MODULE 5: LIMITATIONS OF ALGORITHMIC POWER

Decision Trees, P, NP, and NP-Complete Problems

LIMITATIONS OF ALGORITHMIC POWER:

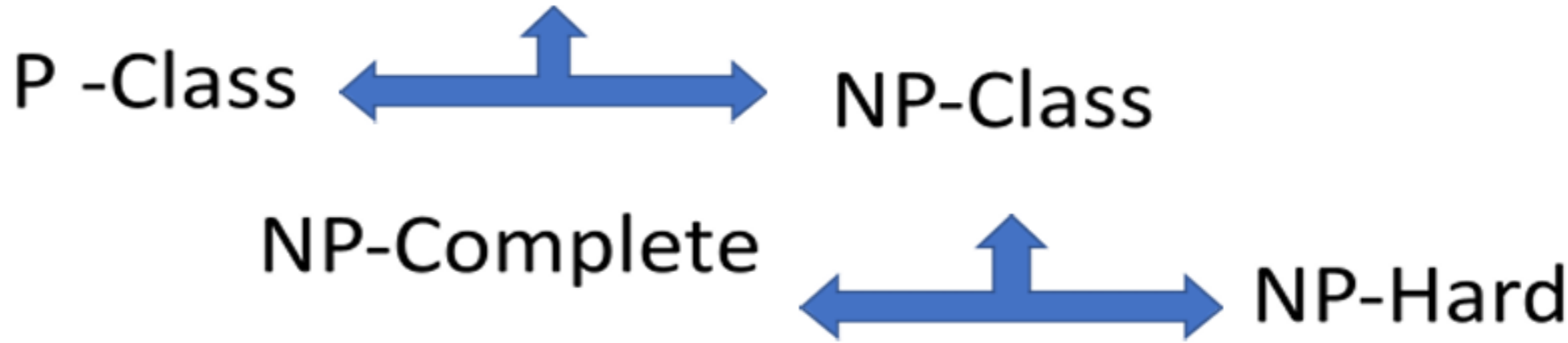
Some problems cannot be solved by an algorithm
Some problems can be solved algorithmically but not in polynomial time
Even though solution exists for a problem in polynomial time, there are lower bounds on the efficiency of algorithms i.e, efficiency of those algorithms cannot be improved further.

Polynomial time:

The amount of time it takes for an algorithm to solve a polymomial function, which is a mathematical expression that does not contain fractions or negative numbers (non-negative integers). The time is proportional to the input and very efficient.

Classification of computational theory:

Computational theory:



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.***

A ***nondeterministic algorithm is a two-stage procedure that*** takes as its input an instance *I of a decision problem and does the following.*

Nondeterministic ("guessing") stage: An arbitrary string *S is generated that* can be thought of as a candidate solution to the given instance *I*

Deterministic ("verification") stage: A deterministic algorithm takes both *I* and *S as its input and outputs, It produces yes if S represents a solution to instance I.*

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

Decision tree:

8.3 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. Now, let us see "What are decision trees?"

Definition: *A decision tree also called comparison 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*

