B. TECH.

TCS-503

FIFTH SEMESTER EXAMINATION, 2008-09

DESIGN AND ANALYSIS OF ALGORITHMS

Time : **3** *Hours* *Total Marks :* **100**

Note : *(i) Attempt all questions.*

(ii) All parts of a question should be attempted at one contiguous place.

1. Answer any four parts of the following: **(6 × 4 = 20)**

(a) Solve the recurrence relation by iteration:

$T(n) = T(n-1) + n^4$.

Ans.

$$\begin{aligned}
T(n) &= T(n-1) + n^4 \\
T(n-1) &= T(n-2) + (n-1)^4 \\
T(n-2) &= T(n-3) + (n-2)^4 \\
T(n-3) &= T(n-4) + (n-3)^4 \\
\vdots \quad & \quad \vdots \qquad \vdots \\
T(1) &= T(0) + (1)^4 \\
T(n) &= n^4 + (n-1)^4 + (n-2)^4 + (n-3)^4 + \ldots + 3^4 + 2^4 + 1^4 \\
&< n^4 + n^4 + n^4 + \ldots + n^4 \\
&< n^4\,(1 + 1 + \ldots\ n \text{ times}) \\
&< n^4 \,.\, n \\
&< n^5 \\
T(n) &= 0\,(n^5)
\end{aligned}$$

(b) What is the smallest value of n such that an algorithm whose running time is $100\,n^2$ runs faster than an algorithm whose running time is 2^n on the same machine?

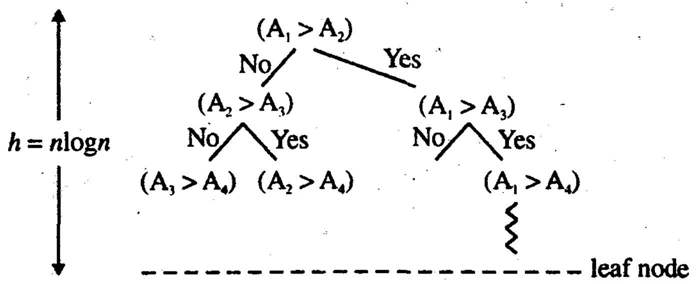
Ans.

$$\begin{aligned}
T &= 100\,n^2 \\
T^2 &= 2n \\
100\,n^2 &< 2n \\
n &= 15
\end{aligned}$$

(c) Prove that any comparison sort algorithm require $\Omega(n \log n)$ comparisons in worst case.

Ans. Let $A_1, A_2, A_3 \ldots A_n$ be n distinct number, the comparison based sort always using by comparing the element to perform sorting.

The comparison may be described by the following tree diagram.



The maximum height of the above tree is $n \log n$. So $\Omega(n \log n)$ comparison is required in worst case.

(d) Illustrate the operation of counting sort on the following array:

A = <6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2>.

Ans. A = < 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2>

	0	1	2	3	4	5	6	7	8	9	10
A =	6	0	2	0	1	3	4	6	1	3	2

	0	1	2	3	4	5	6
C =	2	2	2	2	1	0	2

	0	1	2	3	4	5	6	7	8	9	10
B =											

	0	1	2	3	4	5	6
C =	2	4	6	8	9	9	11

After I^st^ Pass

	0	1	2	3	4	5	6
C =	2	4	6	8	9	9	10

	0	1	2	3	4	5	6	7	8	9	10
B =											

After II^nd^ Pass

	0	1	2	3	4	5	6
C =	1	4	6	8	9	9	10

B =

0	1	2	3	4	5	6	7	8	9	10
	0								6	6

After IIIrd Pass

C =

0	1	2	3	4	5	6
1	4	5	8	9	9	10

B =

0	1	2	3	4	5	6	7	8	9	10
	0				2					6

The final answer

C =

0	1	2	3	4	5
0	2	4	6	8	9

B =

0	1	2	3	4	5	6	7	8	9	10
0	0	1	1	2	2	3	3	4	6	6

Sorted data

=

0	0	1	1	2	2	3	3	4	6	6

(e) How, both the minimum and the maximum of a set of *n* elements can be computed in almost 3 <*n* /2> comparisons?

Ans. At most 3<*n*/2> comparisons are sufficient to find both the min. and max.

