Module - 4 & 5

Theory questions

Q.1. Variants of TM

Q.2. Model of Linear bounded Automata

Q.3. Recursive & Recursively enumerable language.

Q.4. Algorithm

Q.5. Decidable & Undecidable language

Q.6. Halting Problem of TM

Q.7. Post Correspondence Problem

Q.8. Time complexity & order of growth.

Q.9. The classes of P & NP

Q.10. Quantum computation

Q.11. Church - Turing Thesis.

7.1 Variants of Turing Machines (TM)

VTU : June-16, Dec.-16, Marks 5

7.1.1 Multi-tape Turing Machine

The multitape Turing machine is a type of turing machine in which there are more than one input tapes. Each tape is divided into cells and each cell can hold any symbol of finite tape alphabet. The multitape turing machine is as shown in the Fig. 7.1.1

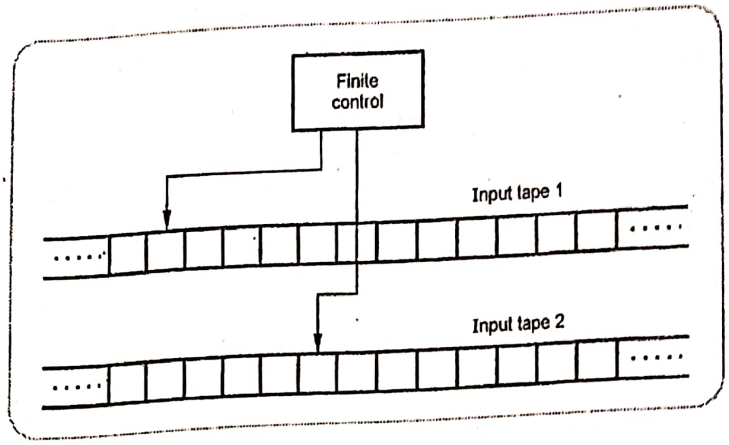


Fig. 7.1.1 A multitape Turing machine

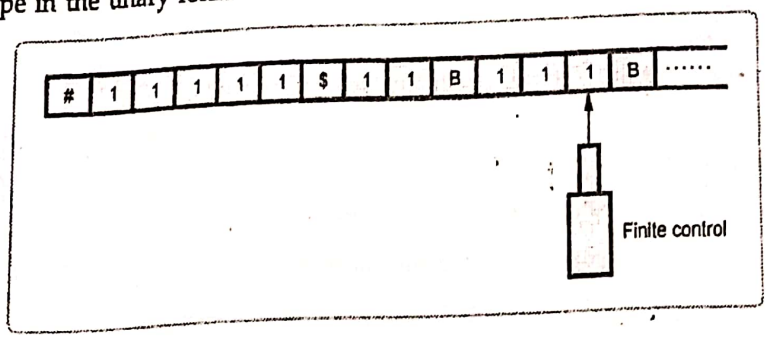
This TM is more powerful than the basic turing machine. Because finite control reads more than one input tape and more symbols can be scanned at a time.

Example 7.1.1 *Show that if L is accepted by a multitape turing machine, it is accepted by single tape Turing machine also.*

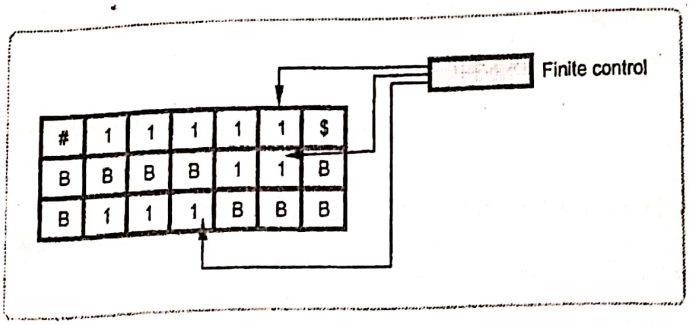
Solution :

The input string which is placed on a single tape can be divided into multiple tapes. The finite control will then read the inputs from each tape.

Consider the unary number equivalent to 5 and 2 we want to perform 5 – 2 and the result will be placed on the same single input tape in the unary form.



This can be shown by multiple tape.



Thus the language accepted by multiple tape is also acceptable by single tape.

7.1.2 Non Deterministic Turing Machine

- The non deterministic turing machine is a kind of turing machine in which the set of rules denote more than one specific action on reading particular input in current specific state.
- This kind of TM just similar to NFA.

For example -

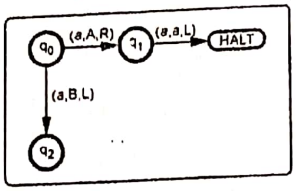


Fig. 7.1.2

- Note that any language accepted by non deterministic turing machine can also be accepted by deterministic turing machine.
- The power of non deterministic turing machine is just similar to the power of deterministic turing machine.

