Limitations of Algorithm Power

Introduction: Algorithms are very powerful problem-solving tools when they are executed by computers. Even then there are some limitations of algorithm power. The power of algorithms is limited because of the following reasons:

- Some problems cannot be solved by any algorithm.
- Some problems can be solved algorithmically not in polynomial time.
- Even though solution exists for a problem in polynomial time, there are lower bounds on the efficiency of algorithms i.e., efficiency of the algorithm cannot be improved further.

Majority of the algorithms we have studied so far are polynomial-time algorithms i.e., for all inputs of size n, the worst-case time complexity is $O(n^k)$ for any constant k. For example, all sorting and searching techniques, BFS and DFS traversal etc.

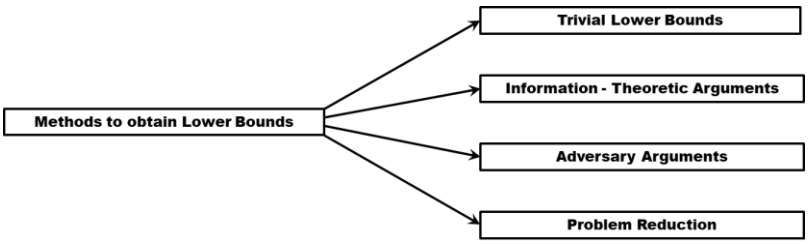
There are problems that can be solved, but not in time $O(n^k)$ for any constant k. For example, the algorithms such as Traveling sales man problem whose time complexity is (n-1!), tower of Hanoi whose time complexity is $O(2^n)$ cannot be solved in polynomial-time.

Lower bound arguments: The efficiency of an algorithm can be looked into using two different ways:

- By establishing the asymptotic efficiency class i.e., obtaining the upper bound which is expressed using Big-oh notation and compare the upper bound with other class of algorithms. For example, compare bubble sort algorithm whose time efficiency is $O(N^2)$ and tower of Hanoi problem whose time efficiency is $O(2^n)$. Comparing these two time efficiencies we know that bubble sort algorithm is efficient when compared to tower of Hanoi problem. This comparison has to be avoided because these two problems solve different problems. By establishing the efficiency class of solving the same problem with different methods. For example, time efficiency of bubble sort is $O(N^2)$ whereas the time efficiency of quick sort is N log N. By comparing these two classes of efficiencies we know that quick sort works faster.

To ascertain the efficiency of an algorithm with respect to other algorithms for the same problem, we should know the best efficiency or the lower bound

Definition: Lower bound is defined as estimating the minimum amount of time needed to solve a given problem. If we have an algorithm with the lower bound, we can look for other improvements in the algorithm to improve the efficiency further. The various methods of obtaining the lower bound are shown below:



Trivial lower bounds: This is the simplest method of obtaining the lower bound. This is normally obtained by looking into input and output parameters i.e.

- The number of items in the input that must be processed
- The number of outputs that have to be produced

For example, while generating the permutations, the input size is n and the number of distinct outputs will be n!. So, the time efficiency is Ω(n!). The various limitations using trivial bound approach are shown below:

- Trivial lower bounds are often not useful. For example, in a traveling salesman problem, given n cities and n (n-1)/2 distances, the output will be a list of n+1 cities, with time efficiency of $\Omega(n^2)$. There is no other algorithm to compare with this time efficiency.
- Another problem is which part of the input must be processed. For example, in binary search problem, the elements are arranged in ascending order. We have seen that there is no need to search for the entire using this technique.