Rather than processing each element of the impact by comparing it against the current min. and max. at a cast of 2 comparisons per element, the procession should be alone in pais of elements. We compare pairs of elements from the input first with each other, then we compare the smaller with current min. and larger with current max. at a cost of there comprise per pair.

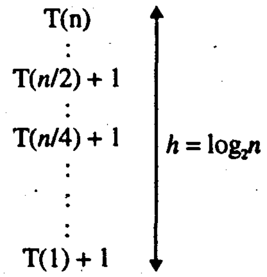
If *n* is odd then we perform 3<*n*/2> comparison. If *n* is even we perform one initial comparison followed by $3\left(\frac{n-2}{2}\right)$ comparison for a total of $\frac{3n}{2}-2$. Thus in either case total no. of comparisons. 3<*n*/2>.

(f) Solve the recurrence relation for binary search algorithms using tree method.

Ans. For binary search, the recurrence relation will be

$$\begin{aligned} T(n) &= T(n/2) + 1, \quad n \geq 2 \\ &= 1 \qquad\qquad n = 1 \end{aligned}$$

Solving using Tree method:



The height of the above tree is $\log_2 n$ and at every level the comparison required is 1. So $T(n) = 1.\log_2 n$

$$\text{So} \qquad T(n) = 1.\log_2 n$$

$$T(n) = \log_2 n.$$

2. Answer any four parts of the following: (5 × 4 = 20)

(a) Prove that a red-black tree with *n* internal nodes has height at most 2 log (*n* + 1).

Ans. $$n \geq 2^{h/2} - 1$$

$$\Rightarrow \quad n + 1 = 2h/2$$

$$\Rightarrow \quad \log(n+1) = \log 2^{h/2}$$

$$\Rightarrow \quad \log(n+1) = h/2 \log 2$$

$$\Rightarrow \quad \frac{h}{2} = \frac{\log(n+1)}{\log 2}$$

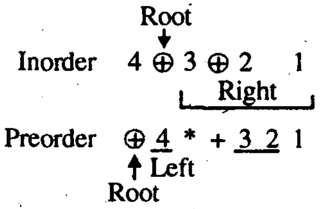
$$\Rightarrow \quad h = 2\log_2(n+1)$$

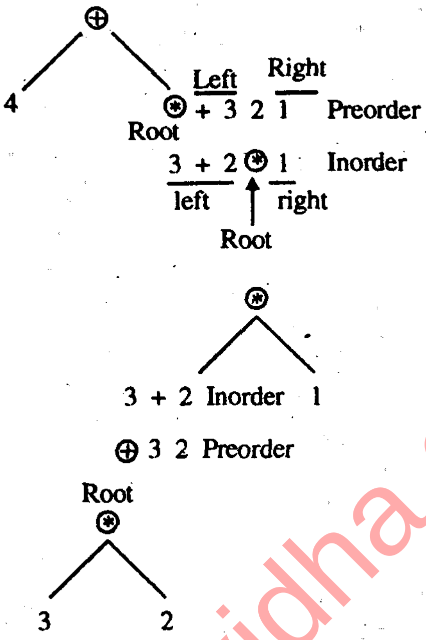
(b) Construct binary expression tree from the following traversals:

Inorder : 4 + 3 + 2 * 1

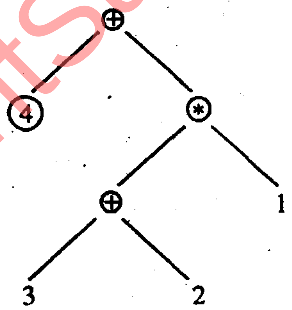
Prorater : +4 * + 3 2 1

Ans.





So, the tree will be



(c) Give an algorithm HEAP-DELETE (*A*, *i*), which deletes the item in node *i* from heap *A*.

Ans. Heap-delete (*A*, *i*, *n*)

```
1.  Exchange (A [i] ↔ A(n))
2.  Geap-size [A] ← heap-size [A] – 1
3.  Max heapify (A, i)
Max-heapify
1.  l ← left (i)
2.  r ← Right (i)
```

```
If $l \leq$ heap-size $[A]$ and $A[l] > A[i]$
    then largest $\leftarrow l$
    else largest $\leftarrow i$
If $r \leq$ heap-size $[A]$ and $A[r] > A$ [largest]
then largest $\leftarrow r$
if largest $\leftarrow i$
Then exchange $A[i] \leftrightarrow A$[ largest]
Max-heapify ($A$, largest).
```

(d) Write three operations and their implementation on dynamic sets.

