UNIT 1.
ALGORITHMS & PROBLEM SOLVING.

Q1) Define an algorithm. Discuss different characteristics of a good algorithm.

→ • **Algorithm.**

- An algorithm is a finite set of unambiguous steps needed to be followed in certain order to accomplish a specific task.
- In case of computational algorithms, it refers to the instructions that contain fundamental operators like +, -, *, /, % etc.
- They provide a precise description of the procedure to be followed in certain order to solve a well-defined computational problem.

• **Characteristics of an good algorithm:**

a) **Inputs:**
Each instruction that contains a fundamental operator must accept zero or more inputs.

b) **Output**
Every instruction that contains a fundamental operator must ~~accept~~ give zero or more outputs.

c) **Definiteness.**
- All instructions in an algorithm must be unambiguous, precise & easy to interpret.
- Every fundamental operator in instruction must be defined without any ambiguity.

d) **Finiteness.**
- An algorithm must terminate after a finite no. of steps in all test cases
- Every instruction which contains a ~~finite~~ fundamental operator must be terminated within finite amount of time.

e) Effectiveness

An algorithm must be developed using very basic, simple & feasible operations so that one can trace it out by using paper & pencil.

- A good algorithm never compromises the precise & unambiguous requirements at every step.
- The same problem can be solved by different algorithmic properties / strategies.
- Same algo. can be specified in multiple ways.
- The legitimate inputs of an algorithm should be well specified

Q2) Write a short note on algorithm as a technology with example.

→
- Computing time & memory space are limited resources so we should use them sensibly.
- It is achieved by the usage of efficient algorithms that need less space & time.
- This problem can be solved by different algorithms which exhibit radical differences in their efficiency.
- Eg. Consider the following where Computer 1 is 100 times faster than computer 2.
  - Two different sorting algos are executed on these two computers to sort 10^6 numbers.
  - It shows that even with 100 times slower compiler, Comp 2 completes the sorting of 10^6 nos. 20 times faster than Comp 1.
  - This is achieved because of the usage of an efficient algo. heap sort.
- Many other computer technologies need the knowledge of algorithms.
- Eg. LAN, Web designing, AI, Robotics, etc.

Q3) What are the different types of problems in computation?

→ 1) Searching Problems.
- include searching any item / search key in given data.
- Eg. retrieving info. from large databases.

2) Sorting Problems
- include rearrangement of items in given data in ascending or descending order.
- Eg. Arranging names in alphabetical order.

3) String processing
- include the computation on strings. (sequence of characters)
- Eg. String encoding.

4) Graph problems
- include processing of graphs.
- graph is a collection of points & line segments connecting them.
- Eg. graph traversal.

5) Combinatorial problems.
- these problems explicitly / implicitly ask to find a combinatorial object like a permutation / combination or subset that satisfies the specified constraints.
- Eg. 8-queens problem.

6) Geometric Problems
- problems which deal with geometric objects such as points, lines & polygons.
- Eg. convex-hull problem.

7) Optimization problem.
It is a computational problem that determines optimal value of a specified cost function.
- Eg. Travelling Salesman Problem.

8) Tractable problem.
- It is solved ~~thorig~~ in polynomial time using deterministic algorithms
Eg. Merge sort.

10) Intractable problem.
- cannot be solved in polynomial time using deterministic algos.
- Eg: Knapsack problem.

Q4) State & explain various stages in problem solving.

→ 1) Identifying the problem
- To find the solution to any real world problem we must understand the problem & its constraints.

2) Designing a computational mathematical model
- It presents the abstraction of real world problems.
- removes unnecessary & irrelevant data from problem description & simplifies it to get a precise computational model.

3) Data Organization
- The essential data needed must be organized effectively.

4) Algorithm designing
By analyzing the problem we should design a finite set of unambiguous steps to get the solution.

5) Algorithm specification.
- It is the way of describing the algorithmic steps for the programmer, & which is written in the form of pseudo-code.

6) Algorithm validation
After defining the algorithmic steps we should validate our logic. It checks whether algorithm produces the correct output in finite amount of time.

7) Analysis of algorithm.
The ~~correctness~~ performance of the correct algorithm is analyzed & its efficiency is checked by checking usage of memory, time etc.

8) The correctness & efficiency of algo. for inputs are verified through mathematical proof.

9) Implementation
By referring to the specifications of a verified algorithm a correct computer programme is written using a specific programming language & technology.

10) Testing & debugging
- The computer program written for a specific algo. is ~~written~~ executed on a machine.
- It is tested for all inputs & debugged to trace the expected workflow.

11) Documentation
The details of the solved problem, its algo, analysis, proof of correctness, implementation, test cases is well documented for future applications & research.

Q5) What are the different types of algorithmic strategies?

→ i) Brute force method.
- It enumerates all possible solutions to a given problem without applying any heuristics

ii) Exhaustive search
It generates all candidate solutions to a ~~co~~ given combinatorial problem & identifies feasible solution.

iii) Divide & conquer.
By applying top-down approach it divides a large problems in small parts, solve the sub-problems independently & then combine their solutions.

iv) Greedy method.
It builds solution in stages. At every stage it selects the best choice concerning the considerations.

v) Dynamic programming.
Suitable for solving optimization problems with over-lapping sub-problems.

6) Backtracking
It is an algorithmic strategy that explores all solutions to a problem & abandons them if they are not fulfilling.

7) Branch & Bound.
It is a state space algo. where ~~E-bound~~ an E-node remains an E-node until it is dead.

8) Exotic algorithms like genetic algorithms, online algorithms, parallel algorithms etc.

Q6) List types of algorithms & classic problems solved by each of them.

→ 1) Brute force method
- Sequential search
- Bubble sort
- N-queens problem

2) Divide & conquer
- Binary search
- Merge sort
- Quicksort

3) Greedy algorithms
- Job scheduling problem
- Fractional knapsack problem
- Activity selection problem

4) Dynamic programming
- Travelling salesperson problem
- Multistage graph problem

5) Backtracking
- N-queens problem.
- Graph coloring
- 0/1 Knapsack problem.

Q7. What are the rules of writing a pseudocode?

→ • Comments are given by // (single line) or /* .... */ (multiline)
• Each collection of simple statements is described as a block enclosed in { }.
• Simple data types like int, float, char, boolean etc are assumed & not specified explicitely.
• To assign a value to a variable, assignment operator (:=) is used.
• The arithmetic operators are: +, −, *, /, %. etc
• The logical operators are: && (logical and), || (logical or), ~ (logical not)
• Relational operators are: <, ≤, ≥, ≠, =, >.
• An array is given by [ ].
• Boolean variables have TRUE & FALSE values.

