Date:
Page No:

Q.1. (i)

Given: If a decision problem in P is NP-Complete

To prove: $P = NP$

let $A \in P$ & A is NP-complete

Since A is NP-Complete

$A \in NP$

$\forall L \in NP \quad L \leq_p A$

All problems of NP are reducible to A which can be solved in polynomial time. Therefore all problems of NP can be solved in polynomial time.

Therefore $P = NP$.

proved

Date:

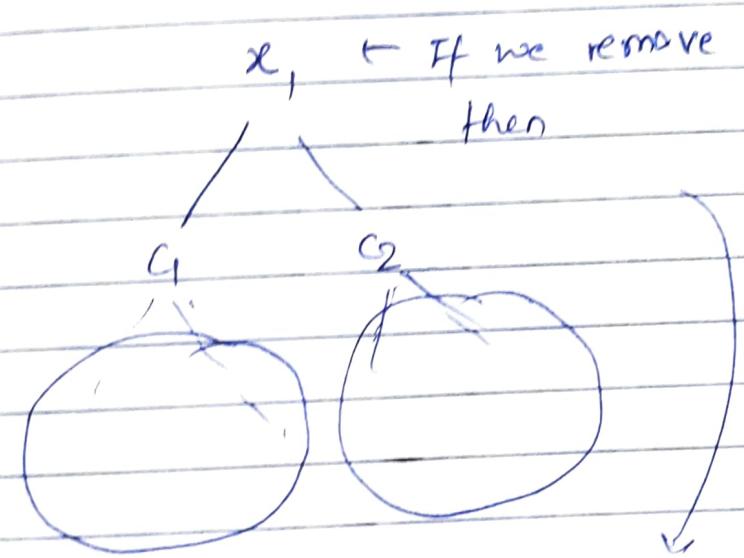
Page No.

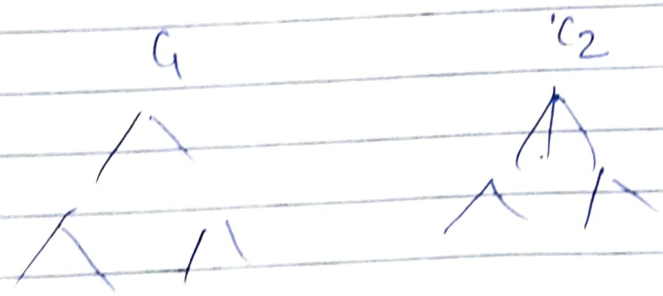
(ii) Given G – connected undirected graph

True

If running DFS using x as root gives us 2 child c_1 & c_2

Then, the subtree with c_1 & c_2 as root will become disconnected if we remove x.





we get 2 disconnected graphs. Therefore x is not a safe vertex.

Date:

Page No:

Q.1.

Ans. Given: $f(n) = O(g(n))$

To prove: $2^{f(n)} = 2^{O(g(n))}$ or not

let $f(n) = 2\log_2 n$

& $g(n) = \log_2 n$

Now $f(n) \leq c\,g(n)$ for $c \geq 2$ & $n \geq 1$

$2\log_2 n \leq c\log_2 n$

So $f(n) = O(g(n))$

Now

$2^{f(n)}$ $\quad$ 2^{g}

$\Rightarrow 2^{2\log_2 n}$ $\quad$ $\Rightarrow 2^{\log_2 n^2}$

$\Rightarrow n^{2\log_2 2} \Rightarrow n^2$

$2^{f(n)} = n^2$

Now $2^{g(n)} = 2^{\log_2 n} = n^{\log_2 2} = n.$

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

Date:

Page No:

We know that n^2 is not $O(n)$

So $2^{f(n)} \neq 2^{O(g(n))}$.

Date:
Page No:

Q.2.
Ans

S - n+1 distinct numbers.

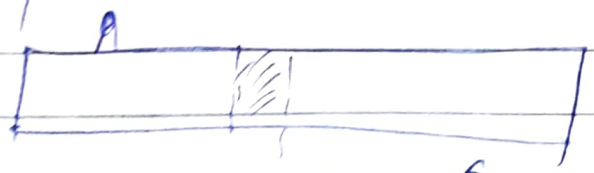
A → array of n numbers.

