6 LIMITATIONS OF ALGORITHMIC POWER

Introduction :-

So far we have studied many algorithms, to solve variety of problems. But it is not possible that every problem could be solved by some algorithm. In other words we can say that power of algorithms is limited to some extent. It is seen that power of algorithm is limited because of following reasons.

* There are some problems which can not be solved with algorithms.
* There are some problems that can be solved by algorithm but such solving is not within polynomial time.
* There are some problems that can be solved in polynomial time but there are lower bounds for efficiency of these algorithms.

Lower bound :- Basically obtaining lower bounds means "Estimating minimum amount of work needed to solve the problem".

When we try to obtain lower bounds of the algorithm we basically look for limits of efficiency of any algorithm that may solve the problem.

Eg :- The Eg in which lower bounds of corresponding algorithm can be obtained :

1.) NO of comparisions needed to sort an array of size n.

2.) NO of comparisions that are required to search a desired element from a sorted array.

3.) NO of additions needed to add two $n \times n$ matrices.

To obtain efficiency of particular algorithm there are 2 ways :-

(1) By asymptotic efficiency class for a given problem.

(2) Check where the class of given problem fits in the hierarchy of efficiency classes. In other words, whether the problem lies in linear, quadratic, logarithmic (or) exponential category of efficiency class.

Various Methods of Establishing

1) Trivial Lower Bounds
2) Information theoretic arguments
3) Adversary arguments
4) Problem Reduction.

Trivial Lower Bound :- This Technique is based on counting the no of items in the problems input that must be processed & the no of output items that need to be produced.

1 Eg:- Any algorithm for generating all permutations of 'n' distinct items must be in $\Omega(n!)$ because the size of the output is $n!$.

2 Eg:- The product of 2 $n \times n$ matrices algorithm. The trivial lower bound is $O(n^2)$ ∵ in this algorithm $2n^2$ elements (Input) are to be processed & n^2 elements get produced as any output.

Drawbacks:

① Trivial lower bounds are less useful. To know how consider the problem of TSP (Travelling salesperson problem). The trivial lower bound of this algorithm is $\Omega(n^2)$ ∵ there are $\frac{n(n-1)}{2}$ distances (Input) & $[n+1]$ cities as output. But this trivial lower bound is not useful because there is no similar algorithm in existence for comparision.

② It is necessary to determine which part of input has to be processed.

...ation - Theoretic arguments

This method is based on the amount of information produced by algorithm.

This approach has connection with information theory hence the name information theoretic argument. This method makes use of mechanism called decision trees.

Mrs. VANI K.A, B.E, M.Tech
Lecturer

(3) **Adversary arguments** :- It is a method of proving lower bound by playing a role of adversary in which algorithm has to work more for adjusting I/P (Input) consistently.

Eg:- Game of guessing no between 1 & n with yes/no questions.

adversary:- when adversary gives the answer one can get a larger set of no from which a secret no has to be found out.

(4) **Problem Reduction** :- A method in which a difficult & unsolvable problem 'A' is to be reduced to another solvable problem 'B' with known algorithm. The same principle is used for obtaining lower bounds. i.e. If problem 'A' is at least as hard as problem 'B' then a lower bound for 'B' is also lower bound for A.

Hence we have to find problem 'B' with a known lower bound which is further used for finding lower bound of problem A. This technique is recognised as reduction.

Eg:-

Let 'A' is an algorithm for finding Minimum spanning tree (MST) & 'B' is an algorithm of element uniqueness problem.

Consider a set $X = \{x_1, x_2, \ldots x_n\}$ from which unique elements can be found out. Thus set X is an instance of uniqueness problem.

Now we can create an instance of MST in the cartesian plane!

$Y = \{(0, x_1), (0, x_2), \ldots (0, x_n)\}$ Then using uniqueness of element x_i we can solve the problem of MST.

once, if we obtain lower bound of element uniqueness prob O(nlogn) that means the lower bound for MST is Ω(n log n) Trees

Thus in reduction technique arbitrary instance of problem 'B' transformed to instance of A. This ultimately results that lower bound of 'B' will be lower bound of 'A' too.

Scanned by CamScanner

Decision Trees

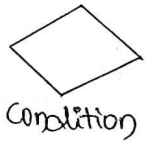
All the sorting & searching algorithms are based on comparision method. A model is prepared to study the performance of such algorithms which is called Decision Tree.