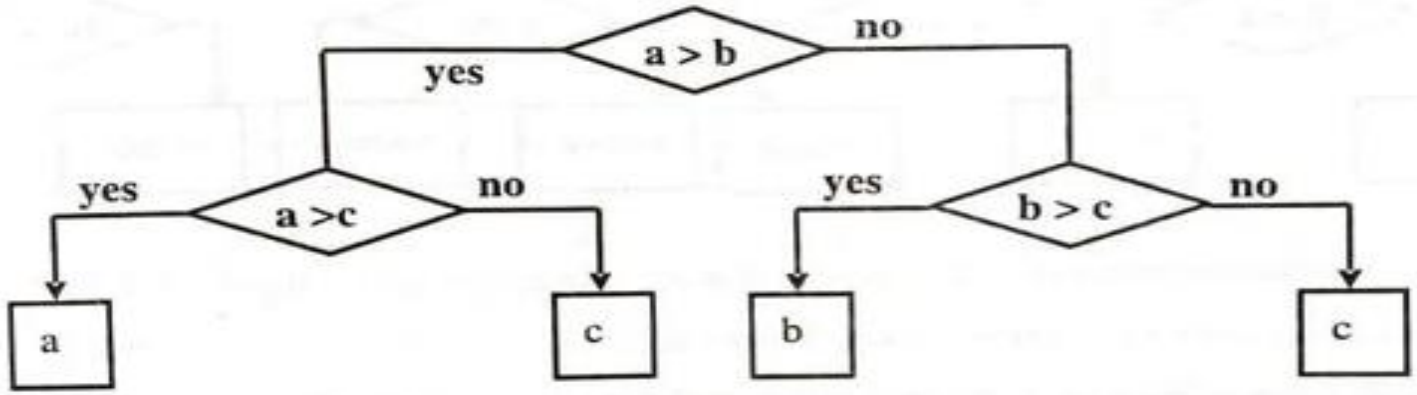
Decision Trees for Sorting

Most sorting algorithms are comparison based, i.e., they work by comparing elements in a list to be sorted. By studying properties of decision trees for such algorithms, we can derive important lower bounds on their time efficiencies.

We can interpret an outcome of a sorting algorithm as finding a permutation of the element indices of an input list that puts the list's elements in ascending order. Consider, as an example, a three-element list a, b, c of orderable items such as real numbers or strings. For the outcome $a < c < b$ obtained by sorting this list (see Figure 11.2), the permutation in question is 1, 3, 2. In general, the number of possible outcomes for sorting an arbitrary n-element list is equal to $n!$.

Example 8.1: Obtain the decision tree to find minimum of three numbers

The decision tree or comparison tree shows only the comparisons of two elements in each of the node. The nodes at the leaf level are the minimum numbers obtained. The decision tree to find minimum of three numbers is shown below:



If a binary tree of height h has maximum number of leaves then it has only the leaves on the last level. The number of nodes in each level are:

The number of nodes in level $0 = 1 = 2^0$
The number of nodes in level $1 = 2 = 2^1$
The number of nodes in level $2 = 4 = 2^2$
.....
.....
Let the number of nodes in last level $= l \leq 2^h$ i.e., $2^h \geq l$. Taking log on both sides we get:

$$h \log_2 2 \geq \log_2 l$$

$$\text{i.e., } h \geq \log_2 l$$

Observe that the lower bound depends on the heights of binary decision trees. Such a bound is called *information theoretic lower bound*.

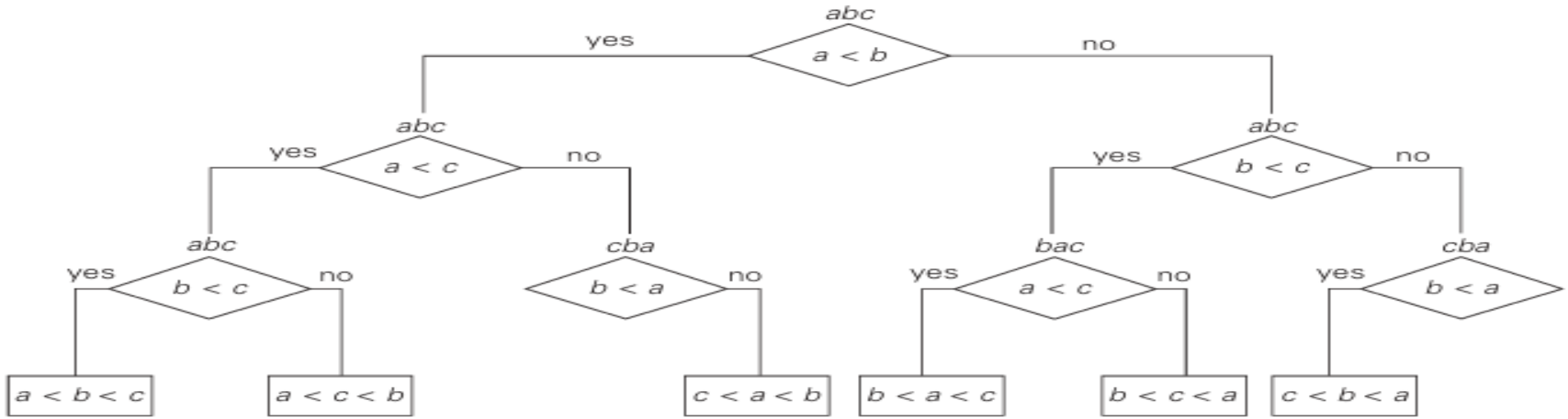


FIGURE 11.2 Decision tree for the tree-element selection sort. A triple above a node indicates the state of the array being sorted. Note two redundant comparisons $b < a$ with a single possible outcome because of the results of some previously made comparisons.