One number is missing in A.

- We will use the median to partition the array into subparts.
We know that median divides into equal parts.

We find the median of A in O(n) time as discussed in the class.
let say this median is m.
Now, the numbers which are less than m are called A_L & numbers which are more than m is called A_R.

Similarily the numbers which are less than m in array S is called S_L & S_R respectively for more than m.

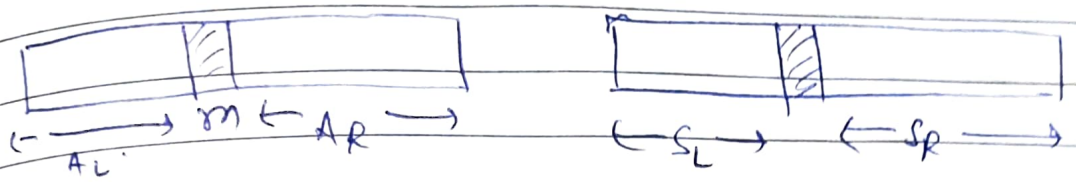


Consider this as sorted A in increasing order

Consider this as sorted S in increasing order

Date:

Page No:



Now $|A_L| + |A_R| + |m| = n$ — (i) → number of elements in A

$|S_L| + |S_R| + m = n+1$ — (ii) - " in S

from eq (i) & (ii)

$|S_L| + |S_R| + \cancel{m} = |A_L| + |A_R| + \cancel{m} + 1$

$|S_L| + |S_R| = |A_L| + |A_R| + 1$ — (iii)

Case 1:- Now if $|S_L| = |A_L|$

then in eq (iii)

$\cancel{|S_L|} + |S_R| = \cancel{|A_L|} + |A_R| + 1$

$|S_R| = 1 = |A_R|$ we need to search in other half which is more than m.

Case 2: if $|S_R| = |A_R|$

then in eq (iii)

$|S_L| + \cancel{|S_R|} = |A_L| + \cancel{|A_R|} + 1$

Date:
Page No:

$|S_L| = 1 = |A_L|$

Then search in the first half or which elements are less than m.

Depending on the cases above mentioned we either move to 1 or 2.

So our Recurrence Relation is : $T(n) = T(n/2) + O(n)$ (iv)

one half of array S.

median

So $T(n) = O(n)$ as discussed in the class from (iv).

Q3 (a) Given $A = [1 \ldots n] =$ "DYNAMIC" (eg)

$B = [1 \ldots m]$ "PROGRAMMING"

$C = [1 \ldots m+n] =$ "PRODGYRNAMAMMIINGC"

O/P → True or false if C is shuffle of A & B or not

We define $T[i,j]$ = True if $C_1 \ldots C_{i+j}$ is a shuffle of $A_1 \ldots A_i$ & $B_1 \ldots B_j$.

We will make a table of ~~dep~~ $T[n][m]$ in this case.

$$T[i,j] = \begin{cases} \text{True} & \text{if } i=0 \text{ and } j=0 \\ (T[0,j-1]) \,\&\, (B[j]=C[j]) & \text{if } i=0 \,\&\, j>0 \\ (T[i-1,0]) \,\&\, (A[i]=C[i]) & \text{if } i>0 \,\&\, j=0 \\ (T[i-1,j] \,\&\, (A[i]=C[i+j])) \text{ OR } (T[i,j-1] \,\&\, (B[j]=C[i+j])) & \text{if } i>0 \,\&\, j>0 \end{cases}$$

1 → If both A & B & C are empty string

→ 2 If A is empty then every character of B should match with every corresponding character in C