In decision tree

* Internal nodes represent the comparisions made between input items.
* Leaf nodes represent the result of the algorithm.

Eg:- Draw a decision tree for finding largest no among 3 variables X, Y, & Z.

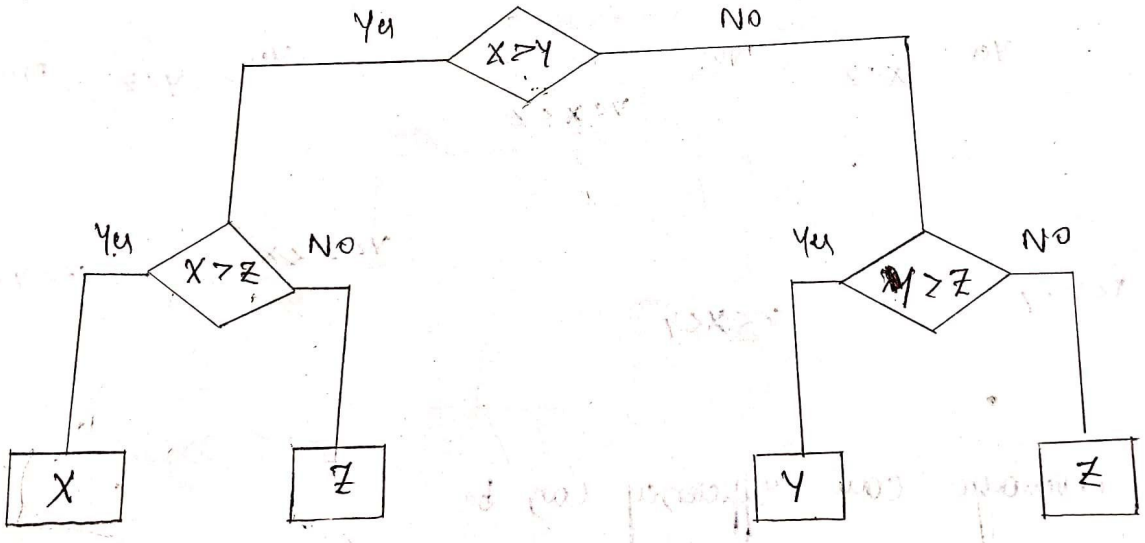
The simple graphical notations used in decision trees are.


Condition

Transition

result / outcome.

The decision tree for the given problem is.



For the above decision tree the following observations can be made.

① Let 'h' be the height of a tree &
'n' be the total no of leaf nodes.

Then

$h \geq \lceil \log n! \rceil$

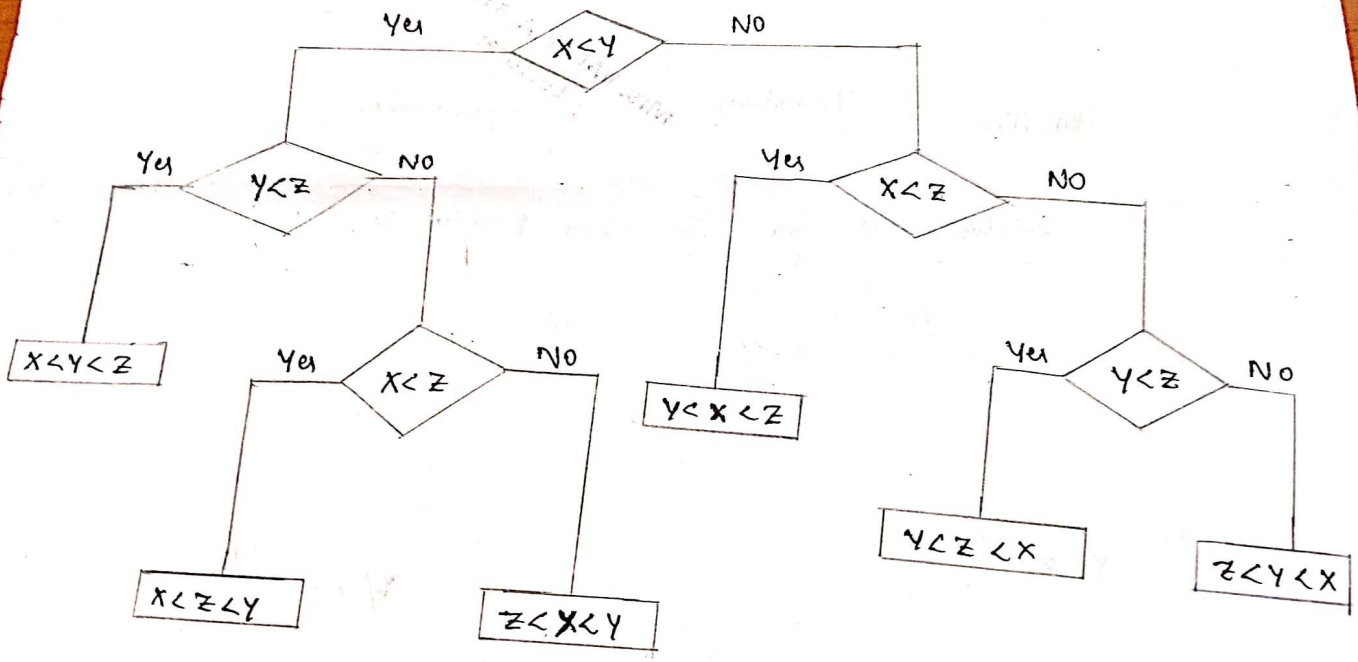
Note :- $\log n!$ is rounded up value of ceil function.