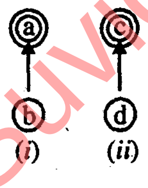
Ans. Following are the three operations supported by dynamic set.

(i) MAKE-SET (x)—Create a new set whose only member is pointed to by x. Since the sets are disjoint we require that x not already be in a set

(ii) Union (x, y) – Unit is the dynamic sets that contain x and y into new set that is union of these two sets where S_x and S_y are disjoint sets initially

(iii) Find-SET (x)—Returns a pointer to the representative of the (unique) set containing x

Implementation.



Make set for Figure (i) is $\{a, b\}$

$a = \{a, b\}$

Find set $(d) = c$

Union $(a, c) = \{a, b, c, d\}$

(e) Give an algorithm to count the number of leaf nodes in a binary tree t. What is its computing time?

Ans. int $m = 0$

Count (x) || This subordinate count for country the number of leaf node in a binary tree. x is address of the root node.

```
{
  int t = 0;
Inorder (m)
print "m";
}
int Inorder (x) || Inorder Traversal
  {
    while ($x \neq$ NULL)
    Inorder ($x \to$ left)
    if ($x \to$ left = = NULL & $x \to$ right = = NULL) || Leaf node
    $m = m + 1$
    Inorder ($x \to$ right)
}
```

Computing time is $o(n)$.

(f) Explain the process of augmenting a data structure with an example.

Ans. Augmenting a data structure can be broken into four steps.

(i) Choose an underlying data structure.

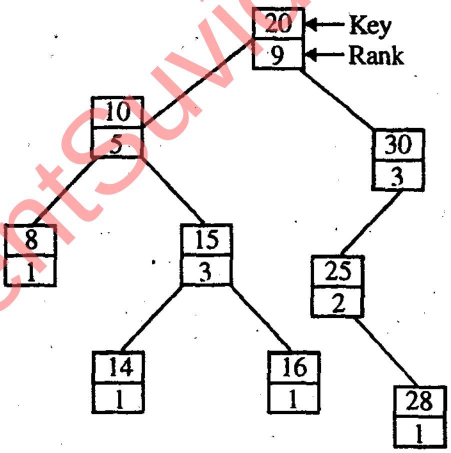
(ii) Determine the underlying data structure.

(iii) Verify that the additional information can be maintained for the basic modifying operation on underlying data structure.

(iv) Developing new operation.

For example

Red Black Tree with Rank



Size [x] = Size [LEFT(x)] + size [RIGHT (x)] + 1

Step 1. It supports Maximum, Successor, Predecessor

Step 2. Provides the size field

Step 3. Insertion and deletion still take $\log_2$ time

Step 4. We develop RANK data operation.

3. Attempt any two of the following: **(10 × 2 = 20)**

(a) What steps are used in Dynamic Programming Approach? Discuss the 0/1 Knapsack problem with respect to Dynamic Programming. Is Greedy method equally applicable for the above problem?

Ans. There are four steps in dynamic programming:

1. Characterize the structure of an optional solution
2. Recursively define the value of an optional solution
3. Compute the value of optional solution in a bottom-up fashion.
4. Constant an optional solution from computed information.

A solution to the 0/1 knapsack problem can be obtained by making a sequence of decisions on the variables $x_1, x_2, \ldots x_n$. A decision on variable x_1 involves determining which of the values 0 or 1 is to be assigned to it.

Let us assume that decisions on the x_i are made in order $x_n, x_{n-1} \ldots x_1$. Following a decision on x_n, we may be in one of two possible states: The capacity remaining in the knapsack is m and no profit has accured or the capacity remaining $m - w_n$ and a profit of p_n has accrued. It is clear that the remaining decisions $x_{n-1}, \ldots x_1$ must be optional w.r.t. the problem state resulting from the decision on x_n. Otherwise $x_n, \ldots, x_1$ will not be optional. Hence the principle of optimality holds.