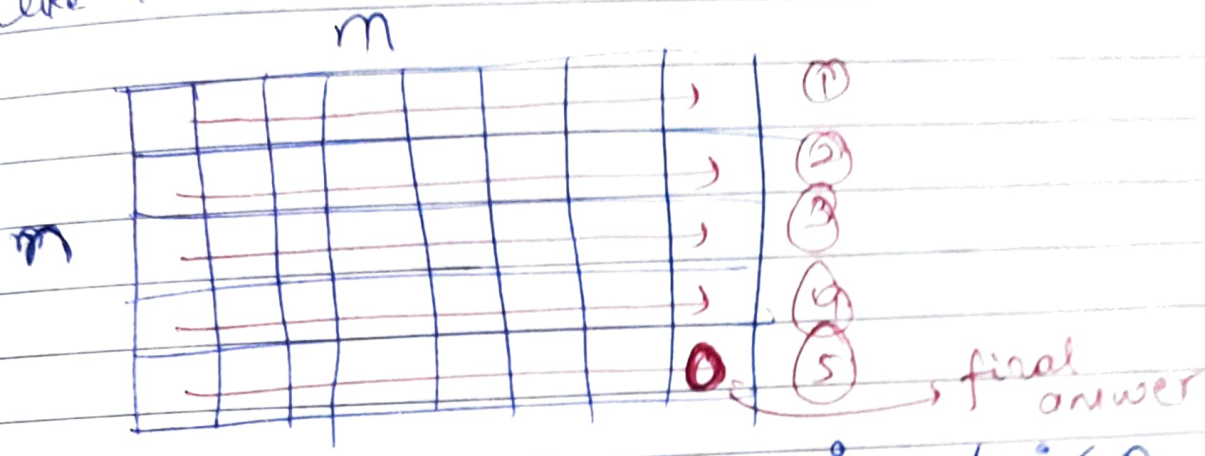
3 If B is empty, then every character of A should match with every corresponding character in C.

4 If C^{th} $i+j$ characters matches with either A^{th} i character or B^{th} j character.

· We know each $T[i,j]$ entry depends on $T[i-1][j]$ OR $T[i][j-1]$.

Q. 3(a):

Now we fill the table T in Row major order like this



Initialize all T[i,j] = False for all $i \le m$ & $j \le n$.

```
for (i=0 to m)
{ for (j=0 to m)
  {if (i==0 AND j==0)
     { T[i][j] = True. }
  else if ((i==0 AND j>0) AND (B[j] == C[j]))
     { T[i][j] = T[i][j-1] }
  else if ((i>0 & j==0) AND (A[i] == C[?]))
     { T[i][j] = T[i-1][j] }
```

```
else
{ if (A[i] == C[i+j])
        T[i][j] = T[i-1][j]

  elif (B[j] == C[i+j])
        T[i][j] = T[i][j-1]

    }
  }
}
```

The value T[m][n] is True then shuffle is possible & if false shuffle is not possible.

Time Required = $O(mn)$
Space Required = $O(mn)$

Date:
Page No:

Q.4.

Ans.

Given:

Town has f families.
k clubs.

i^{th} club needs a_i members.
but can take atmost 3 members from any family.
Each person belongs to atmost 1 club.
j^{th} family has b_j members.

We make the following Network flow G.

(i) we have source node s and sink t.

(ii) for every club c_i, we have c_i node in G.

(iii) for every family f_i, we have f_i node in G.

(iv) Connect s to club node c_i with edge capacity c_i.

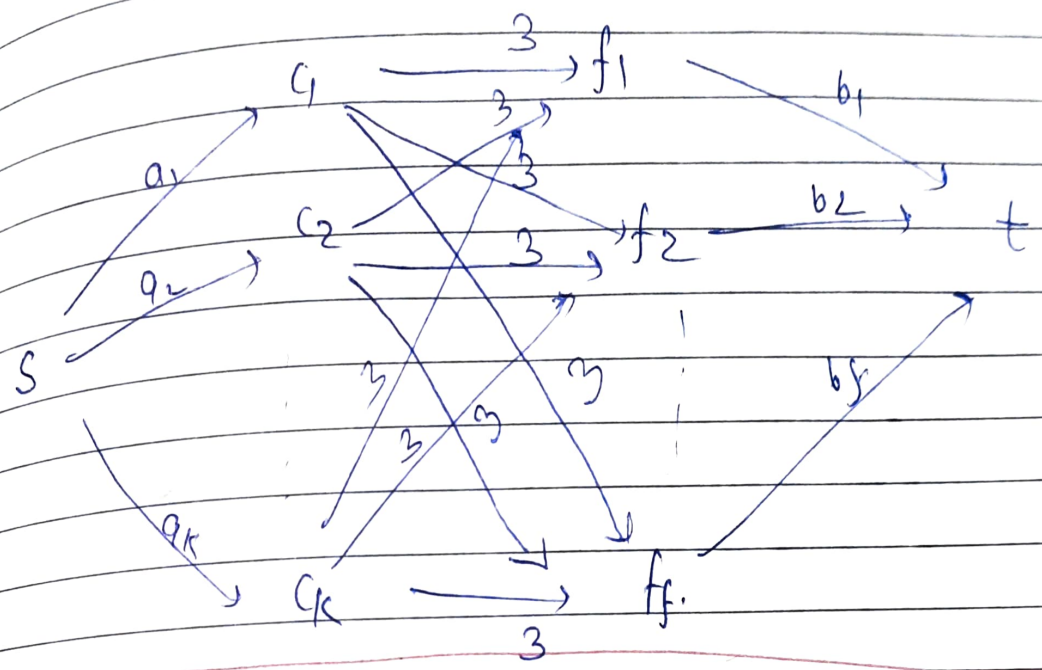
(v) connect family node f_j to t with edge capacity b_j.