University Question

1. *Write Short note on - Multitape Turing Machine* [illegible]

7.2 The Model of Linear Bounded Automata

- The Linear Bounded Automata (LBA) is a model which was originally developed as a model for actual computers rather than model for computational process.
- A linear bounded automaton is a restricted form of a non deterministic Turing machine.

Concept : A linear bounded automaton is a multitrack Turing machine which has only one tape and this tape is exactly of same length as that of input.

- The Linear Bounded Automaton (LBA) accepts the string in the similar manner as that of Turing machine does.
- For LBA halting means accepting. In LBA computation is restricted to an area bounded by length of the input. This is very much similar to programming environment where size of variable is bounded by its data type.
- The LBA is powerful than Non deterministic Push Down Automata(NPDA) but less powerful than Turing machine.

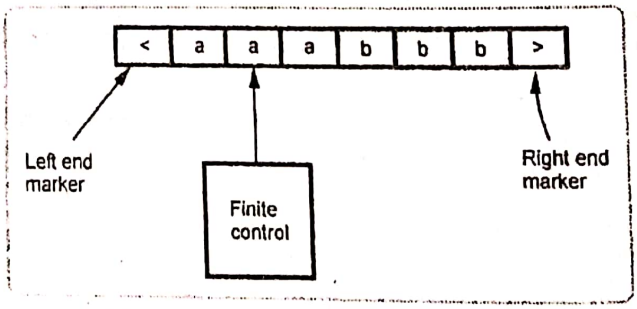


Fig. 7.2.1 Linear bound automaton

- The input is placed on the input tape with beginning and end markers.

In the given Fig. 7.2.1 the input is bounded by < and >.

A linear bounded automata can be **formally defined as :**

It is 7 - tuple non deterministic Turing machine with

$$M = \left(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}\right) \text{ having}$$

1. Two extra symbols of left end marker and right end marker which are not elements of Γ.
2. The input lies between these end markers.
3. The TM cannot replace < or > with anything else nor move the tape head left of < or right of >.

7.3 Decidability

- If a language is recursive then it is called decidable language and if the language is not recursive then such a language is called undecidable language.
- The class of decidable problems are also called as solvable problems and the class of undecidable problems is called unsolvable problems.

7.3.1 Recursive and Recursively Enumerable Languages

- A language is said to be recursive if there exists a turing machine that accepts every string of the language and every string is rejected if it is not belonging to that language.

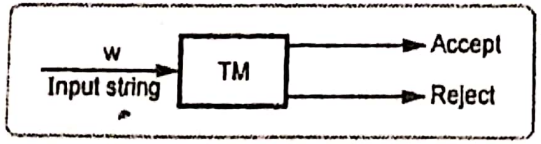


Fig. 7.3.1 Recursive languages

- A language is recursively enumerable if there exists a turing machine that accepts every string belonging to that language. And if the string does not belong to that language then it can cause a turing machine to enter in an infinite loop.

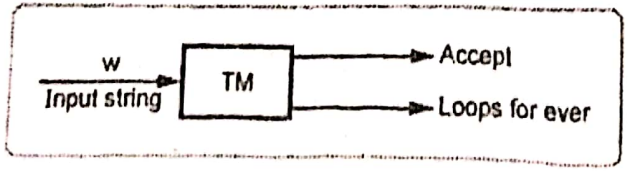


Fig. 7.3.2 Recursively enumerable languages

7.3.2 Definition of an Algorithm

Definition of Algorithm : The algorithm is defined as a collection of unambiguous instructions occurring in some specific sequence and such an algorithm should produce output for given set of input in finite amount of time.

This definition of algorithm is represented in Fig. 7.3.3

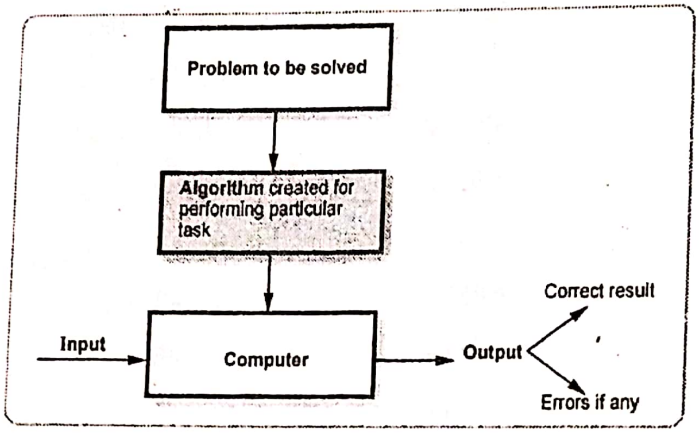


Fig. 7.3.3 Notion of Algorithm

- After understanding the problem statement we have to create an algorithm carefully for the given problem.
- The algorithm is then converted into some programming language and then given to some computing device (computer).
- The computer then executes this algorithm which is actually submitted in the form of source program.
- During the process of execution it requires certain set of input. With the help of algorithm (in the form of program) and input set, the result is produced as an output.
- If the given input is invalid then it should raise appropriate error message ; otherwise correct result will be produced as an output.

