# UNIT 8. LIMITATIONS OF ALGORITHM POWER

In the previous chapters, various algorithms and design techniques (like brute force, divide and conquer etc) have been considered. But, every methodology and algorithm has limitations. Some problems cannot be solved by any algorithm. Some problems can be solved algorithmically, but not in a polynomial time. And few problems have lower bound for their efficiency. There are certain ways of establishing lower bounds on efficiency of algorithms.

## 8.1 Decision Trees

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences. Decision trees are helpful in establishing the lower bounds on efficiency of comparison-based algorithms like sorting and searching. Consider a decision tree given in Figure 8.1, which is used for finding minimum of three numbers. Here, each leaf represents the possible outcome of the algorithm. The work of the algorithm on a particular input of size can be traced by a path from the root to a leaf in its decision tree. Number of comparisons made by the algorithm is equal to the number of edges in that path. Hence, maximum number of comparisons required for the algorithm is equal to the height of the decision tree.

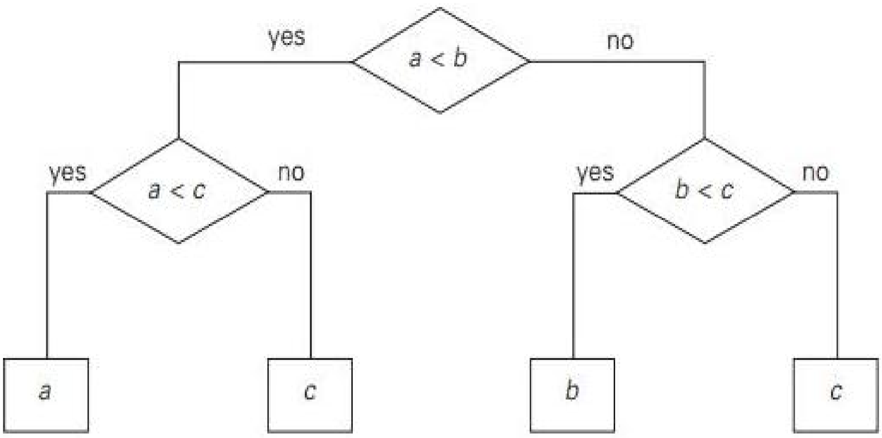


Figure 8.1 Decision tree for finding minimum of three numbers

**Decision Trees for Sorting Algorithms:**

Most of the sorting algorithms are based on comparison of elements in a given array. Hence, decision tree can be drawn for solving sorting problems. Selection sort is one of the sorting techniques based on comparison of elements. Figure 8.2 is a decision tree for applying selection sort on 3 elements viz. *a, b, c.*

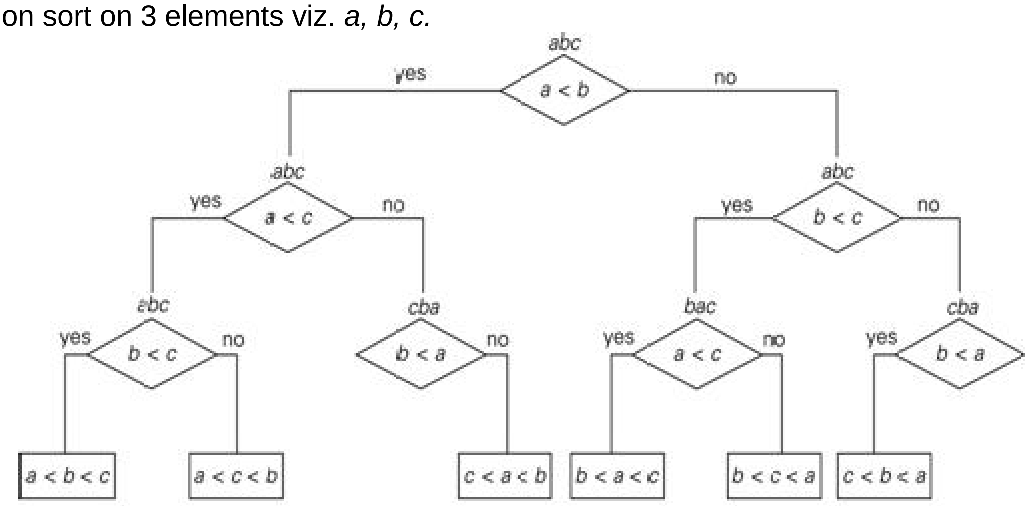


Figure 8.2 Decision tree for three-element selection sort

It can be observed that in the worst case, number of comparisons can be given as –

$$C(n) \geq \lceil \log_2 n \rceil$$

**Decision Tree for three-element Insertion sort:**
The figure 8.3 shows the decision tree for implementing insertion sort for an array of three elements.

