Total No. of Pages
IVth SEMESTER
MID SEMESTER EXAMINATION

Roll No. .....
B.Tech.(Computer Engg.)
(March 2018)

Paper Code: COE-208 Title of the subject: Algorithm Design and Analysis
Time: 1:30 Hours Max. Marks: 25

Note: Answer all questions. Write pseudo code for all algorithms asked.
Assume suitable missing data, if any.

Total No. of Pages : 2
IV TH SEMESTER
MID SEMESTER EXAMINATION
CO/IT 208 ALGORITHM DESIGN AND ANALYSIS
Time: 1:30 Hours
Note: Answer all questions
Assume suitable missing data, if any

Roll No. .....
B.TECH. (CO/IT)
(March - 2017)
Max. Marks: 25

Total No. of Pages : 1
IVth SEMESTER
MID SEMESTER EXAMINATION

Roll No. .....
B.Tech.(Computer Engg.)
(March 2020)

Paper Code: CO-208 Title of the subject: Algorithm Design and Analysis
Time: 1:30 Hours Max. Marks: 25

Note: Answer all questions. Write pseudo code for all algorithms asked.
Assume suitable missing data, if any.

Total No. of Pages:3
IVth SEMESTER
END SEMESTER EXAMINATION

Roll No .....
B.Tech.(Computer Engg.)
(May 2018)

Paper Code: COE-208 Title of the subject: Algorithm Design and Analysis
Time: 3:00 Hours Max. Marks: 50

Note: Answer 5 questions. Write pseudo code for all algorithms asked.
Assume suitable missing data, if any.

Total No. of Pages:03
4 TH SEMESTER
END SEM EXAM
CO/IT-208 ALGORITHMS DESIGN AND ANALYSIS
Time: 3 Hours

Roll No. .....
[B.TECH. – CO/IT]
(MAY- 2017)
Max. Marks: 50

Note: Attempt any 5 Questions Each Question Carry Equal Marks.
Assume suitable missing data, if any.

Total No. of Pages: 02
IV[th] SEMESTER
END SEMESTER EXAMINATION
CO208 ALGORITHM DESIGN AND ANALYSIS
Time: 3:00 Hours

Roll No. .....
B.Tech. (CSE)
SEPT-2019
Max. Marks: 50

Note: Attempt any 5 questions. All questions carry equal marks.
Assume suitable missing data, if any.

Total No. of Pages02
IV[th] SEMESTER
SUPPLIMENTRY EXAMINATION
CO-214 ALGORITHM DESIGN AND ANALYSIS
(Old Scheme)
Time: 3:00 Hours

Roll No. .....
B.Tech. (CO)
AUG-2018
Max. Marks: 70

Note: Attempt any 5 questions. All questions carry equal marks.
Assume suitable missing data, if any.

Total No. of Pages : 3
4[TH] TH SEMESTER
END SEM EXAM
CO/SE-214 ALGORITHMS DESIGN AND ANALYSIS
Time: 3 Hours

Roll No. 045
[B.TECH. – CO/SE]
(MAY- 2016)
Max. Marks: 70

Note: Attempt any 5 Questions.
Each Question Carry Equal Marks.
Assume suitable missing data, if any.

Total No. of Pages:4
IVth SEMESTER
SUPPLIMENTRY EXAMINATION

Roll No. .....
B.Tech.(Computer Engg.)
(May- 2019)

Paper Code: COE-208 Title of the subject: Algorithm Design and Analysis
Time: 3:00 Hours Max. Marks: 50

Note: Answer 5 questions. Write pseudo code for all algorithms asked.
Assume suitable missing data, if any.

Total No. of Pages02
IV[th] SEMESTER
END SEMESTER EXAMINATION
CO-214 ALGORITHM DESIGN AND ANALYSIS
SE
(Old Scheme)
Time: 3:00 Hours

Roll No. .....
B.Tech. (CO)
MAY-2018
Max. Marks: 70

Note: Attempt any 5 questions. All questions carry equal marks.
Assume suitable missing data, if any.

Total No. of Pages 2
FOURTH SEMESTER
END SEMESTER EXAMINATION
CO/SW-214 ALGORITHM DESIGN & ANALYSIS
Time: 3:00 Hours

Roll No. .....
B.Tech. [CO/SW]
MAY-2012
Max. Marks: 70

Note: Question No. ONE is compulsory.
Answer any FOUR questions from the rest.
Assume suitable missing data, if any.