Q8. Why is correctness of an algorithm important?

→ • For any problem, we get the correct solution by performing valid logical computations.
• Once the problem is cracked we can think of better solution to solve the same problem in a more efficient way.
• If algo. is incorrect then the efforts for improving efficiency will be in vain.
• ∴ before improving efficiency we confirm correctness.
• To test the correctness of an algorithm we can give several sets of valid inputs to an algo. & compare the resulting outputs with known/manually computed results.
• It needs rigorous proofs based on mathematical & logical reasoning for confirming the correctness of algos.
• Such proofs not only provide us with more confidence in working of our algo. but also help us to rectify subtle errors in the algo.

Q9. How to confirm the correctness of an algorithm? Explain with example.

→ The basic steps for confirming the correctness of an algorithm:

i) Give the statement a postcondition to be satisfied.
A postcondition is a predicate based on an expected output of the algorithm.

ii) Assume the necessary preconditions / assumptions to be TRUE.

iii) Apply a chain of logical & mathematical reasoning from preconditions to satisfy the postcondition

Q10. What is a loop Invariant property? Explain with an example.

→
- A loop invariant property is a property of an iterative algorithm.
- The loop invariant relation is true before, after & during all iterations of the iterative algorithm.
- It defines the goal or the desired output of the iterative algo
- After each iteration of a loop, it gives an idea about the current progress towards the final output.
- Thus loop invariant property is very important in understanding outcome of a loop.
- We can also check the correct working of the loop.

UNIT - 2
Analysis of Algorithms & Complexity Theory.

Q1) What is algorithm analysis framework?

→ The analysis framework includes:

i) Computing best case, worst case & average case efficiencies.
ii) Measuring input size
iii) Measuring running time
iv) Computing orders of growth of algorithms.
v) Measuring time complexity
vi) Measuring space complexity.

- The efficiency of an algorithm can be decided by measuring the performance of an algorithm.
- We can measure the performance of an algorithm using 2 factors:

a) Amount of time taken by an algorithm to execute
b) Amount of storage required by the algorithm.

- It is also known as time complexity & space complexity.

Q2) What is space complexity? What are its basic components?

→
- Space complexity can be defined as the space ~~task~~ / memory required by an algorithm to run.
- To compute the space complexity we use two factors: Constant & instance characteristics.

$$S(p) = C + Sp$$

where C = fixed part which denotes the space of inputs & outputs
Sp = space dependent on instance characteristics.

- There are two types of components which contribute to space complexity - fixed part & variable part.

- The fixed part includes space for instructions, variables, array size & space for constants.
- The variable part includes space for variables whose size is dependent upon the particular problem instance being solved.
  Eg - control statements (for, do, while, etc)
- For example,
  Compute the space complexity of the following algorithm:

```
Algo:
Sum(a, n)
{
    s = 0.0;                    ---------- O(1) [Constant]
    For i := 1 to n do          ---------- O(n) [n times exec]
        s := s + a[i];          ---------- O(n) [n times exec]
    return s                    ---------- O(1) [runs once]
}
```

∴ Space complexity = O(n).

Q3. Explain time complexity of an algorithm.

→ .
- Time complexity of an algorithm is defined as the amount of time required by an algorithm to run to completion.
- As it is difficult to compute time using physically clocked Times it is given in terms of frequency count.
- Frequency count is defined as count which denotes how many times a particular statement is executed.
- In multiuser system, executing time depends upon factors such as:

a) System load
b) No. of other programs running
c) Instruction set used
d) Speed of underlying hardware.

Eg. Obtain the frequency count for the following.

```
void fun()
{
  int a
  a=0;                          ---------- (1) time
  for (i=0; i<n; i++)           ---------- (n+1) times
  {
    a = a+i;                    ---------- n times
  }
  printf("%d, a);               ---------- 1 time
}
```

Time frequency = $1 + (n+1) + n + 1$

$= 2n + 3$.

The for loop is executed 'n' times when condition is true & once more when it becomes false. ∴ for loop's frequency count is n+1. Statement inside loop will be executed if the condition inside the loop is true. ∴ It is executed n times.

- The Big oh notation is most commonly used algorithmic notation.
- The order of the magnitude of the polynomial is considered.
- The higher order polynomial is always considered.

Q4) What are Asymptotic Notations?

→ • Asymptotic notation is a shorthand way of representing the time complexity.

- Using asymptotic notations, we can give time complexity as "fastest possible", "slowest possible" & "average time".
- Various asymptotic notations are:

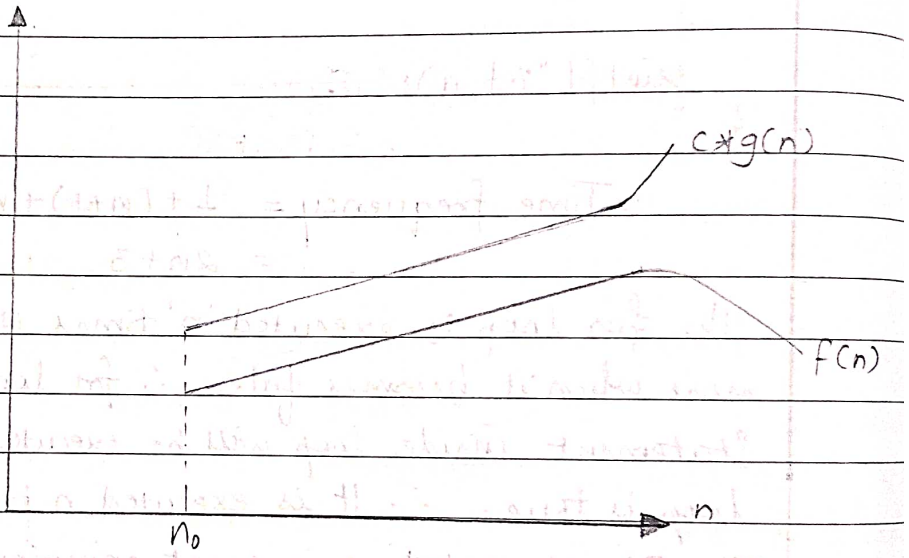
a) Ω Omega

b) Θ Theta

c) O Oh.