In other words we can also state that

$2^h \geq n!$

Due to the inequality indicated by above equation the lower bound on height of the binary decision tree can be obtained. And we get such a lower bound using the information about the output of algorithm. Hence this bound is called the Information theoretic lower bound.

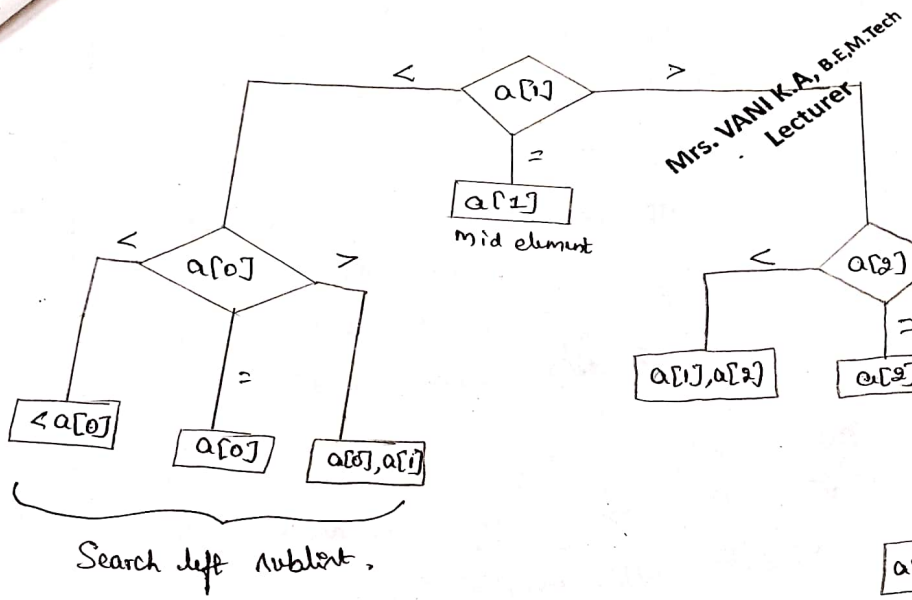
Eg.) Draw the decision tree for insertion sort of 3 elements