Total No. of Pages:
IVth SEMESTER
SUPPLIMENTRY EXAMINATION

Roll No. .....
B.Tech.(Computer Engg.)
(AUG- 2018)

Paper Code: COE-208 Title of the subject: Algorithm Design and Analysis
Time: 3:00 Hours Max. Marks: 50

Note: Answer 5 questions. Write pseudo code for all algorithms asked.
Assume suitable missing data, if any.

2[a] What are Asymptotic notations? Define Big-Oh and Big-Omega notations. 2012E 3

a) Explain the following terms: (2x3=6)

(i) Big-oh 2018S
(ii) Big-omega 2016E
(iii) Big-theta
(iv) Big-oh
(v) Big-omega 2018E
(vi) Big-theta

b) Explain Big O, big Ω, big θ asymptotic notations. 2017E

Q6. Write short notes on
(i) Big-oh
(II) Big-omega 2019E
(III) Big-theta
(iv) RAM Model

UNIT-1
(INTRODUCTION)

Asymptotic notations are used to analyse algorithms to describe the growth rate of the running time or space usage of an algorithm as the input size grows to infinity. They are used to worst-case running TC of an algorithm.

1. **Big Oh (O)**: This notation is used to describe the upper bound on the running time of an algorithm.
   ∴ If $f(n) = O(g(n))$
   then ∃ $c>0, n_0>0$ $0 \le f(n) \le cg(n)$ for all $n \ge n_0$

2. **Big Omega (Ω)** " " lower "
   ∴ If $f(n) = \Omega(g(n))$
   then there exist $c>0, n_0>0$ and $f(n) \ge c \cdot g(n)) \ \forall \ n \ge n_0$

3. **Big Theta (Θ)**: " " both the upper and lower bounds
   ∴ If $f(n) = \theta(g(n))$
   then for $c_1>0; c_2>0; n_0 \ge 0 \Rightarrow c_1 g(n) \le f(n) \le c_2 g(n) \ \forall \ n \ge n_0$

Q1 (a) Solve following recurrences: 2018S
i. $T(n) = 3T(n/4) + n \log n$
ii. $T(n) = 2^n T(n/2) + n^n$
iii. $T(n) = 64T(n/8) + n^2 \log n$

Q1. (a) For master theo
(i) T(
(ii) T(
Q1
a) For theor
(i) T
(ii) T
(iii) T
(iv) T

Date / /
Page No.

MASTER'S THEOREM / METHOD

$T(n) = aT(n/b) + f(n)$ $\quad a \geq 1,\ b > 1,\ f(n) > 1$

$h(n)$	$U(n)$
n^r ; $r > 0$	$O(n^r)$
n^r ; $r < 0$	$O(1)$
$(\log n)^i$; $i \geq 0$	$\frac{(\log_2 n)^{i+1}}{i+1}$

Q) Solve the following:

a) Apply master method to find the complexity of the following: 2017 M [2 x 2 = 4]

METHOD (1)

i) $T(n) = 4T(n/2) + n^3$

ii) $T(n) = 4T(n/2) + n^2$

i) $T(n) = 4T(n/2) + n^3$

we have, $a = 4,\ b = 2,\ f(n) = n^3$

by master's theorem,

$T(n) = n^{\log_b a} \cdot U(n) = n^{\log_2 4} \cdot U(n) = n^2\, U(n)$

now, $h(n) = \frac{f(n)}{n^{\log_b a}} = \frac{n^3}{n^2} = n \Rightarrow U(n) = (n)$

$\therefore T(n) = n^2 \cdot n = n^3 \Rightarrow \boxed{O(n^3)}$

ii) $T(n) = 4T(n/2) + n^2$

we have, $a = 4,\ b = 2,\ f(n) = n^2$

by master's theorem,

$T(n) = n^{\log_b a} \cdot U(n) = n^{\log_2 4}\, U(n) = n^2\, U(n)$

now, $h(n) = \frac{f(n)}{n^{\log_b a}} = \frac{n^2}{n^2} = 1 = (\log n)^0 \Rightarrow U(n) = \frac{(\log_2 n)^{0+1}}{0+1}$

