10. LIMITATIONS OF ALGORITHM POWER

Algorithms are very powerful problem solving tools. The power of algorithms is limited because of the following reasons. -

- Some problems can not be solved by any 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 algorithm can not be improved further.

Majority of the algorithms are polynomial-time algorithms that is, for all i/ps of size n, the worst-case time complexity is $O(n^k)$ for some constant k.

Ex: all sorting & searching techniques, DFS and BFS traversals etc.

Some problems can be solved but not in time $O(n^k)$.

Ex: Traveling salesman problem $O(n-1!)$
Tower of hanoi $O(2^n)$
can not be solved in polynomial time.

Lower Bound Arguments.

The efficiency of an algorithm can be looked into two different ways.

1. By establishing the asympotic efficiency class i.e, obtaining & comparing upper bound with other class of algorithms. Upper bound is expressed using (O) Big-oh notation.

Scanned by CamScanner

Ex: comparing bubble sort $O(N^2)$ & Tower of Hanoi $O(2^n)$, bubble sort is efficient But comparision has to be avoided because these two problems solve different problems.

2. By establishing the efficiency class of solving the same problem with different methods.

Ex: Bubble sort is $O(N^2)$ & Quick sort is $N \log N$. So Quick sort is efficient.

Lower bound is defined as estimating the minimum amount of time needed to solve a given problem

Methods to obtain lower bounds are —

1. Trival Lower Bounds (I/p & o/p parameters)
2. Information Theoretic arguments (Decision tree & amt of info to produce)
3. Adversary arguments — measuring amt of work required to reduce a set of potential i/p to single i/p along most time consuming path.
4. problem reduction.

1. Trival Lower Bounds.

This is simplest method & is obtained by looking into input & output parameters ~~etc.~~ i.e,

- The no of items in the i/p that must be processed.
- The no of o/ps that have to be probluced.

Ex: while generating permutations, input size is n & the no of distinct o/ps are n!. ∴ There efficiency is $\Omega(n!)$

Consider the polynomial eqn of degree n.

$p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_0$ at a given

All the cofficients $a_n, a_{n-1}, a_{n-2}, \ldots, a_0$ have to be processed by any polynomial evaluation algorithm.

Limitations of trival lower bound.

- They are often not useful.
  Ex: In a traveling salesman problem, given n cities & $n(n-1)/2$ distances, the o/p 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 i/p must be processed.
  Ex: In binary search, the elements are arranged in ascending order. We know that there is no need to search for entire array using this technique.

2. Information - Theoretic arguments:

The lower bound is based on the amount of information it has to produce. This method is very useful for the lower bounds for problems like searching & sorting etc.

The lower bound using this method can be obtained using decision trees.

3. Adversary Arguments:

The lower bound is obtained by measuring the amount of work required to ~~pro~~ reduce a set of potential inputs to a single-input along the most time consuming path.

Ex: In game of guessing a number, the no of questions asked by adversary must be

4) Problem reduction:

In this, a given problem is transformed into another problem for which the solution exists in the form of known algorithm.

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 lower bound of P.

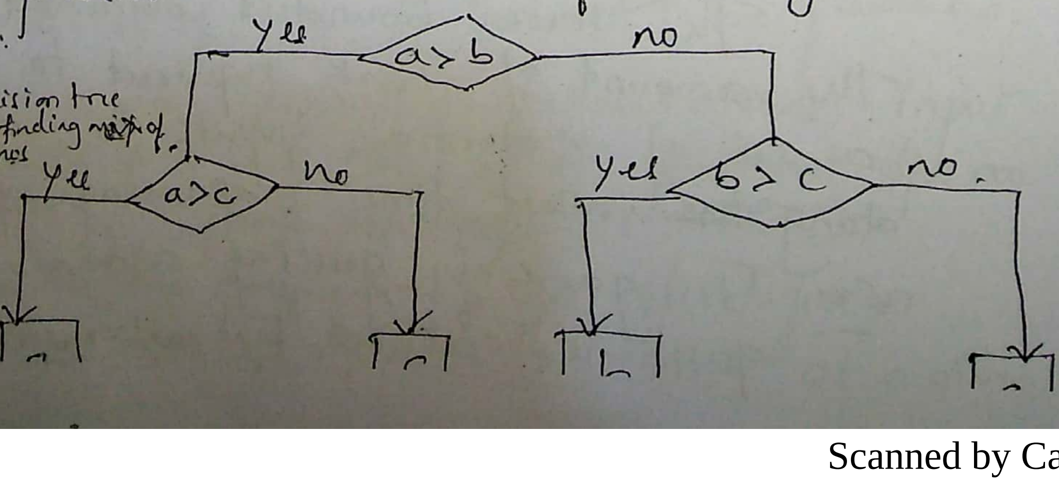
Decision Trees:

The algorithms such as sorting and searching work only by comparing the given elements. The performance of these algorithms which involve comparision can be obtained using decision trees.

