Classifying algorithms

An algorithm is **efficient** if its running-time, as a function of its input size, is bounded above by a polynomial (e.g., n^k for some constant k).

Some running times bounded by a polynomial: $\lg n$ (since $\lg n < n^k$), $\sqrt{n} = n^{1/2}$, n, n^2, $n^{\lg 7}$

Example: mergeSort is efficient because its running-time is $O(n \lg n)$ and $n \lg n < n * n = n^2$, so it is bounded above by a polynomial

Example: Karatsuba is efficient because its running-time is $O(n^{\lg 3})$ and $n^{\lg 3} \approx n^{1.585}$, a polynomial

Be able to give some other examples of efficient algorithms.

An algorithm is **inefficient** if its running-time is not bounded by a polynomial.

Some non-polynomial running times: 2^n, $n!$, n^n

Example: The backtracking algorithm to solve the Hamiltonian Cycle Problem is inefficient because its running time is $O(n!)$

Example: The backtracking algorithm to solve the Knapsack Problem is inefficient because its running time is $O(2^n)$

Be able to give some other examples of inefficient algorithms.

Classifying computational problems

An algorithm **solves** a computational problem if it correctly assigns to each instance (input) a corresponding solution. A problem is **solvable** if there exists an algorithm to solve it.

Example: Multiplication of Large Integers is solvable because the Karatsuba algorithm solves the problem

Example: Traveling Salesperson Problem is solvable because the backtracking algorithm solves the problem

Be able to give some other examples of solvable problems. For each one, name an algorithm that solves the problem.

A problem is **unsolvable** if there is no algorithm possible to solve it.

NOTE: The term "unsolvable" is different from the term "unsolved". Unsolvable means it has been proven that no algorithm is ever possible to solve a problem. Unsolved is a term that means "the search is still on" to either find a solution or prove that no solution exists.

Give an example of an unsolvable problem. (Hint: There is a really famous one!)

For problems that **<u>are</u>** solvable, problems are classified according to the efficiency of an algorithm to solve the problem.

A computational problem is **feasible** or **tractable** if there is an <u>efficient</u> algorithm to solve it.

Example: Matrix Multiplication is feasible (or tractable) because the Strassen algorithm solves the problem in time $O(n^{\lg 7})$

Example: Closest Pair of Points Problem is tractable because brute force solves the problem in time $O(n^2)$.

A problem is **intractable** if there is no efficient algorithm that solves the problem for all inputs. The problem can be solved "in theory" - given an infinite amount of time. But any algorithm that solves the problem would take too long for all practical purposes.

Example: The Towers of Hanoi Problem is intractable because it has been proven that any algorithm to solve it has a worst case running time of $O(2^n)$.

Example: We **<u>believe</u>** that the Traveling Salesperson Problem is intractable because there is no known efficient algorithm to solve it. (It seems unlikely that we will find an efficient algorithm, but this has not been proven yet.)