$\therefore T(n) = n^2 \cdot \log_2 n \Rightarrow \boxed{O(n^2 \log n)}$

OTHER WAY OF MASTER'S THEOREM

$T(n) = aT(n/b) + f(n)$ and $f(n) = O(n^k \log^p n)$

CASE 1: if $a > b^k$ then $O(n^{\log_b a})$

CASE 2: if $a = b^k$ if $p > -1$ then $O(n^k \log^{p+1} n)$
if $p = -1$ then $O(n^k \log(\log n))$
if $p < -1$ then $O(n^k)$

CASE 3: if $a < b^k$ if $p \geq 0$ then $O(n^k \log^p n)$
if $p < 0$ then $O(n^k)$

1. Solve following recurrences using Master's Method 2018M

(i) $T(n) = 3T(n/3) + n/2$

(ii) $T(n) = 7T(n/3) + n^2$

(iii) $T(n) = 2T(n/2) + n \log n$ (2+2+2=6)

Date / /
Page No.

i) $T(n) = 3T(n/3) + n/2$

we have, $a=3$, $b=3$, $f(n) = n/2$

by master's theorem $n^{\log_b a} = n^{\log_3 3} = n$

$T(n) = n^{\log_3 3} \cdot U(n)$

now, $h(n) = \frac{f(n)}{n^{\log_3 3}} = \frac{n/2}{n} = \frac{1}{2} (\log n)^0 \Rightarrow U(n) = \frac{(\log_2 n)^{0+1}}{0+1} \cdot \frac{1}{2}$

$\therefore T(n) = n \times \frac{\log_2 n}{2} \Rightarrow \boxed{O(n \log n)}$ CASE 2

ii) $T(n) = 7(T(n/3)) + n^2$

we have, $a=7$, $b=3$, $f(n) = n^2$

by MT,

$T(n) = n^{\log_3 7} \cdot U(n)$

now, $h(n) = \frac{n^2}{n^{\log_3 7}} = n^{2-1.77} = n^{0.23} \Rightarrow U(n) = O(1)$

$T(n) = n^{\log_3 7} \Rightarrow \boxed{O(n^2)}$ CASE 3

$\therefore n^{1.77} \leq c \cdot n^2$

iii) $T(n) = 2T(n/2) + n \log n$

we have, $a=2$, $b=2$, $f(n) = n \log n$

by MT,

$T(n) = n^{\log_2 2} U(n) = n \cdot U(n)$

now, $h(n) = \frac{f(n)}{n^1} = \frac{n \log n}{n} = \log n = (\log n)^1 \Rightarrow U(n) = \frac{(\log_2 n)^{1+1}}{1+1}$

$\therefore T(n) = \frac{n \times (\log n)^2}{2} \Rightarrow \boxed{O(n \log^2 n)}$ CASE 2

METHOD 2 DIFF WAY

Q2

a) For each of the following recurrence, solve them with the help of master theorem? (2X2=4) 2017E

(i) $T(n) = 0.5T\left(\frac{n}{2}\right) + n^2$

(ii) $T(n) = \sqrt{2}T\left(\frac{n}{2}\right) + \log n$

i) $T(n) = 0.5\,T(n/2) + n^2$

$a = 0.5 \mid b = 2 \mid k = 2 \mid p = 0$

$a < b^k \rightarrow$ CASE 3

$p \geq 0$

$T(n) = O(n^k \log^p n) = O(n^2)$

ii) $T(n) = \sqrt{2}\,T(n/2) + \log n$

$a = \sqrt{2} \mid b = 2 \mid k = 0 \mid p = 1$

$a = \sqrt{2} \mid b^n = 0 \Rightarrow a > b^k \rightarrow$ CASE 1.