Let $f_i(y)$ be the value of an optimal solution to KNAPSACK $(1, i, y)$, since the principle of optimality holds, we obtain

$$f_n(m) = \max \{f_{n-1}(m), f_{n-1}(m - w_n) + P_n\} \qquad \ldots(1)$$

For arbitrary $f_i(y)$, $i > 0$ equation (1) generalized

$$f_i(y) = \max \{f_{i-1}(y), f_{i-1}(y - w_i) + P_i\}$$

Equation (2) can be solved for $f_n(m)$ by beginning with the knowledge $f_0(y) = 0$ for all y and $f_i(y) = -\infty$, $y < 0$ Then $f_1, f_2 \ldots f_n$ can be successively computed using equation (2).

Dynamic programming approach is better for the solving the 0/1 knapsack.

(b) Apply greedy approach to an activity-selection problem of scheduling several competing activities that require exclusive use of common resources, with a goal of selecting a maximum size set of mutually comparable activities.

Ans. Greedy approach to an activity selection problem

The problem is as follows:

Job	J_1	J_2	J_3	......	J_n
Start time	S_1	S_2	S_3	......	S_n
Finish time	f_1	f_2	f_3	......	f_n

Their are n-number of jobs whose start time and finish time is given. The objective is to Maximize the number of jobs scheduled such that no two jobs intersect in start time and finish time.

Greedy approach is such a way that first job are started according to finish time in increasing order, then first job is scheduled. If next job start time not intersect with the finish time of first job and so on.

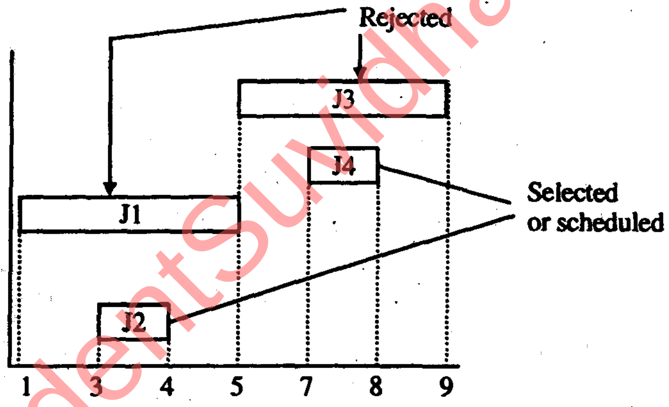
Example:

Job	J_1	J_2	J_3	J_4
Start time	1	3	5	7
Finish time	5	4	9	8

Sort the job accessding to finish time in increasing order :

Job	J_2	J_1	J_4	J_3
Start time	3	1	7	5
Finish time	4	5	8	9

J_2 is selected first, J_1 is rejected, J_4 is accepted, J_3 is rejected so scheduled job is $< J_2, J_4 >$ These can be shown by the following figure.



(c) **What do you understand by amortized analysis? What are different methods for it? Explain with an example.**

Ans. Amortized Analysis: Amortized analysis is the sequence of operations averaged over all operation by analyzing individual operations. It differs from average case analysis in that probability is not involved. It takes into account the interdependence between operation and their result.

In normal analysis technique of algorithm sometimes we overestimate the complexity but amortized analysis gives more accurate complexity.

Ex: Data structure -STACK

Operation-push, pop, multipop.

Normal Analysis:

Push of n item take – 0 (1)

Pop of n item take – 0 (1)

n-multipof of n item take – min o (n, k) o (min (n, k)) o (n)

Amortized analysis will take o (n) but normal analysis have taken o(n^2) which are overestimated. There are three methods for Amortized Analysis:

(i) Aggregate Method
(ii) Potential Method
(iii) Credit Method

Potential Method. For push, pop and multi pop of n item on the STACK Data Structure.

Amortized cost = Actual cost + change in potential

Potential is the number of item into the stack at any instant i.