Properties of Algorithm

1. **Non-ambiguity :** Each step in an algorithm should be non-ambiguous. That means each instruction should be clear and precise. The instruction in an algorithm should not denote any conflicting meaning.
2. **Range of input :** The range of input should be specified. This is because normally the algorithm is input driven and if the range of the input is not been specified then algorithm can go in an infinite state.
3. **Multiplicity :** The same algorithm can be represented in several different ways. That means we can write in simple english the sequence of instructions or we can write it in the form of pseudo code. Similarly for solving the same problem we can write several different algorithms. For instance : For searching a number from the given list we can use sequential search or a binary search method. Here *"searching"* is a task and use of either a *"sequential search method"* or *"binary search method"* is an algorithm.
4. **Speed :** The algorithms are written using some specific ideas (which is popularly known as logic of algorithm). But such algorithms should be efficient and should produce the output with fast speed.
5. **Finiteness :** The algorithm should be finite. That means after performing required operations it should terminate.

7.3.4 Decidable Languages and Undecidable Languages

- The recursively enumerable (RE) languages are categorized into two classes -
  1. The class of languages that has Turing machine. This Turing machine decides whether the input string belongs to that language or not. Such a Turing machine always halts, whether or not it reaches to accept state.
  2. The second class of languages consists of those RE languages that are not accepted by any Turing machine with the guarantee of halting.
- A language is L actually denoted by L(M) called **recursive** if it is accepted by some Turing machine such that
  1. If string w is in L, turing machine M accepts it and then halts.
  2. If w is not in L, then M eventually halts although it never enters in accepting states.
- We can also call recursive languages as the **definite languages** or the languages which can be represented by some **algorithm** and such an algorithm helps in construction of Turing machine for that language.

Decidable and Undecidable Languages -

- If a language is **recursive** then it is called **decidable languages** and if the language is **not recursive** then such a language is called **undecidable** language.
- Hence broadly there are three categories of the languages.
  1. Recursive language for which the algorithm exists.
  2. Recursively enumerable language for which it is not sure that on which input the TM will ever halt. Such languages are not recursive.
  3. The non-recursively enumerable languages for which there is no Turing machine at all.
- Fig. 7.3.4 shows the relationship between these languages.

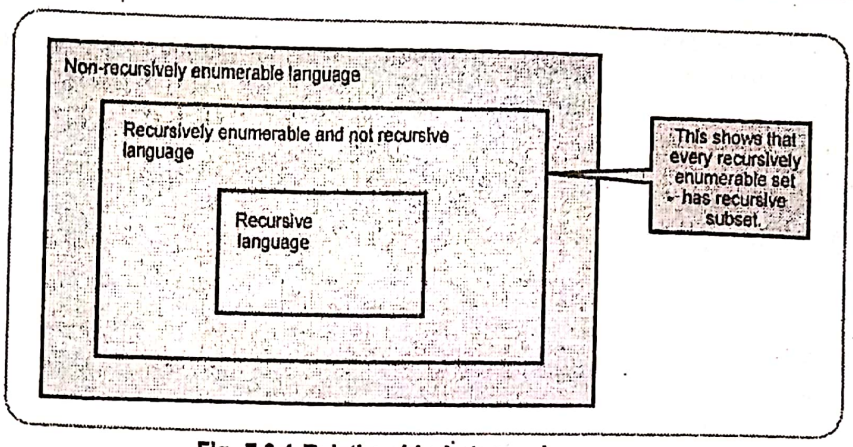


Fig. 7.3.4 Relationship between languages

- The important fact about recursive and recursively enumerable languages can be seen with the help of following theorem.

Theorem 1 : If L is recursive language then L' is also a recursive language.

Proof :

Let there will be some L that can be accepted by turing machine M. Hence we can denote language L by L(M). On acceptance of L(M) the machine M always halts. Now, we construct a TM M' such that L' = (M'). for construction of M' following steps are followed -

1. The accepting steps of M are made non-accepting states of M' and there is no transition from M'. That means we have created the states such that M' will halt without accepting.
2. Now create a new accepting state for M' say r and there is no transition from r.
3. In machine M, for each of the transition with combination of nonaccepting state and input tape symbol, make the same transition having the combination of accepting state and input tape symbol for machine M'.

