, or space, of the algorithm is less than $k * f(n)$ for all $n > n_0$. The most common types of functions are exponential functions such as 2^n, 3^n, or 1.015^n; polynomial functions such as n^5, n^2, n^n, or $n^{1/2}$; and logarithmic functions, $\log n$. In general, exponential algorithms get worse more quickly than polynomial algorithms which, in turn, are worse than logarithmic algorithms.

An algorithm has time or space complexity $\Omega(f(n))$ for input size n if there exist constants n_0 and k such that the time or space of the algorithm is greater than $k*f(n)$ for all $n>n_0$. An algorithm has time or space complexity $\Theta(f(n))$ if it has complexity $O(f(n))$ and $\Omega(f(n))$. Typically, you cannot give a $\Theta(f(n))$ complexity on an algorithm, because most algorithms take different times for different inputs. Thus, when comparing algorithms, one has to specify the class of problems that will be considered.

Algorithm A is better than B, using a measure of either time, space, or accuracy, could mean any one of:

- the worst case of A is better than the worst case of B
- A works better in practice, or the average case of A is better than the average case of B, where you average over typical problems
- there is a subclass of problems for which A is better than B, so that which algorithm is better depends on the problem
- for every problem, A is better than B.

The worst-case asymptotic complexity is often the easiest to show, but it is usually the least useful. Characterizing the subclass of problems for which one algorithm is better than another is usually the most useful, if it is easy to

determine which subclass a given problem is in. Unfortunately, this characterization is usually very difficult to obtain.
Characterizing when one algorithm is better than the other can be done either theoretically using mathematics or empirically by building implementations. Theorems are only as valid as the assumptions on which they are based. Similarly, empirical investigations are only as good as the suite of test cases and the actual implementations of the algorithms. It is easy to disprove a conjecture that one algorithm is better than another for some class of problems by showing a counterexample, but it is usually much more difficult to prove such a conjecture.