Date:

Page No:

(vi) Connect all club nodes to all family nodes with edge capacity 3.

So G looks like this..



If the max flow in G is $f \geq \sum_{i=1}^{k} a_i$, then it means assignment of people to clubs is possible.

① A flow f in edge (s, c_i) will mean c_i has f members.

~~flow can be f~~

Date:
Page No:

② A flow f in edge (c_i, f_j) will mean f member of family f_j are assigned to c_i.

③ A flow f in edge (f_j, t) means total f members of family f_j are assigned to clubs.

Flow can be found through Ford-Fulkerson algorithm.

Time Required $= O(mn)$.

Q.5.
Ans.

Given Graph $G = (V, E)$ $K > 0$
need to find clique of size K

let $|V| = n$.

Number of ways to choose K vertices from $|V| = {}^nC_K = m$ (let say)

There are m ways to choose K vertices.

let $\{S_1, S_2 \ldots S_m\}$ be all the subsets of K vertices of V.

We have to reduce clique to near clique problem.

We will create a new graph G'
$G' = (V, E')$. It contains all original vertices and edges of G

Then, for every S_i, add a vertex x_i in G' and connect all vertices of set S_i to x_i except 1.

$G = (V, E) \longrightarrow G'(V \cup \{x_1 \dots x_m\}, E \cup E')$

Clique of size K

E' → set of all edges between x_i and v_j.

near clique of size $K+1$.

If G' has near clique of size $K+1$, then G has a clique of size K.

Proof:

Suppose S' is a near clique in G'.
If $x_i \in S'$, then $\{S_i \cup x_i\}$ vertices will every possible edge between them except 1 connecting x_i to some v_j.

All vertices of S_i will have edges between any 2 of them, hence S_i is a clique of G.

If G has clique of size k, then G' will have near clique of size $k+1$.

let S_i be clique of G with size k, then when we add x_i to all vertices of S_i except 1, then $S_i \cup x_i$ becomes a near clique of size $k+1$ in G'.

G' can be made from G by creating $({}^{n}C_{k})$ vertex and connecting each of those vertex to a unique set of k vertices in polynomial time.

Hence, clique problem is reducible to near clique problem.

Date:
Page No:

Q6.
Ans: **Given** X & Y n positive integer.

Goal: $\prod_{i=1}^{n} x_i y_i = x_1 y_i$ is maximized.

Greedy approach:

1. We sort the X values in decreasing order
   let this ordering be $x_1 x_2 \ldots x_n$.

2. we sort the Y values in decreasing order
   let this ordering be $y_1 y_2 \ldots y_n$.