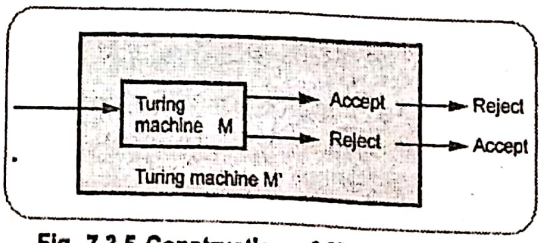


Fig. 7.3.5 Construction of M ' accepting L'

Since M is guaranteed to halt M' is also guaranteed to halt. In fact, M' accepts exactly those strings that M does not accept. Thus we can say that M' accepts L'.

Theorem 2 : If a language L and its complement L' both are RE then L is a recursive language.

Proof :

Consider a turing machine M made up of two turing machines M1 and M2. The machine M2 is complement of machine M1. We can also denote that L(M) = L(M1) and L(M2). Both M1 and M2 are simulated in parallel by machine M. Machine M is a two tape TM, which can be made one tape TM for the ease of simulation. This One tape then will consist of tape of machine M1 and machine M2. The states of M consists of all the states of machine M1 and all the states of machine M2. The machine M made up of M1 and M2 is as shown in Fig. 7.3.6.

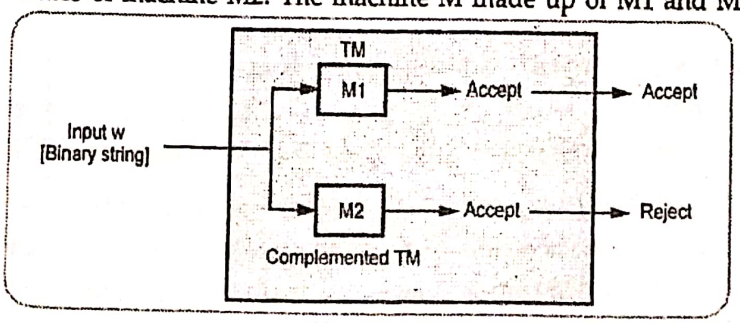


Fig. 7.3.6

7.4 Halting Problem of TM

VTU : Dec.-16, M

To state halting problem we will consider the given configuration of a turing machine. The output of TM can

i) **Halt :** The machine starting at this configuration will halt after a finite number of states.

ii) **No Halt :** The machine starting at this configuration never reaches a halt state, no matter how long it ru

- Now the question arises based on these two observation : Given any functional matrix, input data tape initial configuration, then is it possible to determine whether the process will ever halt? This is called **ha problem.**
- That means we are asking for a procedure which enable us to solve the halting problem for every (machine, tape). The answer is "no".
- That is the halting problem is unsolvable.
- Now we will prove why it is unsolvable.
- Let, there exists a TM M_1 which decides whether or not any computation by a TM T will ever halt wh description d_T of T and tape t of T is given [That means the input to machine M_1 will be (machine, pair].
- Then for every input (t, d_T) to M_1 if T halt for input t, M_1 also halts which is called **accept halt.**
- Similarly if T does not halt for input t then the M_1 will halt which is called **reject halt.** This is shown in Fig. 7.4.1

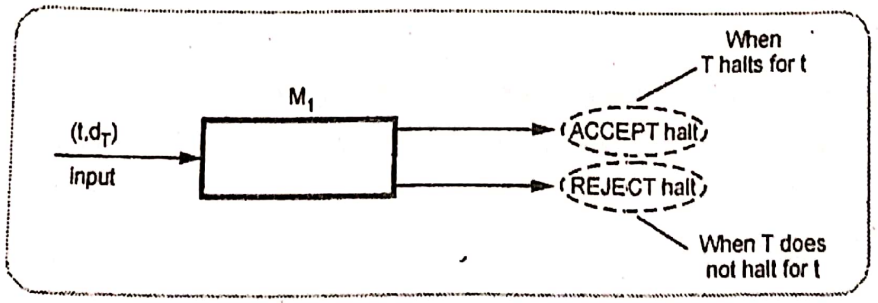


Fig. 7.4.1

- Now we will consider another Turing Machine M_2 which takes an input d_T.
- It first copies d_T and duplicates d_T on its tape and then this duplicated tape information is given as inp machine M_1. But machine M_1 is a modified machine with the modification that whenever M_1 is suppos reach an accept halt, M_2 loops forever.

- Hence behavior of M_2 is as given. It loops if T halts for input $t = d_T$ and halts if T does not halt for $T = d_T$
- The T is any orbitary Turing machine.