Q5) Explain Big Oh Notation.

→
- Big Oh notation is denoted by 'O'.
- It is a method of representing upper bound of algorithm's running time.
- We can give the longest amount of time taken by the algorithm to complete.
- Representation.



- Let $f(n)$ & $g(n)$ be two non-negative functions.
- Let n_0 & constant c be two integers such that $n > n_0$ & $c > 0$.
- n_0 denotes some value of input & so does c.
- ∴ We can write

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

then $f(n)$ is big Oh of $g(n)$.

$$f(n) \in O(g(n))$$

Eg- $f(n) = 2n+2$

$g(n) = n^2$

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

∴ consider $n = 1$

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

$= 4$

$g(n) = n^2$

$= 1^2$

$= 1$

$\therefore f(n) > g(n)$

for $n = 2$

$f(2) = 4+2$

$= 6.$

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

$= 4$

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

for $n = 3$

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

$= 8$

$g(3) = 3^2$

$= 9.$

$\therefore f(n) < g(n)$

$\therefore$ for $n > 2$, we get $f(n) < g(n)$.

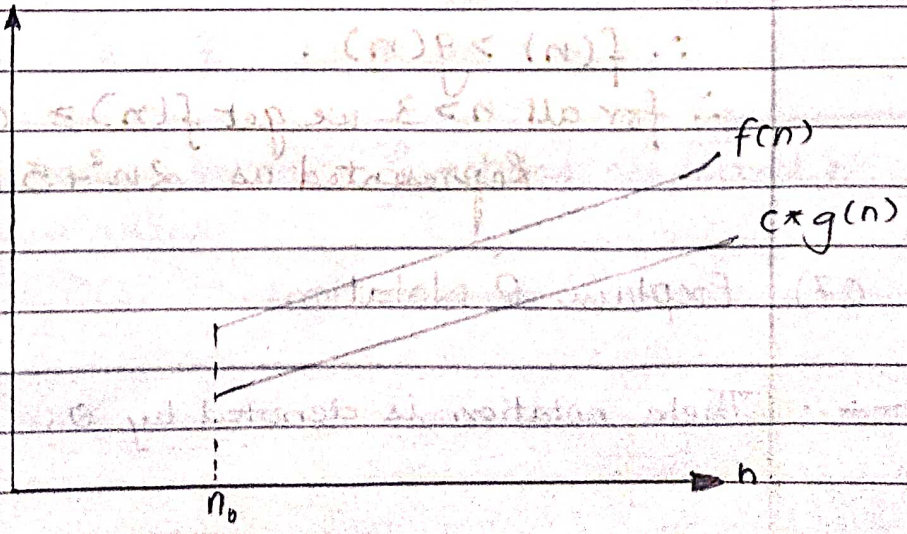
Q6) Explain Omega Notation.

→ • It is represented by 'Ω'.

• This notation is used to represent lower bound of algorithm's running time.

• It is used to denote shortest amount of time taken by the algorithm.

• Representation.



- A function $f(n)$ is ~~used~~ said to be in $\Omega(g(n))$ if $f(n)$ is bounded below by some positive constant multiple of $g(n)$ such that

$$f(n) \geq c * g(n) \quad \text{for all } n \geq n_0.$$

- It is denoted as $f(n) \in \Omega(g(n))$

Eg. $f(n) = 2n^2 + 5$

$g(n) = 7n$.

for $n = 0$

$f(0) = 2(0)^2 + 5$

$= 5$

$g(0) = 0$.

$f(0) > g(0)$.

$f(n) > g(n)$

But if $n = 1$

$f(1) = 2(1)^2 + 5$

$= 7$

$g(1) = 7$

$\therefore f(n) = g(n)$.

If $n = 3$,

$f(3) = 2(3)^2 + 5$

$= 2(9) + 5$

$= 23$

$g(3) = 7(3)$

$= 21$

$\therefore f(n) > g(n)$.

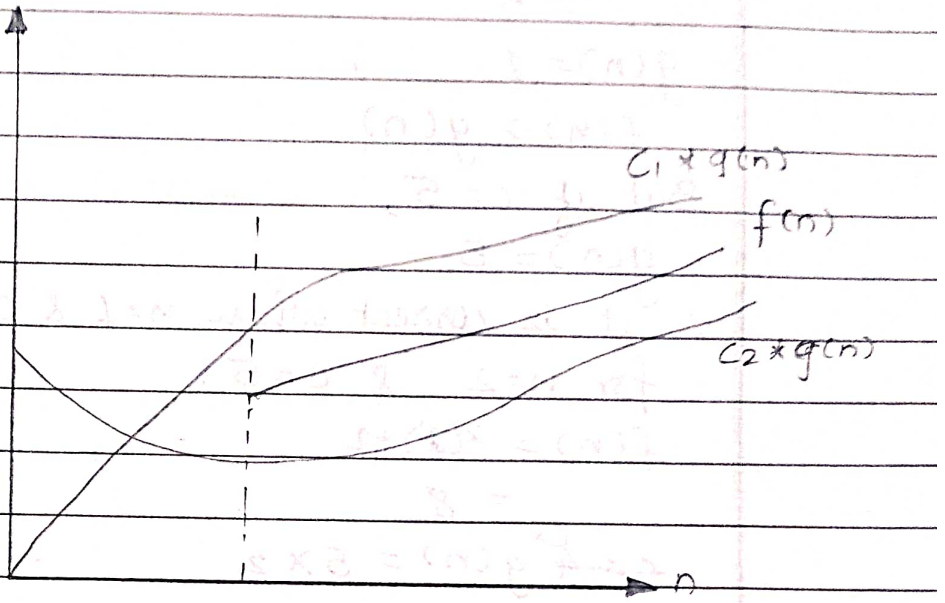
$\therefore$ for all $n > 3$ we get $f(n) \geq c * g(n)$.

Represented as $2n^2 + 5 \in \Omega(n)$.

Q7) Explain θ Notation.

→ . Theta notation is denoted by θ.

- By this method, running time is between upper bound & lower bound
- Representation:



- Let $f(n)$ & $g(n)$ be two non negative functions.
- There are two positive constants namely c_1 & c_2 such that

$$c_2 g(n) \leq f(n) \leq c_1 g(n)$$

Then we say

$$f(n) \in \Theta(g(n))$$

Eg.