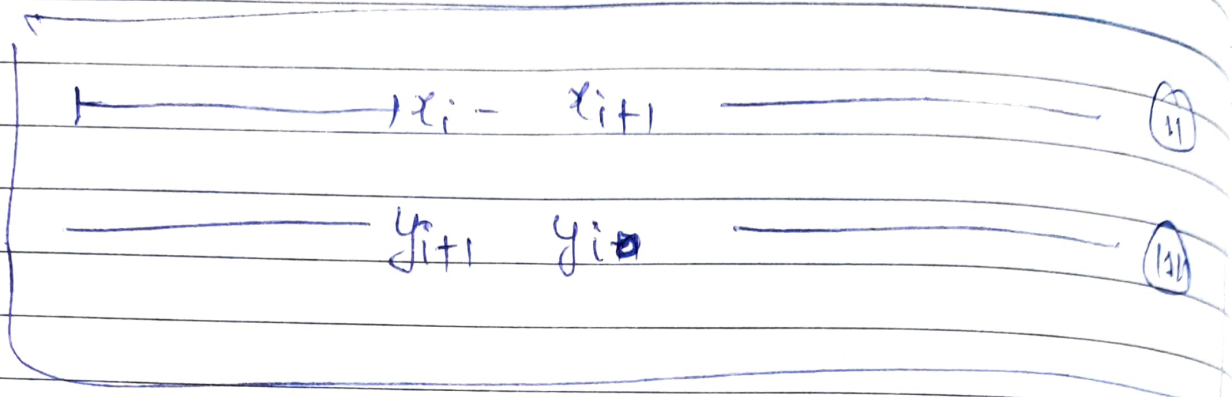
So, Now taking (x_1, y_1) (x_2, y_2) ... $(x_n y_n)$ will result in maximize product.

We can assume that optimal solution orders in $x_1 \ldots x_n$.
If optimal solutions also orders $y_1 \ldots y_n$ then we are done.
Else there must be two consecutive elements such that $y_i \leq y_{i+1}$. ~~and remain~~

We can swap these two y_i & y_{i+1} then we will get much better optimal solution keeping rest of the element as it is.

Date:
Page No:

Now $y_i \le y_{i+1}$ (1)



In the Diagram $x_i \ge x_{i+1}$ and from (1)
$y_i \le y_{i+1}$

Now initial product
is $\Rightarrow x_i^{y_{i+1}} * x_{i+1}^{y_i}$

after swapping

$\Rightarrow x_i^{y_i} * x_{i+1}^{y_{i+1}}$

$y_i \le y_{i+1}$ from (1)

(4) $\rightarrow x_i^{y_i} \le x_{i+1}^{y_{i+1}}$ Raising power to same element will remain as it as both powers are +ve.

Similarily.

$$x_{i+1} y_i \leq x_{i+1} y_{i+1} \quad -(5)$$

Now From (4) & (5)

$$x_i (y_{i+1} - y_i) \geq x_{i+1} (y_{i+1} - y_i)$$

Thus swapping only result in better optimal solution.

Similarily we will swap & reduce the number of inversions in the solution.

Time Required $= O(n \log n)$.

sort both the arrays.

Date:

Page No:

Q.3.6)

This can be solved using minor modification to the solution of 3(a).

We will make a 2D array $T[n][m]$ same as first problem.

$$T[i][j] = \begin{cases} \text{TRUE if } \Big((T[i-1][j]) \text{ AND } (A[i] = C[i+j]) \text{ AND } (A[i-1] \neq C[i+j-1] \text{ OR } A[i-2] \neq C[i+j-2]) \Big) \\ \text{TRUE if } \Big((T[i][j-1]) \text{ AND } (B[j] = C[i+j]) \text{ AND } (B[j-1] \neq C[i+j-1] \text{ OR } B[j-2] \neq C[i+j-2]) \Big) \\ \text{false otherwise} \end{cases}$$

Date:
Page No:

If $C[1 \ldots i+j-2]$ is a shuffle of $A[1 \ldots i-2], B[1 \ldots j]$

and also if

$C[1 \ldots i+j-1]$ is a shuffle of $A[1 \ldots i-1], B[1 \ldots j]$

then

$C[1 \ldots i+j]$ can't be smooth shuffle of $A[1 \ldots i], B[1 \ldots j]$

because then last 3 elements of $C[1 \ldots i+j]$ will be same as $A[1 \ldots i]$

Similarity in case of B.

Time Complexity of algorithm.

= each element of 2D array takes $O(1)$ to compute

So, time complexity $O(nm)$.