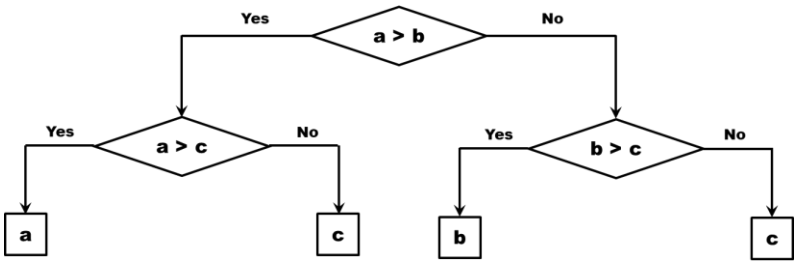
Information-Theoretic arguments: In this approach, the lower bound is based on the amount of information it has to produce. This method is very useful to find the lower bounds for many problems involving searching, sorting etc. The lower bound using this method can be obtained very easily and precisely using decision trees.

Adversary argument: The adversary method is another way of obtaining the lower bound. In this method, the lower bound is obtained by measuring the amount of work required to reduce a set of potential inputs to a single-input along the most time-consuming path. For ex, in the game of guessing a number, the number of questions asked by the adversary must be at least log 2^n.

Problem reduction: In this strategy, we already know that a given problem is transformed into another problem for which the solution exists in the form of a known algorithm. The same idea can be used to find the lower bound also. Suppose, there is a problem P whose lower bound has to be computed. Assume, we have another problem Q whose lower bound is already known. If an instance of a problem P can be transformed into an instance of Q for which solution exists, then lower bound of Q is the lower bound of P. In this way, the lower bound of one instance of a problem can be obtained using the lower bound of another instance of a problem.

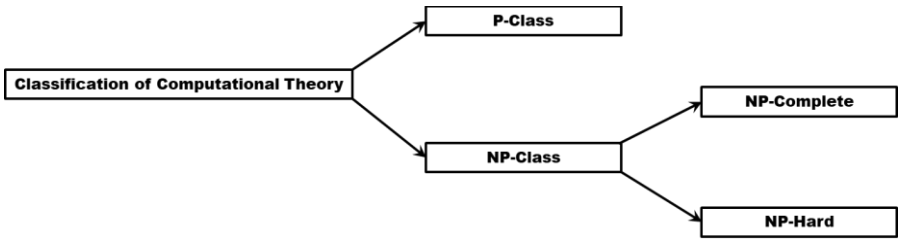
Decision Trees: The various algorithms such as sorting and searching work only by comparing the given elements. The performance of these algorithms that involve comparison can be obtained using decision trees.

A decision tree is a binary tree that represents only the comparisons of given elements in the array while sorting or while searching. In the algorithm, the control transfer, data movement and all other aspects of the algorithm are ignored. In the decision tree, internal nodes represent the comparisons made between the items and leaf nodes represent the result of the algorithm. For example, the decision tree to find minimum of three numbers is shown below:



The decision tree for arranging three elements a, b and c in ascending order can be written in a similar way as shown above:

P, NP and NP-Complete Problems: The computational complexity theory deals with the resources required during computation to solve a given problem. The most common resources are time taken to solve a problem and space or memory required to solve a problem. The computational theory is classified into two groups as shown below:



P Class Definition: In this theory, the class P problem consists of decision problems that can be solved in polynomial time by deterministic algorithms. This class of problems is called polynomial (P). P includes only decision problems which are problems with yes/no answers. In other words, the problems are called P-problems. For example, searching and sorting problems.

NP-Class Definition: The problems that can be solved in polynomial time by non-deterministic algorithms are called NP problems. Here, NP means non-deterministic polynomial. In this theory, the class NP consists of all those decision problems whose positive solutions can be verified in polynomial time given the right information. The solution should be obtained in a polynomial time on a non-deterministic machine. The NP-Class problems are divided as NP-Complete and NP-Hard.

NP-Complete problem Definition: The NP-problems (non-deterministic polynomial time) are NP-Complete if their solutions can be verified quickly for correctness and if the same solving algorithm used can solve all other NP problems. For example, sum of subset problem.

NP-Hard problem Definition: NP-hard is a class of problems that are at least as hard as the hardest problem in NP. A problem is said to be NP-complete problem that can be solved in a polynomial time. For example, decision problems, search problems and optimization problems.