Inequality (11.1) implies that the height of a binary decision tree for any comparison-based sorting algorithm and hence the worst-case number of comparisons made by such an algorithm cannot be less than $\lceil \log_2 n! \rceil$:

$$C_{worst}(n) \geq \lceil \log_2 n! \rceil. \qquad \textbf{(11.2)}$$

Using Stirling's formula for $n!$, we get

$$\lceil \log_2 n! \rceil \approx \log_2 \sqrt{2\pi n}(n/e)^n = n \log_2 n - n \log_2 e + \frac{\log_2 n}{2} + \frac{\log_2 2\pi}{2} \approx n \log_2 n.$$

In other words, about $n \log_2 n$ comparisons are necessary in the worst case to sort an arbitrary n-element list by any comparison-based sorting algorithm. Note that mergesort makes about this number of comparisons in its worst case and hence is asymptotically optimal. This also implies that the asymptotic lower bound $n \log_2 n$ is tight and therefore cannot be substantially improved. We should point out, however, that the lower bound of $\lceil \log_2 n! \rceil$ can be improved for some values of n. For example, $\lceil \log_2 12! \rceil = 29$, but it has been proved that 30 comparisons are necessary (and sufficient) to sort an array of 12 elements in the worst case.

P,NP,NP Complete Problems:

8.8 Limitations of algorithm power

Definition: The set of decision problems that can be solved by deterministic algorithms in polynomial time are called polynomial-class problems or P-class problems or P problems. The P-class problems include only decision problems which are problems with yes/no answers. In other words, the problems that can be solved in polynomial time by deterministic algorithms are called P-problems.

For example, searching and sorting problems, finding GCD of two numbers, check whether the graph is connected or acyclic, finding a minimum spanning tree, finding the shortest path etc. are all P problems

Now, let us see "What is NP-Class?"

Definition: The problems that can be solved in polynomial time by non-deterministic algorithms are called non-deterministic P type problems or N-P problems. Here, N-P 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

Now, let us see "What are NP-Complete problems?"

Definition: The decision problem that belongs to NP-class is NP-complete problem. Here, every problem in NP is polynomially reducible to decision problem. We can show that a decision problem is NP-complete in two steps:

- We should show that the problem in question is in NP i.e., a randomly generated string can be checked in polynomial time to determine whether or not it represents a solution to the problem.
- We should show that every problem in NP is reducible to the problem in question in polynomial time.

For example, the following are NP-complete problems: Hamiltonian circuit, traveling salesman problem, assignment problem, knapsack problem etc.

It is not known whether P = NP or P is just a proper subset of NP. This question is the most important unresolved issue in theoretical computer science. A discovery of a polynomial-time algorithm for any one of known NP-complete problem imply that P = NP.

Now, let us see "What are NP-hard problems?

Definition: The optimization versions of difficult combinatorial problems fall in the class of NP-hard problems. The problems that are at least as hard as NP-complete problems are called NP-hard problems.

NP-Complete and NP-Hard problems:

Basic concepts

Depending on computing times, algorithms are clustered into two groups:

1. **P –class Problems:** Problems whose solution times are bounded by polynomials of small degree.

Example: Ordered search O(n log n), sorting O(n log n), polynomial evaluation O(n) , string editing O(mn) etc. [**P –class Problems** are the set of problems that can be solved by deterministic algorithms in polynomial time]

2. **NP-class Problems:** Problems whose best- known algorithm are non-polynomial.

Example: TSP $O(n^2 2^n)$, Knapsack $O(2^{n/2})$ [**NP –class Problems** are the set of decision problems that can be solved by nondeterministic algorithms]. These problems requires vast amount of time to execute even for moderate values of n. Such problems cannot be solved for large values of 'n'.

Problems which don't have polynomial time algorithm are computationally related and can be classified as follows:

NP-complete Problems: A problem that is NP-complete has the property that it can be solved in polynomial time if and only if all other NP-complete problems can also be solved in polynomial time.

NP-Hard Problems: If an NP-hard problem can be solved in polynomial time, then all NP-complete problems can be solved in polynomial time.