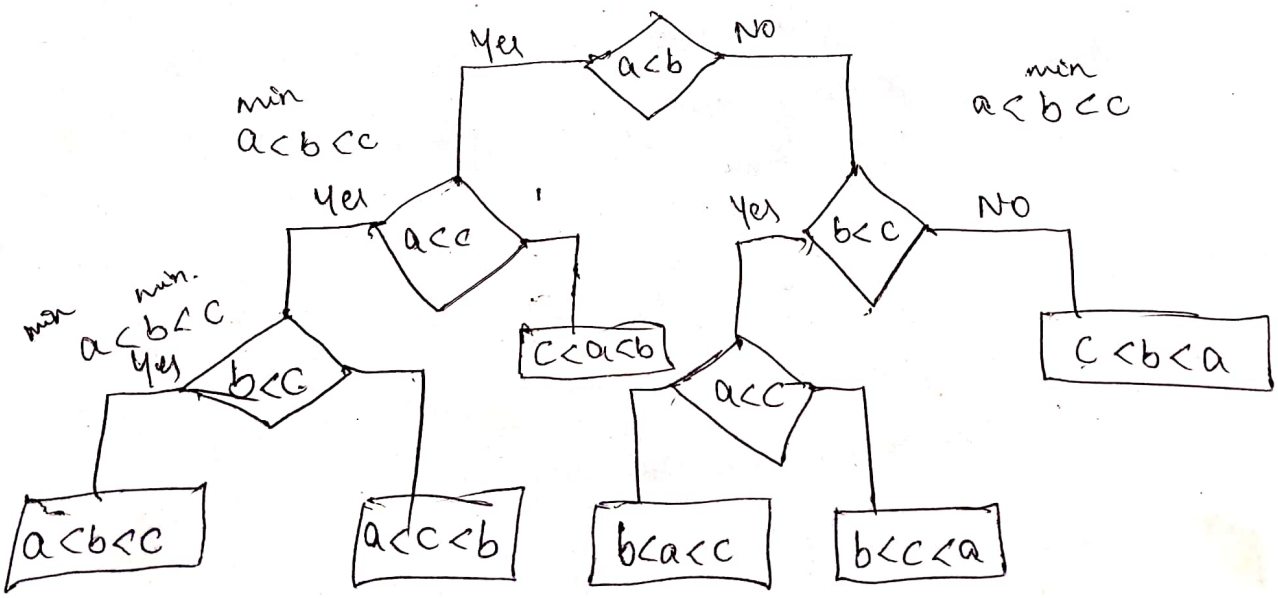
The average case efficiency can be

$$C(n) \geq \log_2 n!$$

Eg: Draw a Ternary decision tree for 'n' no's of elements



Selection sort



9 NP-Complete Problems

Mrs. VANI K.A, B.E.M.Tech
Lecturer

There are 2 groups in which a problem can be classified.

1st group:- Consists of the problems that can be solved in polynomial time. Eg:- Searching of an element from the list $O(\log n)$, Sorting of elements $O(n \log n)$.

2nd group:- Consists of the problems that can be solved in non-deterministic polynomial time.

Eg:- Knapsack problem $O(2^{n/2})$ &
TSP → Travelling salesperson problem. $O(n^2 2^n)$

* Any problem for which answer is either yes or no is called decision problem. The algorithm for decision problem is called decision algorithm.
* Any problem that involves the identification of optimal cost is called optimization problem. The algorithm for optimization problem is called optimization algorithm.

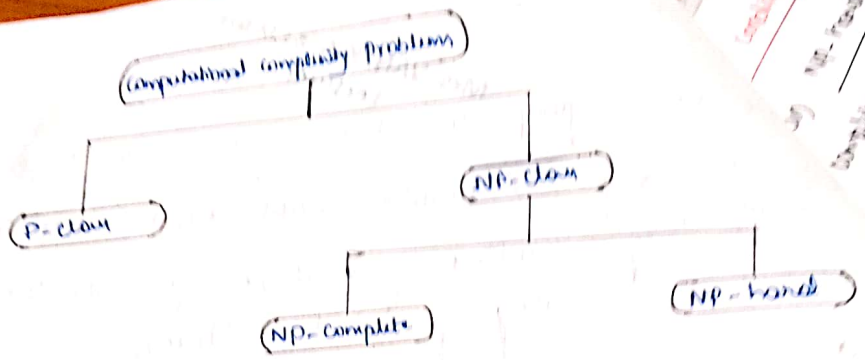
Defination of P:- problems that can be solved in polynomial time ("P" stands for polynomial)

Eg:- Searching of key element, Sorting of elements, All pair shortest path.

Defination of NP:- It stands for "non-deterministic polynomial time". The problems that can be solved in non-deterministic polynomial time. ("NP" stands for Non-deterministic polynomial time).

Eg:- Travelling salesperson problem (TSP), Graph coloring problem, Knapsack problem, Hamiltonian circuit problem.