So, Amortized cost = Actual cost + Δ Q

⇒ **Amortized = Actual cost + $(\phi_{i+1} - \phi_i)$**

⇒ Push of 1 item at any instant

$$\text{A.C.} = 1 + 1\,((n+1) - n) = 2$$

A.C. of push of n item = 2 n

⇒ Pop of 1 item at any instant

$$\text{A.C.} = 1 + (-1)\,(n - 1 - n)$$
$$= 0$$

A.C. of pop of n item at any instant = $0 \times n = 0$

⇒ Multipop of k item at any instant

$$\text{A.C.} = m + (-m)\,(n - m - n)$$
$$= 0$$

Amortized cost for n multipop = $0 \times n = 0$

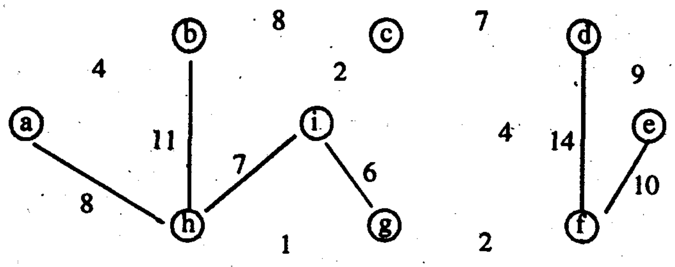
Total Amortized Cost = $2\,n + 0 + 0 = 2\,n = 0\,(n)$.

4. Answer any two parts of the following: (10 × 2 = 20)

(a) Prove that if the weights on the edge of the connected undirected graph are distinct then there is a unique Minimum Spanning Tree. Give an example in this context. Also discuss Kruskal's minimum spanning tree in detail.

Ans. If the weights on the edge are not distinct means there exist two or more edges of same weight that you can have more alternatives to select an edge in a given nondecreasing sequence. But if the weights are distinct then there is always a single selection of weighted edge in given non-decreasing sequence of weights.

For example,



In the above mentioned minimum spanning tree for a connected graph, the total weight of the tree is 37. This minimum spooning tree is not unique. Removing the edge (b, c) and replacing it with the edge (a, h) yields another spanning tree with weight 37.

Kruskal Algorithm:

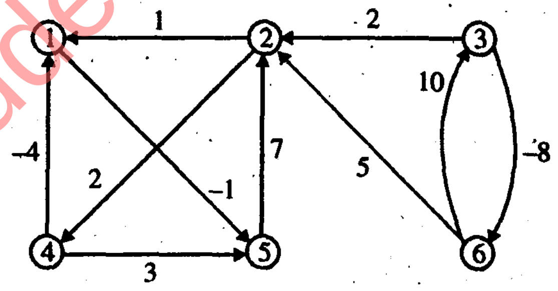
It uses a disjoint set data structure to maintain serval disjoint sets of elements. Each set contains the vertices in a tree of the current forest. The operation Find-set (u) returns a representative element from the set that contains u. Thus, we can determine whether two vertices u and v belong to the same tree by testing whether Find-set (u) equals Find-set (v). The combining of trees is accomplished by the UNION Procedure MST Kruskal (G, w)

```
A = Q
For each vevtex $v \in v[G]$
do Make-set $(v)$
Sort the edges of $E$ into non-decreasing order by weight $(w)$
For each edge $(u, v) \in E$, taken in non-decreasing order by weight
do if Find-set $(u) \neq$ Find-set $(v)$
then $A = A \cup \{(u, v)\}$
UNION $(u, v)$
return A
```

Total running time : o $(E \log E)$

(b) **Explain Floyd Warshall algorithm with suitable example.**

(c) **For the graph (weighted, directed),**



apply Floyd-Warshall algorithm for constructing shortest path. Show the matrix $D^{(K)}$ that results in each iteration.

Ans. $$D^0 = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & \infty/2 & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & 3/-5 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{bmatrix} \qquad D^1 = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & 8 \\ \infty & 2 & 0 & \infty/4 & \infty/2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ \infty & 7 & \infty & \infty/9 & 0 & \infty/15 \\ \infty & 5 & 10 & \infty/7 & \infty/5 & 0 \end{bmatrix}$$

$$D^3 = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1/-2 & 0 & \infty & 2 & 0/-3 & -8 \\ \infty/0 & 2 & 0 & 4 & 2/-1 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ \infty/5 & 7 & \infty & 9 & 0 & 15 \\ \infty/3 & 5 & 10 & 7 & 5/2 & 0 \end{bmatrix} \qquad D^4 = \begin{bmatrix} 0 & \infty/6 & \infty & \infty/8 & -1 & \infty/14 \\ -2 & 0 & \infty & 2 & -3 & -8 \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & \infty/2 & \infty & 0 & -5 & \infty/10 \\ 5 & 7 & \infty & 9 & 0 & 15 \\ 3 & 5 & 10 & 7 & 2 & 0 \end{bmatrix}$$