If $f(n) = 2n+8$ & $g(n) = 5n$

where $n \geq 2$

similarly, $f(n) = 2n+8$

$g(n) = 7n$

$5n < 2n+8 < 7n$ for $n \geq 2$.

Here $c_1 = 5$ & $c_2 = 7$ with $n_0 = 2$.

Q8. State whether the following ~~questions~~ functions are correct or incorrect & justify your answer.

1) $3n+2 = O(n)$

$f(n) = 3n+2$

$g(n) = n$.

for $n=1$

$f(n) = 3+2$

$= 5$

$g(n) = 1$

$f(n) > g(n)$

But if $c=5$,

$g(n) = 5$

$\therefore$ It is correct when $n=1$ & $c=5$

for $n=2$ & $c=5$.

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

$= 8$

$g(n) = 5 \times 2$

$= 10$.

$f(n) \leq g(n)$.

if $n=0$,

$f(n) = 3n+2$

$= 3(0)+2$

$= 2$

$g(n) = 0c \cdot g(n) = 5 \times 0 = 0$.

$\therefore$ It is incorrect for $n=0$.

---

2) $100n+6 = O(n)$.

$f(n) = 100n+6$

$g(n) = n$.

for $n=1$,

$f(n) = 100(1)+6$

$= 106$.

$g(n) = 1$

$c = 106$

$\therefore g(n) = 106$.

$\therefore f(n) \leq c \cdot g(n)$ is true.

for $n_0 = 2$

$f(n) = 100(2) + 6$

$= 206$

$c \cdot g(n) = 106 \times 2$

$= 212$

$\therefore f(n) \leq c \cdot g(n)$ is true

for $n_0 = 0$,

$f(n) = 100(0) + 6$

$= 6$

$c \cdot g(n) = 106 \cdot 0$

$= 0$.

$f(n) \geq c \cdot g(n)$

$\therefore$ It is incorrect.

---

3) $10n^2 + 4n + 2 = O(n^2)$

$f(n) = 10n^2 + 4n + 2$

$g(n) = n^2$

for $n_0 = 1$

$f(n) = 10(1)^2 + 4(1) + 2$

$= 100 + 4 + 2$

$= 106$

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

$= 1$

$c \cdot g(n) = 106$.

$f(n) = c * g(n)$ is true.

for $n_0 = 2$

$f(n) = 100(2)^2 + 4(2) + 2$

$= 400 + 8 + 2$

$= 410$.

$c * g(n) = 106 \times 4$

$= 424$

$\therefore f(n) \leq c * g(n)$ is true.

if $n=0$,
$100n^2+4n+2 \neq O(n^2)$.

Q9) Interpret the following.

a) $2n^2+3n+1 = 2n^2+\Theta(n)$.

$2n^2+3n+1 = 2n^2+\Theta(n)$.

$3n+1 = \Theta(n)$.

$f(n) = 3n+1$

$g(n) = \Theta(n)$

If we have $c_2=1$ $c_1=4$ for $n \geq 1$,

If $n=1$ $c_1=4$ $c_2=1$

$c_2 g(n) \leq f(n) \leq c_1 g(n)$

$1\, c_2 g(n) \leq 3n+1 \leq 4\, g(n)$

$1(1) \leq (3\times1+1) \leq 4$

$1 \leq 4 \leq 4$ is true

Similarly

for $n=2$ $c_1=4$, $c_2=1$,

$c_2 \cdot g(n) \leq f(n) \leq c_1 g(n)$

$1(2) \leq (3\times2+1) \leq 4(2)$

$2 \leq 7 \leq 8$ is true.

Q10) Explain Best case, worst case & average case complexity of an algorithm.

→ a) Best case complexity.

- If an algorithm takes minimum time to run to completion for a specific set of input, it is called best case time complexity.
- In best case time complexity, algorithm runs for a short time.

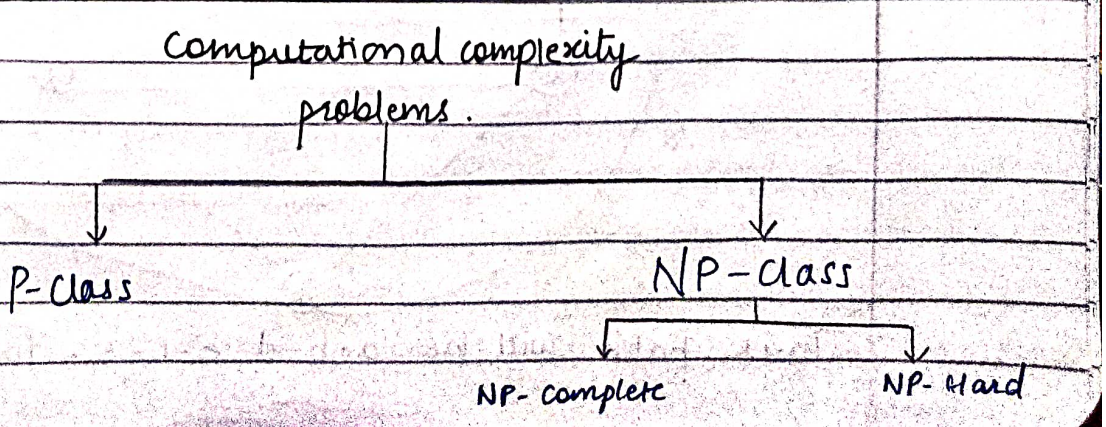
Eg.
- In a searching algorithm, the element key is searched from list of n elements.
If the key element is present at first location in the list then algo. runs for a short time & therefore will get best case time complexity.

$$C_{best} = 1.$$

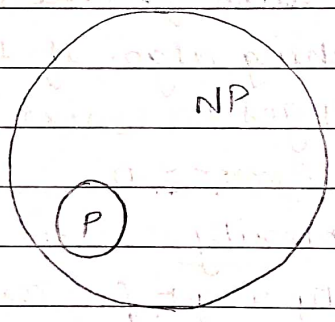
b) Worst Case complexity. (Upper Bound)
- If an algorithm makes maximum amount of time to run to completion for specific set of input, it is known as as worst case complexity.
- In this, algorithm runs for the longest time.
- Eg - In a searching algo. if the key to be found is at n^{th} location, we will get the worst time complexity.

$$C_{worst} = n.$$

c) Average case Complexity.
- The time complexity we get for certain set of inputs is as a average same.
- Then for corresponding input, such time complexity is called average case time complexity.
- For calculating average case time complexity we have to consider probability of getting success of the operation.
- Any operation in the algorithm heavily depends on the input elements
- Thus computing avg. case time complexity is difficult than computing worst case & best case time complexities.