NP class problems can be further classified into NP-Complete & NP hard problems.



* A problem 'D' is called NP_complete if

(i) It belongs to class NP.

(ii) Every problem in NP is polynomially reducible to 'D'.

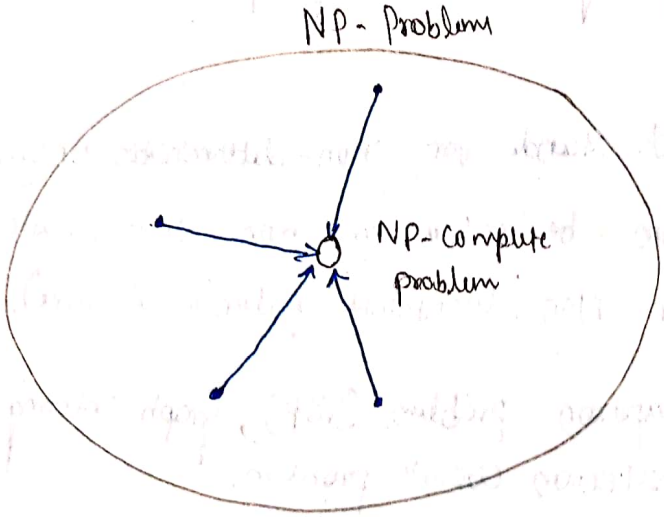
Let us see what is polynomially reducible to 'D' ?

Let D_1 is said to be polynomially reducible to 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.

By the above said definition of a problem D_1 is polynomially reducible to some problem D_2 that can be solved in polynomial time, then problem D_1 can also be solved in polynomial time.



conditions:-

- If any NP-hard problem can be solved in polynomial time then all NP-complete problems can also be solved in polynomial time.
- All NP-complete problems are NP-hard but all NP-hard problems may not be NP-complete.
- The NP-class problems are the decision problems that can be solved by non-deterministic polynomial algorithms.

Non deterministic algorithm

Mrs. VANI K.A, B.E.M.Tech
Lecturer

- The algorithm in which every operation is uniquely defined is called deterministic algorithm.
- The algorithm in which every operation may not have unique results, rather there can be specified set of possibilities for every operation. Such an algorithm is called Non-deterministic algorithm.

Non deterministic means that no particular rule is followed to make the guess.

The algorithm has 2 stages;-

(i) Non deterministic (Guessing) stage:- Generate an arbitrary string that can be thought of as a candidate solution.

(ii) Deterministic (Verification) stage:- In this stage it takes an Input the candidate solution & instance to the problem & returns Yes if the candidate solution represents the actual solution.

eg:- Algorithm Non-Determin( )

// A[1:n] is a set of elements.

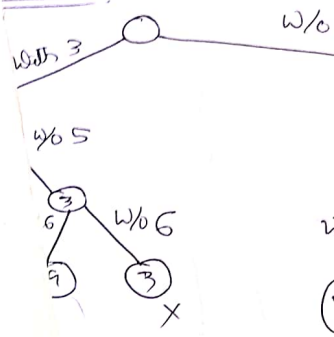
{

The following for-loop in the guessing stage

for i:=1 to n do

A[i]:= choose(i);

Subsets: S = {3, 5, 6,



// Now is the verification (deterministic) stage.
if (A[i] = x) then
{
write (i);
Success();
}
{
write (0);
fail();
}

In the above algorithm there are 3 functions used :-

(i) choose → arbitrarily choosed one of the element from the given i/p set.

(ii) fail → Indicates the unsuccessful completion.

(iii) Success → Indicates successful completion.

Eg:- write a non deterministic algorithm for sorting elements in non decreasing order.