$\therefore$

$T(n) = O(n^{(\log_b a)})$

$= O(n^{(\log_2 \sqrt{2})})$

$= O(n^{0.5})$

$= O(\sqrt{n})$

(ii) $T(n) = 6T(n/3) + n^2 \log n$ using Master's Method

$T(n) = 6T(n/3) + n^2 \log n$

we have, $a = 6 \mid b = 3 \mid f(n) = n^2 \log n \mid k = 2 \mid p = 1$

by MT,

$6 = a < b^k = 9 \rightarrow$ CASE 3

$2 = p \geq 0$

$\therefore T(n) = O(n^k \log^p n) = O(n^2 \log n)$

Q1. (a) For each of the following recurrence sole them with the help of master theorem? 2018S [4X2=8]

(i) $T(n) = T\left(\frac{3n}{4}\right) + n$

(ii) $T(n) = 4T\left(\frac{n}{2}\right) + nlogn$

Q1. (a) Solve following recurrences: 2018E

I. $T(n) = 3T(n/2) + n$

II. $T(n) = 3T(n/3) + n/2$

III. $T(n) = 6T(n/3) + n^2 \log n$

1. Solve following recurrences (use any method) 2018M

(i) $T(n) = 2T(n/2) + n \log n$

(ii) $T(n) = 3T(n/4) + n \log n$

(iii) $T(n) = 5T(n/3) + n \log n$

Roll No.

Q1

a) For each of the following recurrence sole them with the help of master theorem? 2016E

(i) $T(n) = 3T\left(\frac{n}{2}\right) + n^2$

(ii) $T(n) = 16T\left(\frac{n}{4}\right) + n$

(iii) $T(n) = 3T\left(\frac{n}{3}\right) + n/2$

(iv) $T(n) = 2T(n/2) + \Theta(n)$

(b) solve : $T(n) = 8T(n/2) + n^3$ and $T(n) = T(n) = 7T(n/2) + n^3$ 2019S

[a] Solve the following recurrence using master's theorem

$T(n) = 7T\left(\frac{n}{2}\right) + n^2$ 2012E

Q1. (a) For each of the following recurrence sole them with the help of master theorem? 2018E [4X2=8]

(i) $T(n) = 16T\left(\frac{n}{4}\right) + n!$

(ii) $T(n) = 2T\left(\frac{n}{4}\right) + n^{0.51}$

Q1. (a) For each of the following recurrence sole them with the help of m theorem? 2019E [3X

(I) $T(n) = 3T\left(\frac{n}{2}\right) + n^2$

(II) $T(n) = 16T\left(\frac{n}{4}\right) + n$

i) $T(n) = T(3n/4) + n$

compare $T(n) = aT(n/b) + f(n)$

$\therefore a=1 \mid b=4/3 \mid f(n) = n^1 \Rightarrow k=1$

$\Rightarrow \log_b a = \log_{4/3} 1 = 0 < c \Rightarrow T(n) = O(F(n))$

$\Rightarrow \boxed{O(n)}$

ii) $T(n) = 4T(n/2) + n \log n$

compare $T(n) = aT(n/b) + n^k \log^p n$

$\therefore a=4 \mid b=2 \mid k=1 \mid p=1$

$\Rightarrow b^k = 2^1$ and $a > b^k \Rightarrow 4 > 2 \rightarrow$ CASE 1.

$\therefore T(n) = O(n^{\log_b a}) = O(n^{\log_2 4})$

$\Rightarrow \boxed{O(n^2)}$

I. $T(n) = 3T(n/2) + n$

compare $T(n) = aT(n/b) + n^k$

$\therefore a=3 \mid b=2 \mid k=1 \mid f(n) = n$

$a > b^k \Rightarrow$ CASE 1 $\Rightarrow T(n) = O(n^{\log_b a})$

$\Rightarrow T(n) = O(n^{\log_2 3})$

Date / /
Page No.

II. $T(n) = 3T(n/3) + n/2$

compare $T(n) = aT(n/3) + n^k$

$a=3 \mid b=3 \mid k=1 \mid p=0$

$a = b^k \rightarrow$ CASE 2

OR $\Rightarrow \log_b a \rightarrow \log_3 3 = c = 1$

$\therefore T(n) = O(n^k \log n)$

$= \boxed{O(n \log n)}$

III. $T(n) = 6T(n/3) + n^2 \log n$

compare $T(n) = aT(n/b) + n^k \log^p n$

$a = 6 \mid b = 3 \mid k = 2 \mid p = 1 \mid b^k = 3^2 = 9$

$\therefore a < b^k \rightarrow$ CASE 3 $\Rightarrow T(n) = (n \log_b a) = \boxed{O(n \log_3 6)}$