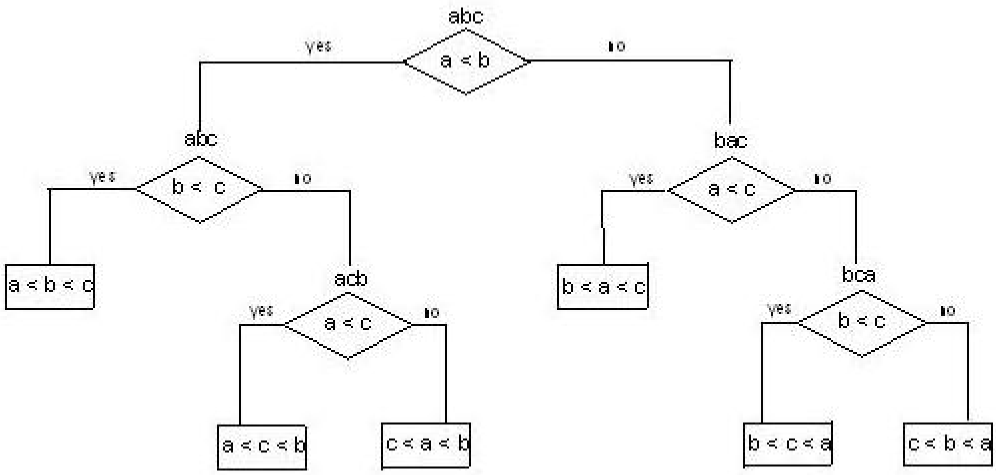


Figure 8.3 Decision tree for three-element insertion sort

## 8.2 *P*, *NP* and *NP – Complete* Problems

In the study of the computational complexity of the problems, the major concern is whether a given problem can be solved by some algorithm in a polynomial time or not.

We can say that an algorithms solves the problem in polynomial time if its worst-case time complexity is O(*p(n)*), where *p(n)* is a polynomial of input size *n.* The problems which are solvable in polynomial time are called as ***tractable*** and problems that cannot be solved in polynomial time are called as ***intractable.***

Class *P* is a class of decision problems that can be solved in polynomial time by deterministic algorithms. This class of problems is called ***polynomial.***

Class *NP* is the class of decision problems that can be solved by nondeterministic polynomial algorithms. This class of problems is called as ***Nondeterministic polynomial.***

Most of the decision problems are in *NP*. That is,

$$P \subseteq NP$$

But, *NP* also contains few decision problems like Hamiltonian circuit problem, travelling salesman problem, knapsack problem etc. This leads to the most important open question of theoretical computer science: ***Is P a proper subset of NP, or are they same?*** That is,

Whether *P = NP ?*

The meaning of P = NP implies that many combinatorial decision problems can be solved by a polynomial-time algorithm, though no such algorithm is invented till today. Moreover, many well-known decision problems are known to be ***NP-Complete.*** A decision problem *D* is said to be ***NP-Complete*** if,

- It belongs to class *NP*
- Every problem in *NP* is polynomially reducible to *D*.

A Decision problem *D1* is said to be ***polynomially reducible*** to a decition problem *D2,* if there exists a function *t* that transforms instances of *D1* to instances of *D2* such that

- *t* maps all yes instances of *D1* to yes instances of D2 and all no instances of D1 to no instances of D2
- *t* is computable by a polynomial-time algorithm

Informally, an NP-complete problem is a problem in NP that is as difficult as any other problem in this class, because, any other problem in NP can be reduced to it in a polynomial time as shown in Figure 8.4. In this diagram, arrows indicate the polynomial-time reductions of NP problems to an NP-complete problem.

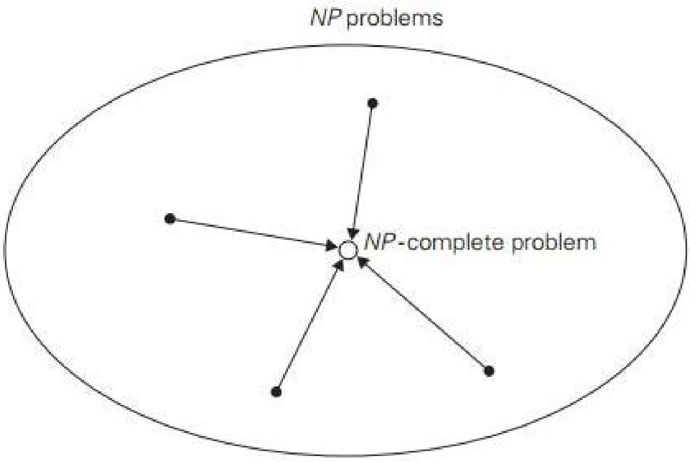


Figure 8.4 Notion of an NP-complete problem.