$$D^5 = \begin{bmatrix} 0 & 6 & \infty/24 & 8 & -1 & 14 \\ -2/5 & 0 & \infty/2 & 2/-1 & -3 & -8 \\ 0/-5 & 2 & 0 & 4/-1 & -1 & -8 \\ -4 & 2 & \infty/20 & 0 & -5 & 10 \\ 5 & 7 & \infty/25 & 9 & 0 & 15 \\ 3 & 5 & 10 & 7 & 2 & 0 \end{bmatrix}$$

$$D^3 = \begin{array}{c|cc|c|ccc|} & 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 0 & \infty & \infty & \infty & -1 & \infty \\ 2 & 1 & 0 & \infty & 2 & 0 & \infty \\ \hline 3 & 3 & 2 & 0 & \infty & 2 & \infty \\ \hline 4 & -1 & \infty & \infty & 0 & -5 & \infty \\ 5 & 8 & 7 & \infty & \infty & 0 & \infty \\ 6 & 6 & 5 & 10 & 7 & 5 & 0 \end{array} \qquad D^4 = \begin{array}{c|cccccc|} & 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 0 & \infty & \infty & \infty & -1 & \infty \\ 2 & -2 & 0 & \infty & 2 & 0 & \\ 3 & & & 0 & \infty & & \\ 4 & -1 & \infty & \infty & 0 & -5 & \infty \\ 5 & & & & \infty & 0 & \\ 6 & & & & 7 & & 0 \end{array}$$

$$D^6 = \begin{bmatrix} 0 & 6 & 24 & 8 & -1 & 14 \\ -5 & 0 & 2 & -1 & -3 & -8 \\ -5 & 2 & 0 & -1 & -1 & -8 \\ -4 & 2 & 20 & 0 & -5 & 10 \\ 5 & 7 & 25 & 9 & 0 & 15 \\ 3 & 5 & 10 & 7 & 2 & 0 \end{bmatrix}$$

$$D^2 = \left[\begin{array}{cc|c|ccc} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & -8 \\ \hline \infty & 2 & 0 & 4 & 2 & -8 \\ \hline -4 & \infty & \infty & 0 & -5 & \infty \\ \infty & 7 & \infty & 9 & 0 & 15 \\ \infty & 5 & 10 & 7 & 5 & 0 \end{array}\right]$$

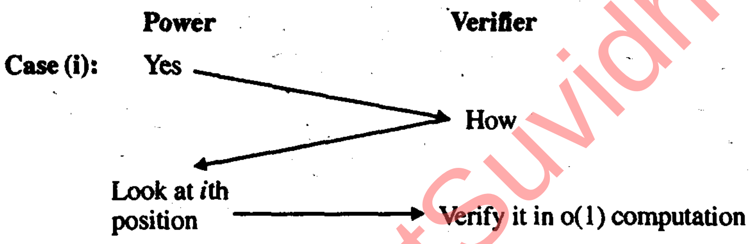
5. Answer any two of the following:

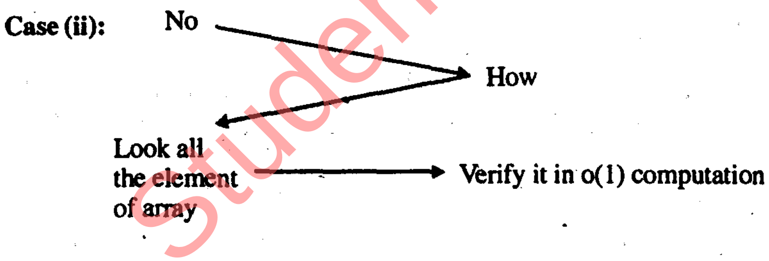
(a) (i) Discuss the decision problems of class P and NP.

(ii) Discuss NP complete problems.

Ans. (i) Decision Problem of class P: For decision problem of any class we have prover and verifier. Prover are all powerful but verifier are less powerful. Here powerful means computing time in polynomial order. In class P, any problem of decision will either be yes or no. In both case the verifier can compute it in polynomial time.

Example: Searching an element in an array problem. Is x is in A?