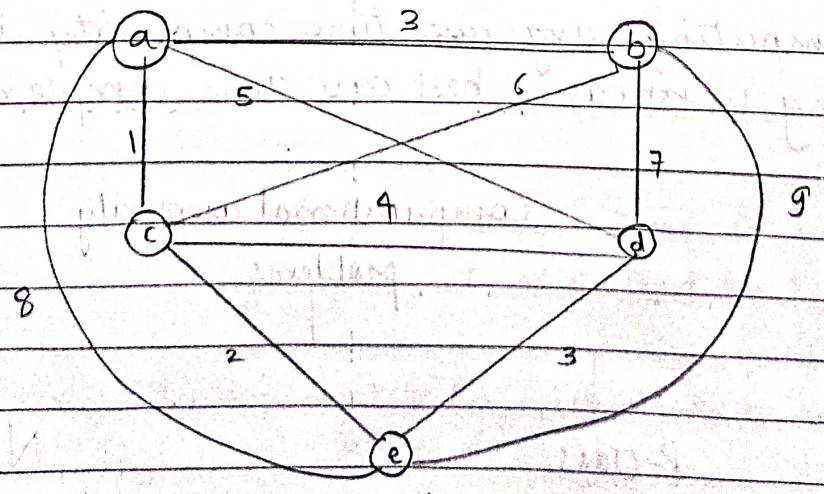
- **P class**
  - Problems that can be solved in polynomial time.
  - The polynomial time is nothing but time expressed in terms of polynomial
  - Eg. Searching of key element, Sorting of elements.
- **NP class**
  - It stands ~~so~~ for 'non-deterministic polynomial time."
  - NP does not stand for "non-~~deterministic~~ Polynomial".
  - The NP class problems are those problems that can be solved in non-deterministic polynomial time but can be verified in polynomial time.



The P & NP class.

Eg - Travelling Salesman Problem. (NP)

This problem can be stated as "Given a set of cities & ~~plan~~ cost to travel between each pair of cities, determine ~~the~~ whether there is a path that visits every city once & returns to the first city such that the cost travelled is less".



Tour path will be a-b-d-e-c-a. Total cost of tour = 16.

- This problem is an NP problem as there may exist some path with shortest distance between cities.
- If you get solution by applying certain algorithm, then Travelling salesman problem is NP complete Problem.
- If we get no solution at all, problem belongs to NP hard class.

- Ex. of P class problem.
- Kruskal's algorithm
- In Kruskal's algorithm the minimum weight is obtained.
- In this algo., a circuit should not be formed.
- Each time the edge of minimum weight has to be selected from the graph.
- It is not necessary to have edges of minimum weights to be adjacent

Q11. Difference between P & NP class Problems.

	P class	NP class
a)	An algorithm in which for given input, definite output is generated is called Polynomial time algorithm (P class)	An algorithm is called non-deterministic polynomial time algo when for given input there are more than one paths that the algorithm can follow, due to which one cannot ~~follow~~ determine which path to follow after a stage.
b)	All P class problems are deterministic.	All NP class problems are basically non-deterministic.
c)	Every problem which is in P class is also in NP class.	Every NP class problem is not a P class problem.
d)	P class problem can be solved efficiently Eg- Binary Search	NP cannot be solved as efficiently as P class Eg- Knapsack problem

Q12. List three problems that have polynomial time algorithms. Justify your answer.

→ • The problems which can be solved in polynomial time are called P-class problems.

1) Binary search
In searching an element using Binary search method, the list is simply divided at the mid & either left or right sublist is searched for key element.
This process is carried out in O(log n).

2) Evaluation of polynomial.
In polynomial evaluation we make out the summation of each term of polynomial.
The evaluated result is simply an integer. This process is carried out in O(n) time.

3) Sorting a list
The elements in a list can be arranged either in ascending or descending order.
The procedure is carried out in O(n log n) time.

• This shows that all the above problems can be solved in polynomial time.

Q13. What are deterministic & non-deterministic algorithms?

→ • The algorithm in which every operation is uniquely defined is called deterministic algorithm.

• The d algorithm in which every operation may not have unique result rather than can be specified set of possibilities for every operation is known as non-deterministic algorithm.

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

- The non-deterministic algorithm has two stages:

a] ~~Guessing~~ Verification stage

In this stage it takes input the candidate solution & the instance to the problem & returns yes if the candidate solution represents actual solution.

b] Verification stage ~~Guessing~~ Guessing stage

In this stage, an arbitrary string is generated which can be thought of as a candidate solution.

Q14. What is 0-1 Knapsack problem? Explain the algorithm as non-deterministic algorithm.

→ - The 0-1 Knapsack Decision Problem is to determine, for a given profit P, whether it is possible to load the knapsack so as to keep the weight ~~to~~ no greater than m, while making total profit at least equal to Pt.
- This problem has the same parameters as the 0-1 Knapsack optimization ~~Ptp~~ problem plus the additional parameter Pt.
- Non-deterministic algorithm for 0/1 knapsack problem:

```
Non_DKnaps()
//p ~~[1:1]~~ [1:n] is a profit object i where 1 ≤ i ≤ n
//w [1:n] is the weight of each object i where 1 ≤ i ≤ n
// Profit & Wt represent the current total profit & weight
//m is the capacity of knapsack.
{
// initially no item is selected.
wt := 0 ;
Profit := 0 ;
// guessing stage
    for i := 1 to n do
    {
        x[i] := choose(0,1);
```

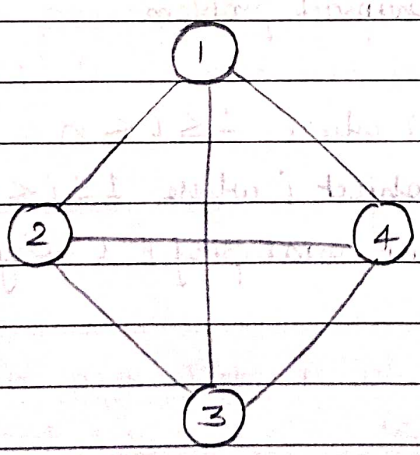
```
        Wt := Wt + x[i] * w[i];
        Profit := Profit + x[i] * p[i];
    }
// verification stage
// selected items exceed the capacity or less profit is earned
    if ((Wt > m) or (Profit < pt)) then
        fail();
    else
        success(); // maximum profit earned.
}
```

§15. Explain Clique Problem.