(i) $T(n) = 3T(n/2) + n^2$

compare $T(n) = aT(n/b) + \underset{(f(n))}{n^k}$

$a=3 \mid b=2 \mid k=2 \rightarrow$ CASE 3

$\log_b a = \log_2 3 < 2 (k) \Rightarrow T(n) = O(f(n))$

$= \boxed{O(n^2)}$

ii) $T(n) = 16T(n/4) + n$

compare $T(n) = aT(n/b) + f(n)$

$a=16 \mid b=4 \mid f(n) = n$

CASE - 1

$\log_b a = \log_4 16 = 2 > k=1 \mid \Rightarrow T(n) = O(n^{\log_b a})$

$= \boxed{O(n^2)}$

iii) $T(n) = 3T(n/3) + n/2$

compare $aT(n/b) + n^k$

$a=3 \mid b=3 \mid k=1 \mid p=0$

CASE - 2

$\log_b a \Rightarrow \log_3 3 = k=1$

$O(n \log^{p+1} n)$

$T(n) = O(n^c \log^{p+1} n)$

$= O(n \log n)$

iv.) $T(n) = 2T(n/2) + n$

Here, $a=2 \mid b=2 \mid k=1 \mid p=0$

$a=2 \mid b^k=2 \Rightarrow a=b^k \rightarrow$ CASE 2.

$\because p=0 > -1$

$\therefore T(n) = \Theta(n^{\log_b a} \log^{p+1} n) = O(n \log n)$

a.) $T(n) = 7T(n/2) + n^2$

$a=7 \mid b=2 \mid k=2 \mid p=0$

So $a=7 \mid b^k=4 \Rightarrow b^k > a \rightarrow$ CASE 3.

$\because p=0 \geq 0$

$\therefore T(n) = \Theta(n^k \log^p n) = \Theta(n^2)$

b.) 1.) $T(n) = 8T(n/2) + n^3$

$a=8 \mid b=2 \mid k=3 \mid p=0$

$a=8 \mid b^k=8 \Rightarrow a=b^k \rightarrow$ CASE 2.

$\because p=0 > -1$

$\therefore T(n) = O(n^{\log_b a} \log^{p+1} n) = O(n^{\log_2 8} \log n)$

$= O(n^3 \log n)$

2.) $T(n) = 7T(n/2) + n^3$

$a=7 \mid b=2 \mid k=3 \mid p=0$

$a=7 \mid b^k=8 \Rightarrow b^k > a \rightarrow$ CASE 3.

$\because p=0 \geq 0$

$\therefore T(n) = \Theta(n^k \log^p n) = O(n^3)$

1.) (i.) $T(n) = 2T(n/2) + n \log n$

$a=2 \mid b=2 \mid k=1 \mid p=1$

$a=2 \mid b^k=2 \Rightarrow a=b^k \rightarrow$ CASE 2.

$\because p=1 > -1$

$\therefore T(n) = O(n^{\log_b a} \log^{p+1} n) = O(n \log^2 n)$

(ii) $T(n) = 3T(n/4) + n\log n$

$a=3 \mid b=4 \mid k=1 \mid p=1$

$a=3 \mid b^k=4 \mid b^k > a \rightarrow$ CASE 3.

$\because p=1 > 0$

$T(n) = O(n^k \log^p n) = O(n\log n)$

(iii) $T(n) = 5T(n/3) + n\log n$

$a=5 \mid b=3 \mid k=1 \mid p=1$

$a=5 \mid b^k=3 \quad a > b^k \rightarrow$ CASE 1.

$T(n) = O(n^{\log_b a}) = O(n^{\log_3 5}) = O(n^{1.46})$

Q1) a) (ii) $T(n) = 2T(n/4) + n^{0.51}$

$a=2 \mid b=4 \mid k=0.51 \mid p=0$

$a=2 \mid b^k=4^{0.51} \quad b^k > a \rightarrow$ CASE 3.

$p=0 > 0$.