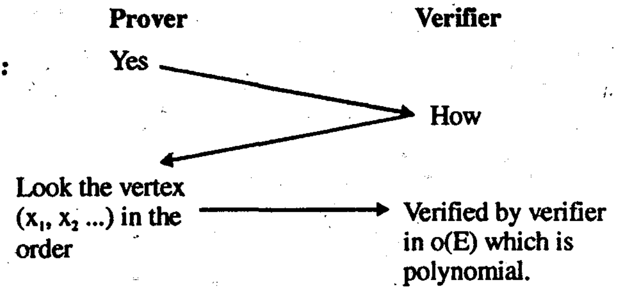
So in the above example the verifier can verify it in polynomial time.

Decision Problem of class NP: In this case the verifier can verify only for yes from prover but it could not verify for NO from NO from prover in polynomial computation.

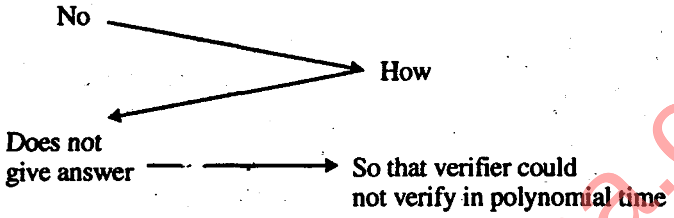
Example: Hamiltonian circuit problem.

Question: In Group G is their Hamiltonian Path?

Case (i):


Prover
Verifier
Yes
How
Look the vertex (x_1, x_2 ...) in the order
Verified by verifier in o(E) which is polynomial.

Case (ii):



The above problem is NP because the verifier cannot verify for no answer in polynomial time.

(ii) NP-Complete Problem: Any NP Complete Problem must be satisfied following two conditions.

(A) The Problem is in NP class

(B) The Problem in NO-Hard.

We know that Hamiltonian path problem is NP class.

There is well known theorem by Cooks that the satisfiability (SAT) problem is NP-Hard. It means that if this problem could be solved by the polynomial time then he could solve all the other NP complex problem in polynomial time.

To show that the problem is NP Hard, we will reduce the problem into SAT problems in polynomial time and we know that SAT is NP Hard. So we can say that the problem reduced to SAT is NP-Hard.

(b) Explain Boyer-Moore algorithm for string matching for text:

“*a b c a a b c c a a b b a b c a*” Pattern *a b c*.

Compute worst time complexity of this algorithm.

Ans. BM Match (T, P)

```
Compute function last
i ← m – 1
j ← m – 1
repeat
if P [j] = T [i] then
   if j = 0 then
```

```
        return i // a match!
    else
        i ← i − 1
        j ← j − 1
else
    i ← i + m − min (j , 1 + last (T [i] ) )
                                    // Jump step
    j ← m − 1
until i > n − 1
return " There is no substring of
        T matching P"
```

(c) **What are approximation algorithms? What is meant by a $P(n)$ approximation algorithm? Give an approximation algorithm for travelling salesman problem.**

Ans. Approximation Algorithm: Approx Algorithm is always used for the problem of NP-Complete. We know that the NP-Complete problem cannot be computed exactly in polynomial time. Here exactly means optimum. There are a lot of real world problem that must be solved which is NP-Complete. Computing NP-Complete is not possible by computer for large number of input. So we come to solution of nearly optimum solution which uses the approximation algorithm. The complicity of Approx Algorithm is always polynomial.

$$\text{Approx Ratio} = \frac{\text{cost of Approx Algo}}{\text{cost of Optimal Algo}}$$

Example. 2- Appox. Algorithm. Means that the cost of approx also is 2 times the cost of optimal algorithm.

P(n)- Approx algorithm. It means its completely is p(n) which is a polynomial.

Travelling Salesman Problem : (2-Aprox Algo)

Approx-TSP-TOUR (G, C)

1. Select a vertex $r \in V[G]$ to be a root vertex
2. Compute a MST T for G from root r using MST - PRIM (G, c, r)
3. Let L be the list of vertices visited in preorder tree walk of T
4. Return the Hamiltorian cycle H that visit the vertices in the order L.

Floyd-Warshall Algo:

FLOYD - WARSHALL (W)

```
$n$ ← rows ($W$)
$D^{(0)}$ ← $W$
for $k$ ← 1 to $n$
    do for $i$ ← 1 to $n$
        do for $j$ ← 1 to $n$
            do $dij^{(k)} \leftarrow \min \{dij^{(k-1)}, dik^{(k-1)} + dk_i^{(k-1)}\}$
return $D$
```