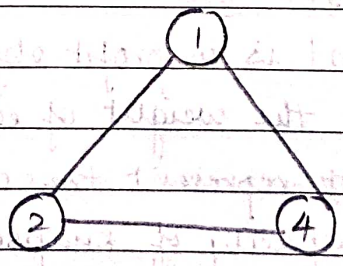
→ • Clique is nothing but maximal complete subgraph of graph G.

• The graph G is a set of (V, E) where V are the vertices & E are the set of edges.

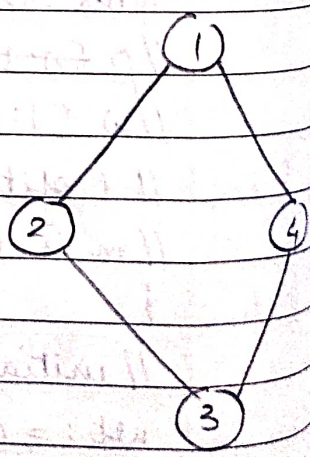
• The size of clique is no. of vertices present in it.



Graph G



Clique size = 3



Clique size = 4

• Max Clique Problem

It is an optimization problem in which the largest size Clique is to be obtained.

In the decision problem of clique we have to ~~find~~ determine whether G has ~~th~~ a clique of size at least k for given k.

- Let DClique (G, k) is a deterministic decision algorithm for determining whether graph G has a clique or not of size atleast k.
- Then we have to apply DClique for k = n, n-1, n-2... until the output is 1.
- The max clique can be obtained in time ≤ n. F(n) where F(n) is the time complexity of DClique.
- If Clique decision problem is solved in polynomial time the max clique problem can be solved in polynomial time.
- Non-Deterministic algorithm for a clique.

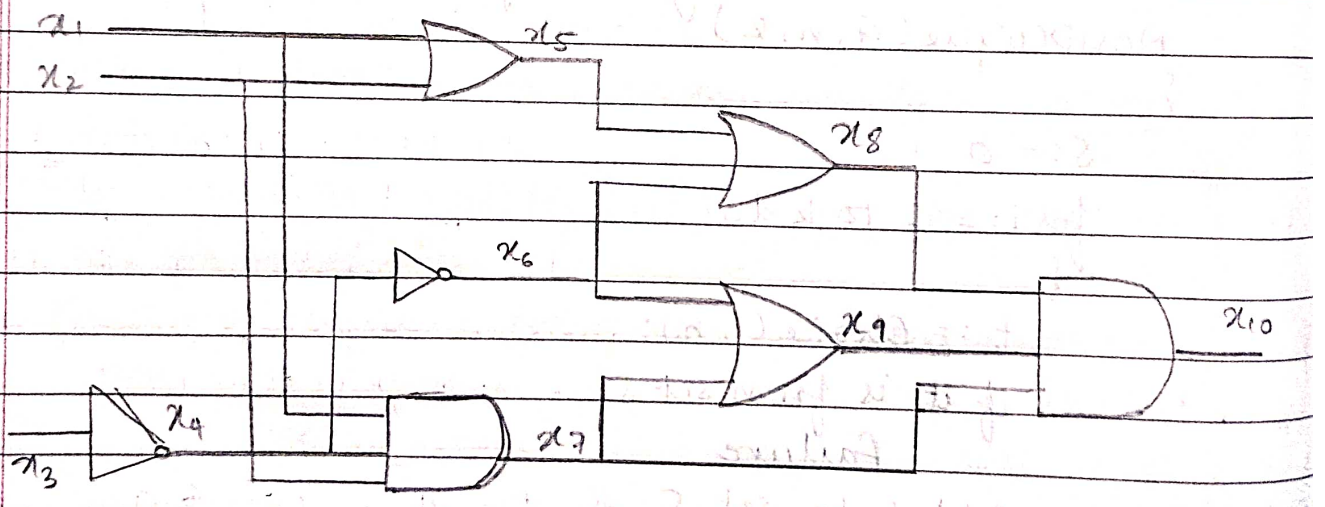
```
NonDClique(G, n, k)
{
    S := 0
    for i := 1 to k do
    {
        t := Choice(1, n);
        if t is from set S
            Failure
        Add t to set S
    }
// Now S contains k distinct vertex indices
for all pairs (i, j) i & j are from set S & i != j do
    if (i, j) is not an edge of graph G then
        Failure();
Success();
}
```

Q16. t

Q16. Prove that SAT problem is NP Complete

→ Proof

1) SAT is NP.
2) Circuit SAT reduces to SAT. The reduction function f is as follo
   i) For every input wire add a new variable
   ii) For every output wire add a new variable
   iii) An equation is prepared for each gate.
   iv) These set of variables are separated by ∧ values & adding final output variable at the end
   . This transformation can be done in polynomial time.



$x_5 = x_1 \vee x_2$

$x_4 = \neg x_3$

$x_6 = \neg x_4$

$x_7 = x_1 \wedge x_2 \wedge x_4$

$x_8 = x_5 \vee x_6$

$x_9 = x_6 \vee x_7$

$x_{10} = x_7 \wedge x_8 \wedge x_9$

$f = x_5 \wedge x_4 \wedge x_6 \wedge x_7 \wedge x_8 \wedge x_9 \wedge x_{10}$.

Q17. Prove that 3SAT is NP complete.

→ A 3-SAT problem is a problem which takes Boolean formula S with each clause having exactly 3 literals & check if S is satisfied or not.

- Proof.

The language 3-SAT is a restriction of SAT.

We replace each clause C that represents the SAT problem to a function f by a family of D_C of clauses that represents satisfiability

for eg.

$$C = a \vee b \vee c \vee d \vee e$$

One can simulate this by

$$D_C = (a \vee b \vee x) \wedge (\bar{x} \vee c \vee y) \wedge (\bar{y} \vee d \vee e)$$

where x & y are new variables.

- Need to verify.

i) If C is false, D_C is false

ii) If C is true, then one can make D_C True.