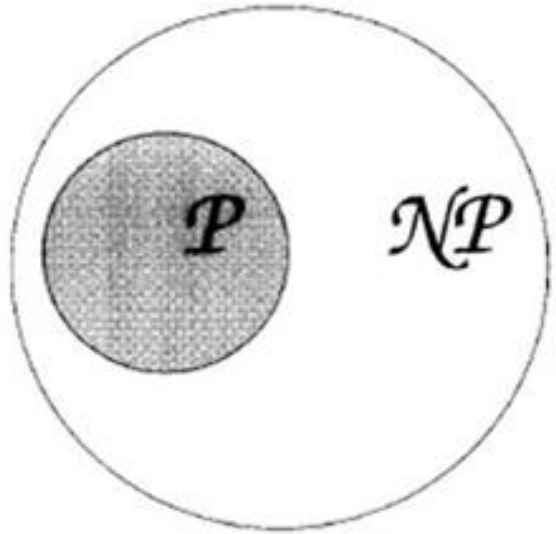
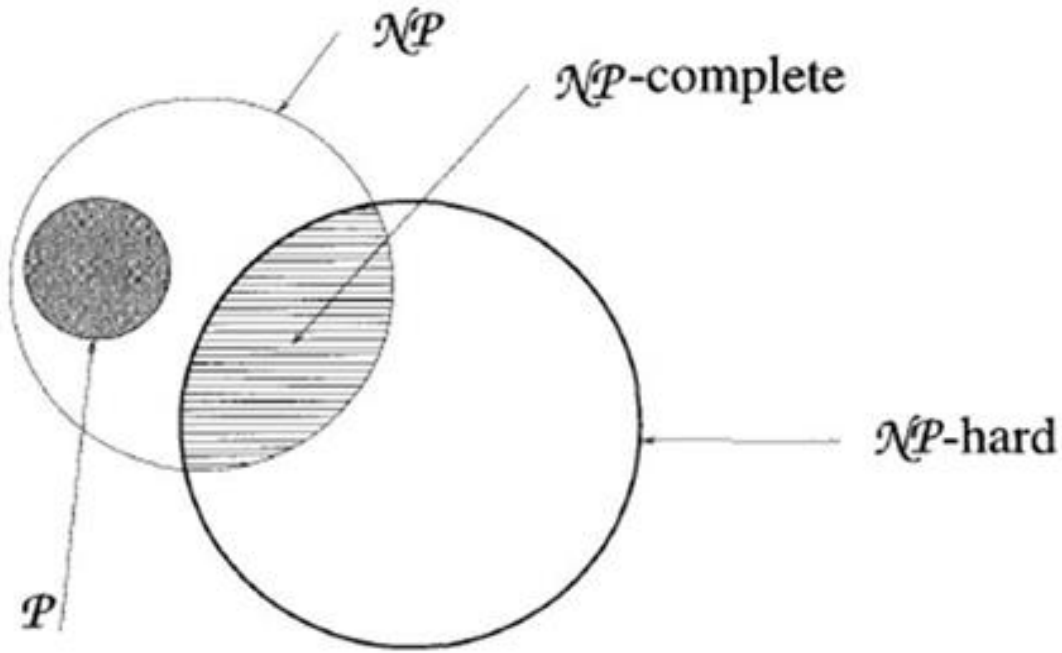


Figure: Commonly believed relationship between P and NP



re: Commonly believed relationship among P, NP, NP-complete, and NP-hard problems

Non-deterministic algorithms

Non deterministic algorithm contains operations whose outcomes are limited to specified sets of possibilities. The following three functions are used:

1. Choice (S) arbitrarily chooses one of the elements of set S.
2. Failure () signals an unsuccessful completion.
3. Success () signals a successful completion.

Problem 1: Searching x on A[1 : n], n ≥ 1 on success returns j if A[j] = x or returns 0 otherwise

```
j= Choice (1 , n) ;
if (A[j] == x) then {write (j) ; Success(); }
write(0), Failure( );
```

Problem 2: Sorting array A[1 : n] of positive integers in ascending order

```
Algorithm NSort (A, n)                  // sort n positive integers
{
        for i = 1 to n do B[i] = 0;     // Initialize B[]
        for i=1 to n do
        {
                j=Choice (1, n);
                if ( B[j] ≠ 0 ) then Failure ( );
                B[ j ] = A [ i ];
        }

        for i=1 to n-1 do                       // verify order
        if (B[i] > B [i+1] ) then Failure( );
        write (B[1: n]);
        Success ( );
}
```