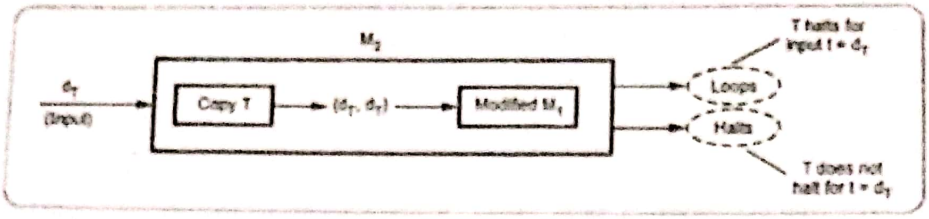


Fig. 7.4.2

As M_2 itself is one turing machine we will take $M_2 = T$. That means we will replace T by M_2 from above given machine.

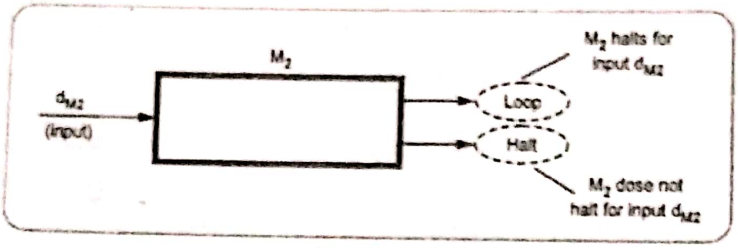


Fig. 7.4.3

Thus machine M_2 halts for input d_{M2} if M_2 does not halt for input d_{M2}. This is a contradiction. That means a machine M_1 which can tell whether any other TM will halt on particular input does not exist. Hence halting problem is unsolvable.

University Question

1. *Write Short note on - Halting Problem in TM* **VTU : Dec.-16, 5 Marks**

7.5 Post Correspondence Problem

VTU : June-16, Dec.-16, Marks 5

In this section, we will discuss the undecidability of strings and not of Turing machines. The undecidability of strings is determined with the help of Post's Correspondence Problem (PCP). Let us define the PCP.

The post's correspondence problem consists of two lists of strings that are of equal length over the input Σ. The two lists are $A = w_1, w_2, w_3, \ldots, w_n$ and $B = x_1, x_2, x_3 \ldots x_n$ then there exists a non empty set of integers $i_1, i_2, i_3 \ldots i_n$ such that,

$w_1, w_2, w_3 \ldots w_n = x_1, x_2, x_3, \ldots x_n^n$

To solve the post correspondence problem we try all the combinations of $i_1, i_2, i_3 \ldots, i_n$ to find the $w_i = x_i$ then we say that PCP has a solution.

Example 7.5.1 *Consider the correspondence system as given below - A = (1; 0; 010; 11) and B = (10; 10; 01; 1). The input set is Σ = (0, 1). Find the solution.*

Solution : A solution is 1; 2; 1; 3; 3; 4. That means $w_1w_2w_1w_3w_3w_4 = x_1x_2x_1x_3x_3x_4$.

The constructed string from both lists is 10101001011.

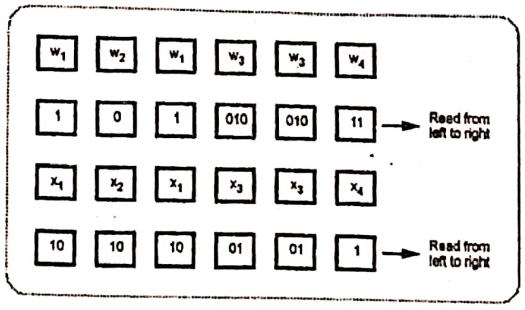


Fig. 7.5.1

Example 7.5.2 *Obtain the solution for the following correspondence system.*
A = {ba, ab, a, baa, b}, B = {bab, baa, ba, a, aba}. The input set {a, b}.

Solution : To obtain the corresponding system the one sequence can be chosen. Hence we get

$w_1w_5w_2w_3w_4w_4w_3w_4 = x_1x_5x_2x_3x_4x_4w_3w_4$. This solution gives a unique string bababababaaabaaa = bababababaaabaa. Hence solution is 15234434.

Example 7.5.3 *Obtain the solution for the following system of posts correspondence problem.*
A = {100, 0, 1}, B = {1, 100, 00}

Solution : The solution is 1 3 1 1 3 2 2. The string is

A1A3A1A1A1A3A2A2 = 100 + 1 + 100 + 100 + 1 + 0 + 0
= 1001100100100

B1B3B1B1B3B2B2 = 1 + 00 + 1 + 1 + 00 + 100 + 100
= 1001100100100

Example 7.5.4 *Obtain the solution for the following system of posts correspondence problem*
A = {ba, abb, bab} B = {bab, bb, abb}.

Solution : Now to consider 1, 3, 2 the string bababab from set A and bababbbb from set B thus the two strings obtained are not equal. As we can try various combinations from both the sets to find the unique sequence but we could not get such a sequence. Hence there is no solution for this system.