- If f is satisfiable then there is assignment where each clause C is true. This can be extended to make D_C true.
- Further if f is evaluated to False, then some clauses say C' must be false & corresponding family $D_{C'}$ evaluates to false.
- This conversion can be done in polynomial time.
- Thus we have shown that SAT problem reduces to 3-SAT in poly. time.
- As we know SAT is NP-complete, ∴ 3-SAT is also NP-complete.

Q18. What is a Vertex cover Problem?

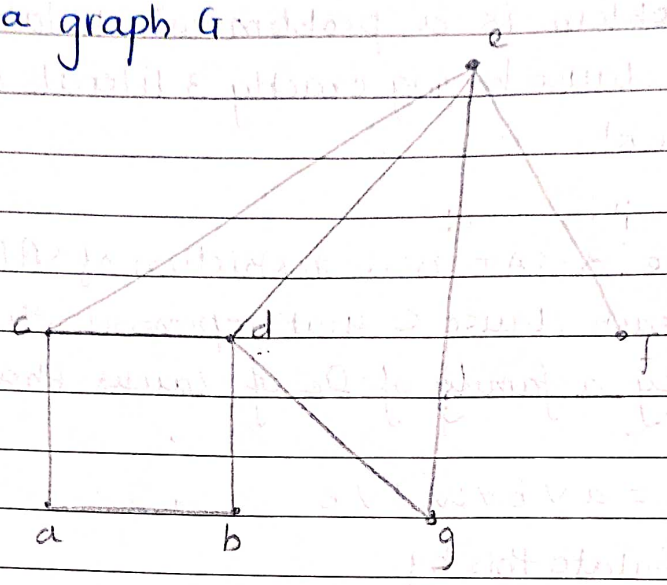
→. The vertex cover problem is to find vertex cover of minimum size from a given graph.

- The word vertex cover means each vertex covers its incident edges.
- Thus by vertex cover we expect to choose the set of vertices that

that cover all edges in a graph.

For eg.

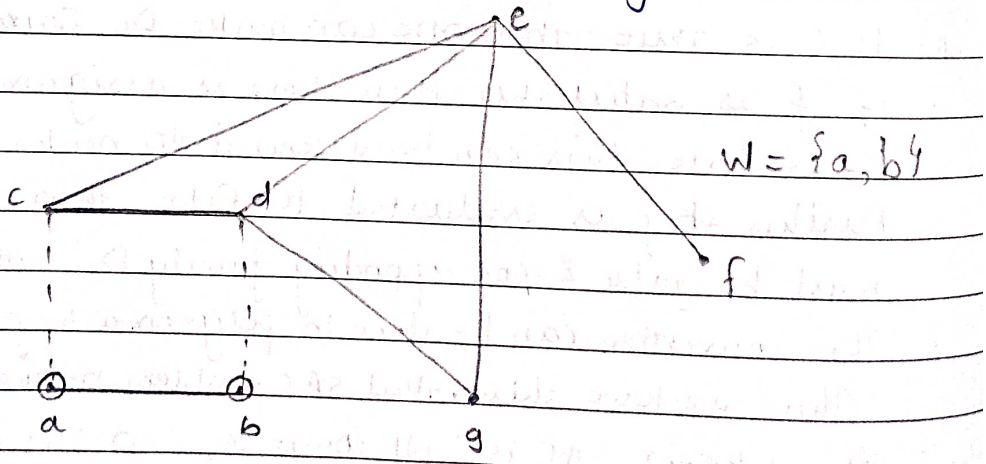
Consider a graph G:



Now we select some arbitrary edge & delete all the incide edges.

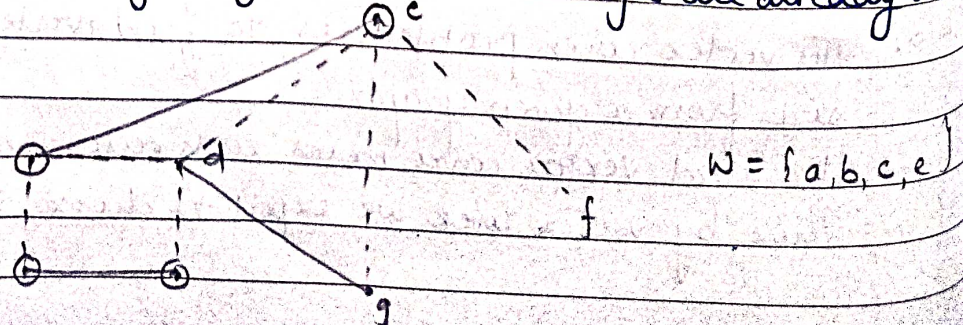
Repeat this process until all incident edges get deleted.

Step 1: Select edge a-b & delete all incident edges to vertex a or b

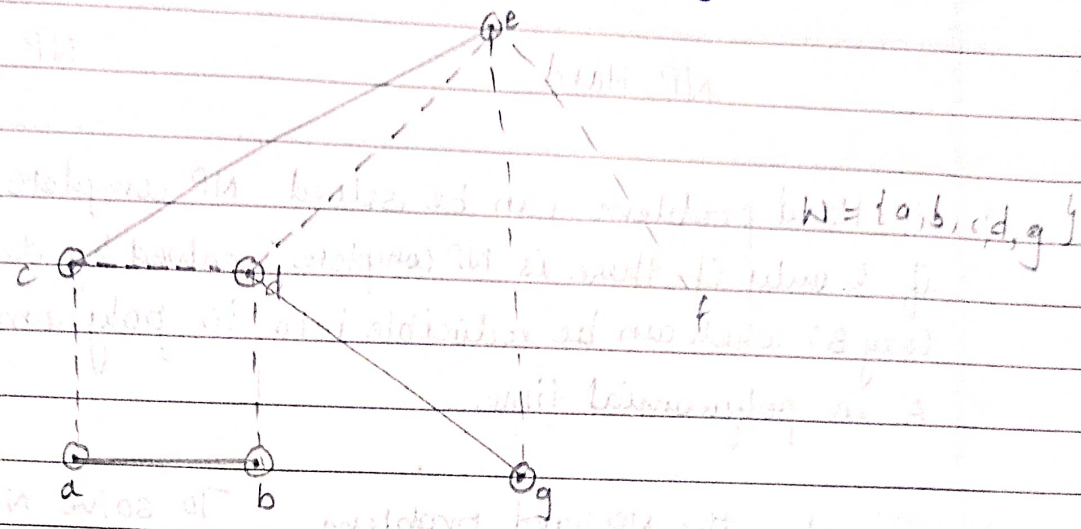


Step 2: Select edge c-e & delete all incident edges to vertex c or e.