Algorithm Non-Dsort (A, n)
// A[1:n] is an array that stores 'n' elements which are positive integers B[1:n] be an auxiliary array in which elements are put at proper positions //
{
// guessing stage
for i := 1 to n do
B[i] := 0; // Initialize B to 0
for i := 1 to n do
{
j := choose (1, n) // select any element from the i/p set
if (B[j] != 0) then
fail();
B[j] := A[i];
}
// verification stage
for i := 1 to n-1 do
if (B[i] > B[i+1]) then
fail(); // not sorted elements

NP-Hard & NP-complete classes

As we know 'P' denotes the class of all deterministic polynomial problems & NP denotes the class of all non deterministic polynomial problems. Hence the class

P ⊆ NP

* Problems which are tractable.
Problems

fail (B[1:n]); // Print the sorted list
success();

```
write (B[i:n]); // print the sorted list of elements
success();
}
```

NP-Hard & NP-Complete Classes

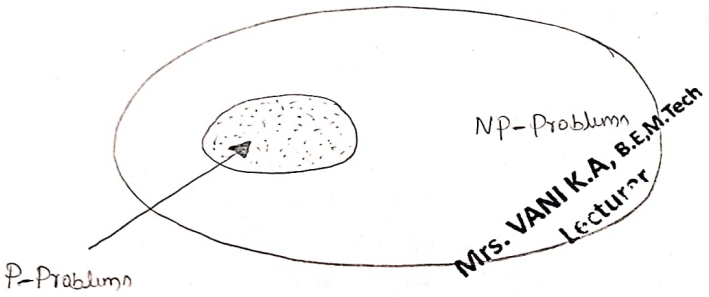
* As we know 'P' denotes the class of all deterministic polynomial language problems & 'NP' denotes the class of all non-deterministic polynomial language problems. Hence

$$P \subseteq NP$$

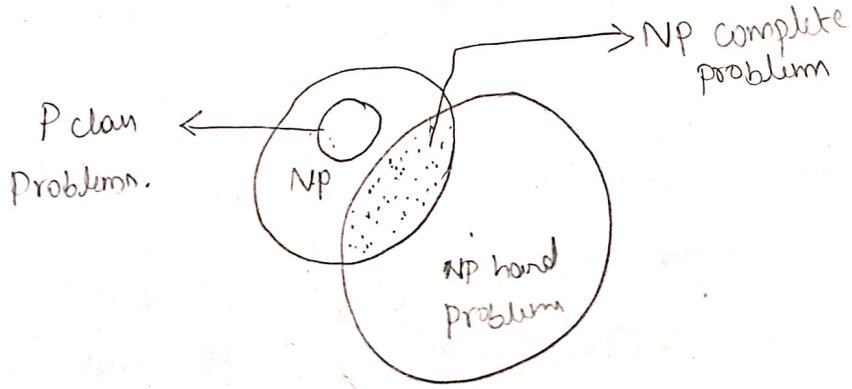
* Problems which are known to lie in 'P' are often called as tractable.

Problems which lie outside of 'P' are often termed as intractable.

The relationship between 'P' & 'NP' is depicted by following figure.



* A NP problem such that if it in in 'P' then NP=P if a problem has this property then it is called "NP-hard". Thus the class of NP-complete problem is the intersection of the NP & NP-hard classes.



Relationship between P, NP, NP-Complete & NP-hard problems.

* Normally the decision problems are NP-complete but problems are NP-hard.

However if problem 'A' is a decision problem & 'B' is a problem then it is possible that A reduces to B i.e. instance 'A ∝ B'

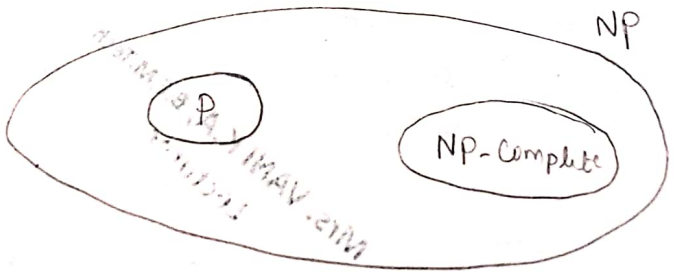
Eg:- Knapsack decision problem can be reduced into knapsack optimization problem.

* There are some NP-hard problems that are not NP-complete.

Eg:- Halting problem.

Halting problem states that :- "It is possible to determine whether an algorithm will ever halt (or) enter in a loop on certain input.

* Two problems 'P' & 'Q' are said to be polynomially equivalent if & only if P ∝ Q & Q ∝ P.



Numerical Algorithm

Numerical analysis is usually ducribed as a branch of computer science with algorithms for solving mathematical problems.

There are different kinds of problems like the one with equations of systems of equations, evaluating integrals of $\sin x$ & $\ln x$.

The other kind of problems with structures