Example 7.5.5 *Does PCP with two lists $x = (b, bab^3, ba)$ and $y = (b^3, ba, a)$ have a solution ?*

Solution : Now we have to find out such a sequence that strings formed by x and y are identical. Such a sequence is 2, 1, 1, 3. Hence from x and y list

2	1	1	3		2	1	1	3
bab^3	b	b	ba	=	ba	b^3	b^3	a

which forms bab^3b^3a. Thus PCP has a solution.

University Question

1. *Write Short note on - Post Correspondence Problem* VTU : June-16, Dec.-16, Marks 5

7.6 Complexity

- The efficiency of an algorithm can be decided by measuring the performance of an algorithm. We can measure the performance of an algorithm by computing two factors.
  1. Amount of time required by an algorithm to execute.
  2. Amount of storage required by an algorithm.

Hence we define two terms - Time Complexity and Space Complexity

- Time complexity of an algorithm means the amount of time taken by an algorithm to run. By computing time complexity we come to know whether the algorithm is slow or fast.
- Space complexity of an algorithm means the amount of space (memory) taken by an algorithm. By computing space complexity we can analyze whether an algorithm requires more or less space.
- To choose the best algorithm, we need to check efficiency of each algorithm. The efficiency can be measured by computing time complexity of each algorithm. Asymptotic notation is a shorthand way to represent the time complexity.
- Using asymptotic notations we can give time complexity as "fastest possible", "slowest possible" or "average time".
- Various notations such as Ω, Θ and O used are called **asymptotic notions.**

Big Oh Notation

The **Big oh** notation is denoted by 'O'. It is a method of representing the **upper bound of algorithm's running** time. Using big oh notation we can give longest amount of time taken by the algorithm to complete.

Definition

Let F(n) and g(n) be two non-negative functions.

Let n_0 and constant c are two integers such that n_0 denotes some value of input and $n > n_0$. Similarly c is some constant such that $c > 0$. We can write

$$f(n) \leq c*g(n)$$

then f(n) is big oh of g(n). It is also denoted as $f(n) \in O(g(n))$. In other words f(n) is less than g(n) if g(n) is multiple of some constant c.

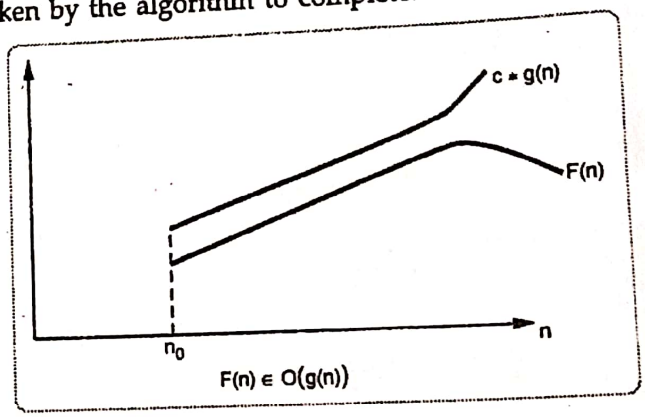


Fig. 7.6.1

Example 7.6.1 *Consider function f(n) = 2n + 2 and g(n) = n^2. Then we have to find some constant c, so that f(n) ≤ c * g(n). As F(n) = 2n + 2 and g(n) = n^2 then we find c for n = 1 then*

Solution :

$$f(n) = 2n+ 2 = 2(1) + 2$$

$$f(n) = 4$$

and $$g(n) = n^2 = (1)^2$$

$$g(n) = 1$$

i.e. $$f(n) > g(n)$$

If n = 2 then,

$$f(n) = 2(2) + 2 = 6$$

$$g(n) = (2)^2$$

$$g(n) = 4$$

i.e. $$f(n) > g(n)$$

If n = 3 then,

$$f(n) = 2(3) + 2 = 8$$

$$g(n) = (3)^2$$

$$g(n) = 9$$

i.e. $f(n) < g(n)$ is true.

Hence we can conclude that for $n > 2$, we obtain

$$f(n) < g(n)$$

Thus always upper bound of existing time is obtained by big oh notation.