Step 3: Select edge d-g. All incident edges are already dele

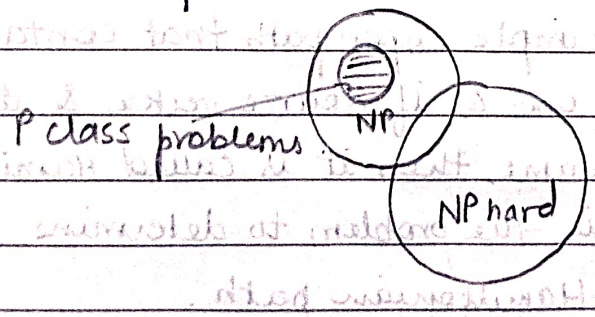


Thus we obtain vertex cover as {a, b, c, d, e, g}



Q19. What is NP-complete & NP-Hard Problem?

→ • A problem A is NP-Hard if there is an NP-complete problem B such that B is reducible to A in polynomial time.
• NP hard problems are as hard as NP-complete problems.
• NP hard problems need not be in NP class.
• An NP class problem such that, if it is in P, P = NP.
• If ~~not~~ a problem has this same property it is called "NP Hard".
• Thus, the class of NP-complete problem is intersection of NP & NP-hard problem.



• Normally, decision problems are NP-complete but optimization problems are NP hard.
• However if problem L1 is a decision problem & L2 is optimization problem then it is possible that L1 ∝ L2.
• For instance the knapsack decision problem can be knapsack optimization problem.

Q21. Differentiate between NP Hard & NP complete.

	NP Hard	NP complete
1)	NP Hard problem can be solved if & only if there is NP complete (say B) which can be reducible into A in polynomial time.	NP complete problems can be solved by deterministic algorithm in polynomial time.
2)	To solve the NP hard problem it must be in NP class.	To solve NP complete problem it must be in NP & NP hard.
3)	It is not a decision problem	It is a decision problem
4)	Eg - Halting problem, Hamiltonian cycle problem.	Eg. Circuit satisfiability problem

Q22) Explain NP Hard Hamiltonian Cycle Problem.

→ • Hamiltonian path is a simple open path that contains each vertex in a graph exactly once & if source vertex & destination vertex of the path is the same, then it is called Hamiltonian cycle.
• Hamiltonian path problem is the problem to determine whether a given graph contains Hamiltonian path.
• To show that the problem is NP-complete we first need to show that it actually belongs to class NP & then find a known NP complete problem that can be reduced to Hamiltonian path.

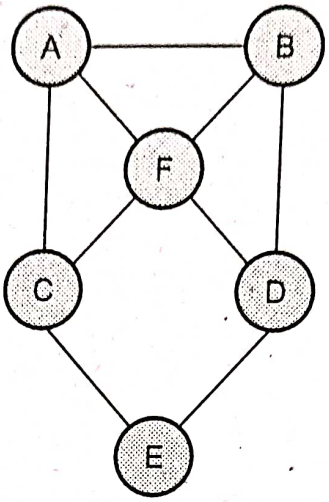
Review Question

1. *Write a short note on NP completeness of algorithm and NP hard.* **SPPU : May-18, Marks 8**

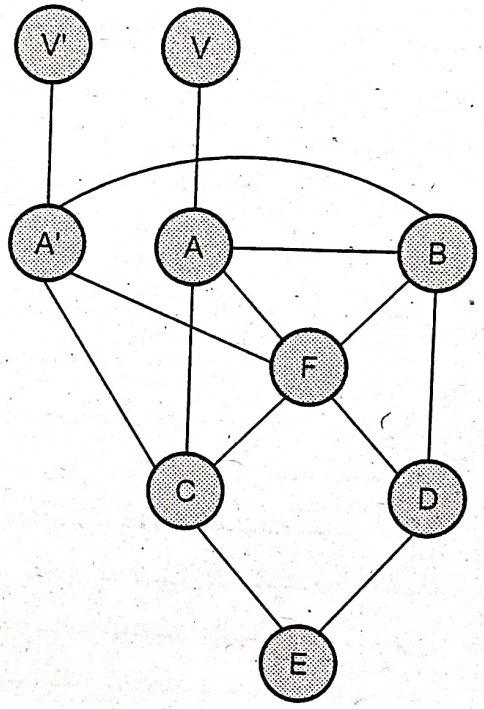
2.11 Hamiltonian Cycle

SPPU : Dec.-19, Marks 8

- Hamiltonian path is a simple open path that contains each vertex in a graph exactly once and if the source vertex and the destination vertex of this Hamiltonian path is the same then it is called Hamiltonian cycle.
- The Hamiltonian path problem is the problem to determine whether a given graph contains a Hamiltonian path.
- To show that this problem is NP-complete we first need to show that it actually belongs to the class NP and then find a known NP-complete problem that can be reduced to Hamiltonian path.
- For a given graph G we can solve Hamiltonian Path by deterministically choosing edges from G that are to be included in the path. Then we traverse the path and make sure that we visit each vertex exactly once. This obviously can be done in polynomial time and hence the problem belongs to NP.
- Now we have to find out an NP complete problem that can be reduced to Hamiltonian path. One such closely related problem is the problem to determine whether the graph contains Hamiltonian cycle (Hamiltonian cycle is basically an Hamiltonian path that begin and end in the same vertex). Hamiltonian cycle is NP complete so we try to reduce this problem to Hamiltonian path.
- For example - Consider a graph G, from which we can construct a graph G'' such that G contains a Hamiltonian cycle if and only if G'' contains Hamiltonian path.
- For instance in Graph G the Hamiltonian cycle is A-B-D-E-C-F-A and G'' contains the Hamiltonian path V- A-B-D-E-C-F-A'-V'.
- Conversely if G'' contains Hamiltonian Path then we can convert it to the Hamiltonian cycle present in G by removing the end points V and V' and mapping A' to A.
- Thus we can show that G contains Hamiltonian cycle if and only if G'' contains Hamiltonian path which concludes the proof that Hamiltonian path is NP complete.
- As every NP complete problem is NP hard problem as well, we conclude that Hamiltonian path problem is also NP hard.



Graph G



Graph G"

Fig. 2.11.1

Review Question

1. *Explain NP hard Hamiltonian cycle problem.* **SPPU : Dec.-19, Marks 8**