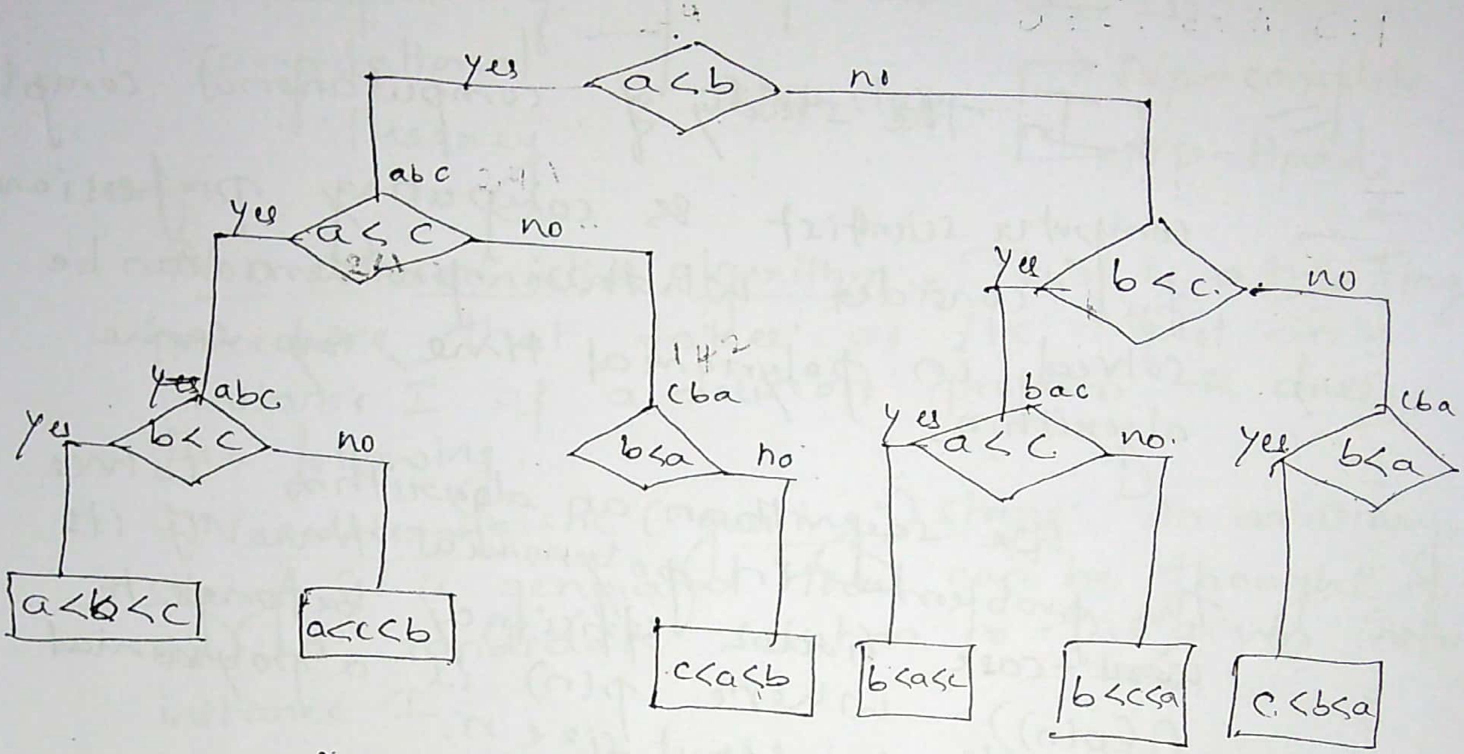
A decision tree is a binary tree that represents only the comparisions of given elements in array while sorting or searching. In decision tree, internal nodes represent the comparisions made between the items & leaf nodes represent the result of the algorithm.

Ex:

Decision tree for finding max of 3 nos



Decision tree for three-element selection sort (3)



A triple indicates the state of the array being sorted. The two redundant comparisions b<a with single possible outcome because of the results of some previously made comparisions.

A tree with a given number of leaves, which is dictated by the no of possible outcomes has to be tall enough to have that many leaves.