7.6.1 Order of Growth

Measuring the performance of an algorithm in relation with the input size n is called order of growth. For example, the order of growth for varying input size of n is as given below.

n	log n	n log n	n^2	2^n
1	0	0	1	2
2	1	2	4	4
4	2	8	16	16
8	3	24	64	256
16	4	64	256	65,536
32	5	160	1024	4,294,967,296

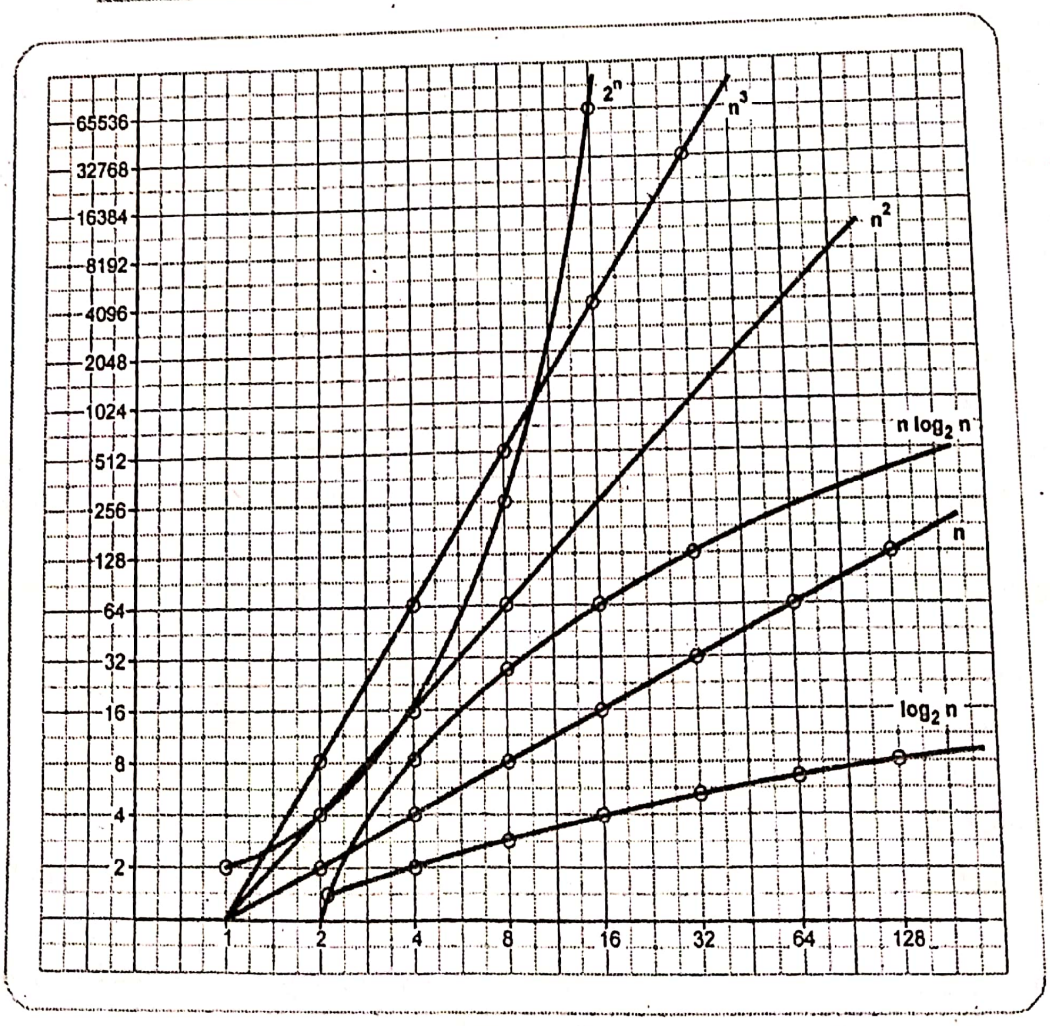


Fig. 7.6.2 Rate of growth of common computing time function

From the above drawn graph it is clear that the logarithmic function is the slowest growing function. And the exponential function 2^n is fastest and grows rapidly with varying input size n. The exponential function gives huge values even for small input n. For instance : For the value of n=16 we get $2^{16} = 65536$.

7.7 The Classes of P and NP

There are two groups in which a problem can be classified. The first group consists of the problems that can be solved in polynomial time. For example : searching of an element from the list O(logn), sorting of elements O(logn).

The second group consists of problems that can be solved in **non-deterministic polynomial time**. For example : Knapsack problem $O(2^{n/2})$ and 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 (minimum or maximum) is called optimization problem. The algorithm for optimization problem is called **optimization algorithm.**
- **Definition of P** - Problems that can be solved in polynomial time. ("P" stands for polynomial). The polynomial time is nothing but the time expressed in terms of polnomial.
  Examples - Searching of key element, Sorting of elements, All pair shortest path.
- **Definition of NP** - It stands for "non-deterministic polynomial time". Note that NP does not stand for "non-polynomial".
- Examples - Travelling Salesperson problem, Graph coloring problem, Knapsack problem, Hamiltonian circuit problems.
- The NP class problems can be further categorized into NP-complete and NP hard problems.

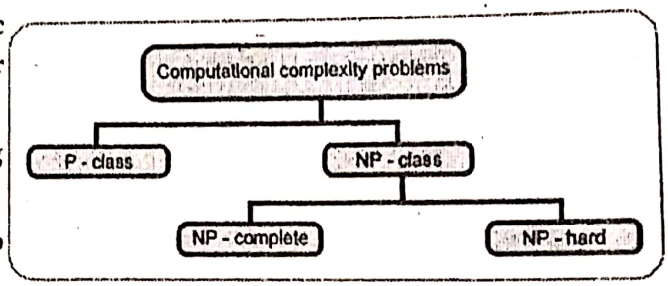


Fig. 7.7.1 Complexity classes

- A problem D is called **NP-complete** if -
  i) It belongs to class NP
  ii) Every problem in NP can also be solved in polynomial time.
- If an 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 cannot be NP-complete.
- The NP class problems are the decision problems that can be solved by non-deterministic polynomial algorithms.
- **Computational complexity** - The computational problems is an infinite collection of instances with a solution for every instance.

In computational complexity the solution to the problem can be "yes" or "no" type. Such type of problems are called **decision problem.** The computational problems can be **function problem.** The function problem is a computational problem where single output is expected for every input. The output of such type of problems is more complex than the decision problem.

The computational problems also consists of a class of problems, whose output can be obtained in polynomial time.

- **Complexity classes** - The complexity classes is a set of problems of related complexity. It includes function problems, P classes, NP classes, optimization problem.
- **Intractability** - Problems that can be solved by taking a long time for their solutions is known as intractable problems.

NP is not same as P then NP complete problems are called intractable problems.

ifference between P class and NP class problems

P Class	NP Class Problem
Problems that can be solved in polynomial time are called P-class problems.	Problems that can be solved in non-deterministically Polynomial time are called NP class problem.
These are simple to solve.	These are complex to solve.
Examples : Searching an element from unordered list, sorting the elements.	Examples : Traveling salesperson problem, knapsack problem.

7.8 Quantum Computation

Quantum computation is the area of study that focuses on development of computer technology based on the principle of quantum theory.

7.8.1 Quantum Computers

- A classical computer has a memory mode up of bits. Each bit represents either a zero and one.
- The quantum computers maintain a sequence of qubits.
- The quantum bit or simply qubit that can be described mathematically as

$$|\psi> = \alpha|0>+\beta|0>$$

- The classical bit has two states 0 and 1. Two possible states for a qubit are the states $|0>$ and $|1>$. Note tha the notation $|\ >$ is used to represent qubit.
- The qubit can be in infinite number of states other than $|0>$ and $|1>$. It can be in state $|\psi> = \alpha|0>+\beta|0>$

  Where α and β are complex numbers such that $|\alpha|^2+|\beta|^2 = 1$.

The 0 and 1 are called the computational basis states and $|\psi>$ is called superposition.

- The $|\psi> = \alpha|0>+\beta|0>$ is **quantum state.**
- In case of classical computers, we can observe 0 or 1 but it is not possible to determine the **quantum state** observation only.
- Multiple qubits can be defined. For example - two qubit system has four basis states $|00>$, $|01>$, $|10>$, a $|11>$.
- The quantum states can be

$$|\psi| = \alpha_{00}|00>+\alpha_{01}|01>+\alpha_{10}|10>+\alpha_{11}|11> \text{ with}$$

$$|\alpha_{00}|^2 + |\alpha_{01}|^2+|\alpha_{10}|^2+|\alpha_{11}|^2 = 1$$

- It is also possible to define logical gate using qubit. The classical NOT gate changes 0 to 1 and 1 to 0. In case of qubit NOT gate $\alpha|0> + \beta|1>$ is changed to $\alpha|1> + \beta|0>$.
- The action of qubit NOT gate can be described using matrix as -

$$\begin{bmatrix}\alpha \\ \beta\end{bmatrix} = \begin{bmatrix}0 & 1 \\ 1 & 0\end{bmatrix}\begin{bmatrix}\alpha \\ \beta\end{bmatrix} = \begin{bmatrix}\beta \\ \alpha\end{bmatrix}$$

- Thus the quantum computer can be defined as

> A quantum computer is a system built from quantum circuits, containing wires and elementary quantum gates, to carry out manipulation of quantum information.

7.8.2 Church-Turing Thesis

Hypothesis means proposing certain facts. The Church's hypothesis or Church's Turing thesis can be stated as,

> "The assumption that the intuitive notion of computable functions can be identified with partial recursive functions".

However, this hypothesis cannot be proved. The computability of recursive functions is based on following assumptions -

1) Each elementary function is computable.
2) Let f be the computable function and g be the another function which can be obtained by applying an elementary operation to f, then g becomes a computable function.
3) Any function becomes computable if it is obtained by rule 1) and 2).