A binary tree with l leaves & height h,

$$h \geq \lceil \log_2 l \rceil.$$

A binary tree ~~of~~ height ~~of~~ h with largest no of its leaves in ~~a such tree~~ at the last level. Hence largest no of leaves in tree is 2^h.

Scanned by CamScanner

P, NP, and NP-complete Problems.

In the study of computational complexion, computer scientist & computing professionals first consider whether problem can be solved in polynomial time, by some algorithm.

We say than an algorithm solves the problem in polynomial time if its worst-case time efficiency belongs to $O(p(n))$ where $p(n)$ is a polynomial of problem's input size n.

Problems that can be solved in polynomial time are called tractable, problems that cannot be solved in polynomial time are called intractable.

P and NP problems.

The problems with yes/no answers are called as decision problems. P includes only decision problems.

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.

Ex: searching & sorting problems.
Hamiltonian circuit, Traveling salesman, Knapsack problem, graph coloring, etc.

Scanned by CamScanner

Classification of computational Theory
- p-class
- Np-class
  - Np-complete
  - Np-Hard.

Non deterministic algorithm: This is a two-stage procedure that takes as its input an instance I of a decision problem & 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 & outputs yes if S represents a solution to instance I, otherwise no.

Finally, a non-deterministic algorithm is said to be non-deterministic polynomial if the time efficiency of its verification stage is polynomial.

NP problems:

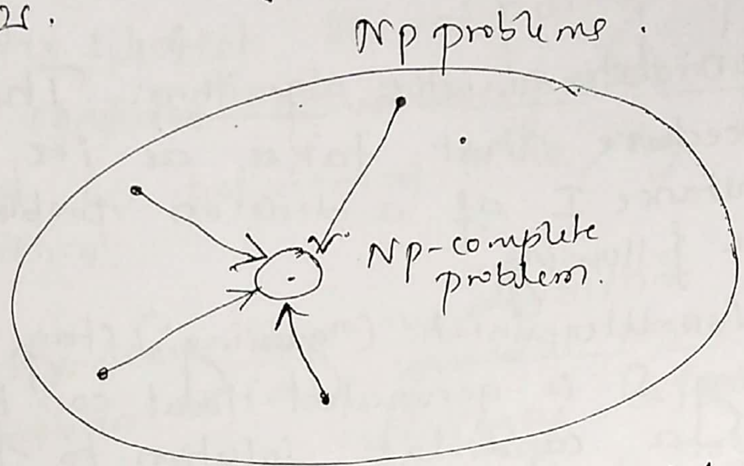
Class NP is the class of decision problems that can be solved by non-deterministic polynomial algorithms. This class of problems is called nondeterministic polynomial.

Most of decision problems are in Np. This class includes all the problems in P:

$$\boxed{P \subseteq NP}$$

Notion of an NP-complete problem.

Polynomial-time reductions of NP-problems to an NP-complete problem are shown by arrows.



A decision problem D_1 is said to be polynomially reducible to a decision problem D_2 if there exists a function t that transforms instances of D_1 to instances of D_2 such that

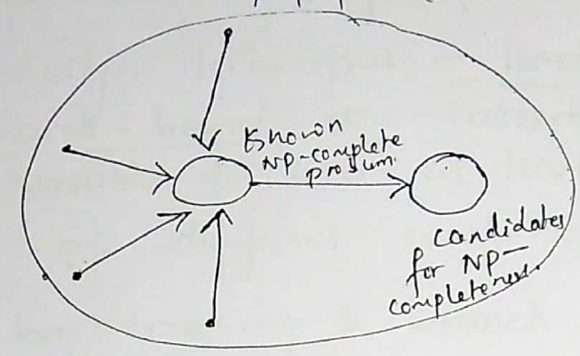
1. t maps all yes instances of D_1 to yes instances of D_2 & all no instances of D_1 to no instances of D_2
2. t is computable by a polynomial-time algorithm.

NP-complete Problems

A decision problem D is said to NP-complete if

1. It belongs to class NP;
2. Every problem in NP is polynomially reducible to D!

Proving NP-completeness by reduction



A decision problem is NP-complete can be shown in two steps.

1. One needs to show that the problem in question is in NP i.e., a randomly generated string can be checked ~~by~~ in polynomial time to determine whether or not it represents a solution to the problem.

2. Show that every problem in NP is reducible to the problem in question in polynomial time. Because of transitivity of polynomial reduction, this step can be done by showing that a known NP-complete problem can be transformed to the problem in question in polynomial time.

The halting problem is an example of an undecidable decision problem i.e., it cannot be solved by any algorithm.