$T(n) = O(n^k \log^p n) = (n^{0.51} \log^0 n) = O(n^{0.51})$

i) $T(n) = 16T(n/4) + n!$

$a=16 \mid b=4 \mid f(n)=n! \mid p=0$

CASE-3: $p > 0$

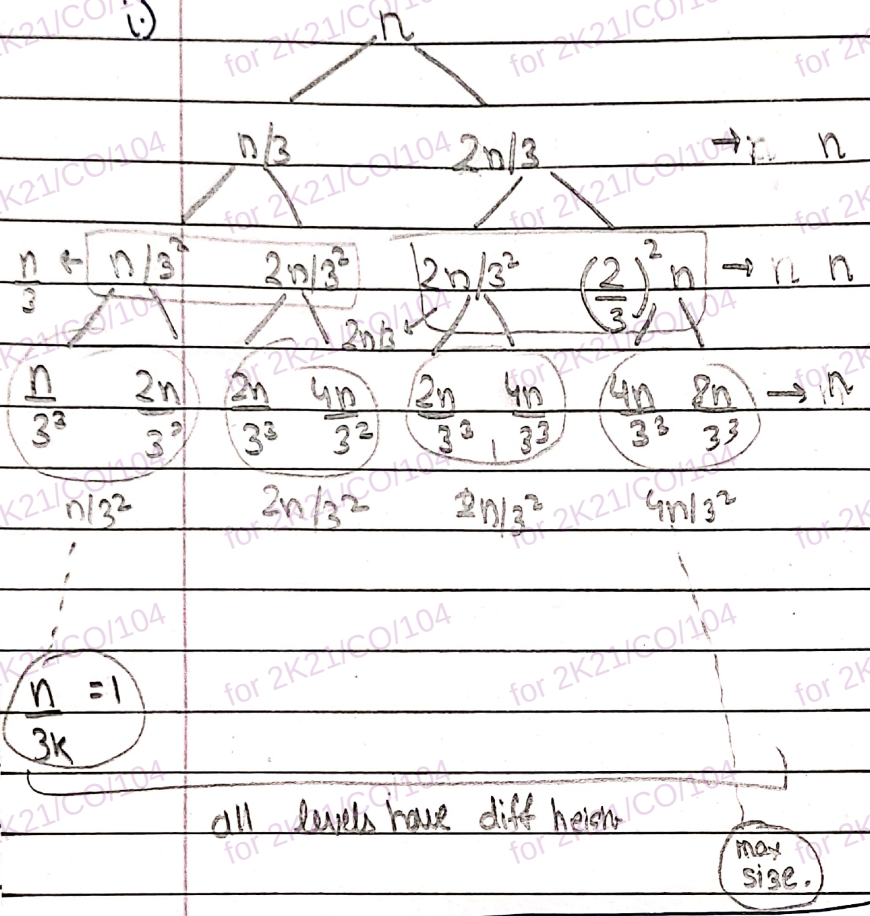
$T(n) = O(f(n)) = \boxed{O(n!)}$

b) For the recurrence equation $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n$ [3 x 2 = 6]

i) Apply recurrence tree method to find complexity. Show each and every step involved in calculation. 2017M

ii) For the complexity calculated in the previous part, use the same to prove with the help of substitution method. Use $\log_2 3$ and $\log_2\left(\frac{3}{2}\right)$ as 3/2 and ½ respectively. Also write the minimum value of constant c for which the proof holds.

i)



lvl 0 = $\left(\frac{1}{3}+\frac{2}{3}\right)^0 n$

lvl 1 = $\left(\frac{1}{3}\right)^1\left(\frac{2}{3}\right)^0 n + \left(\frac{1}{3}\right)^0\left(\frac{2}{3}\right)^1 n$ = n

lvl 2 = cn

lvl k = cn

for height

$\frac{n}{(3/2)^k} = 1 \Rightarrow k = \log_{3/2} n$

∴ TC = work at each lvl × ht

$\therefore O(n \log_{3/2} n)$

ii) using substitution method

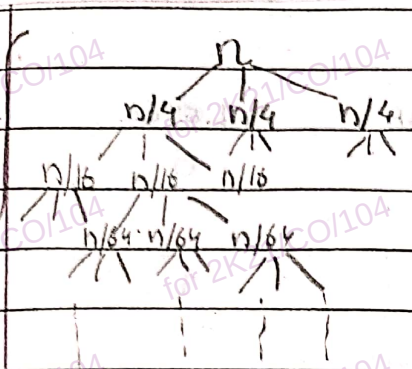
$t(n) = t(n/3) + t(2n/3) + n$

$\leq c(n/3)^c + c(2n/3)^c + n$

$\leq 2cn^c/3^c + n$

1. Solve following recurrences 2020M

(i). $T(n) = 3T(n/4) + \Theta(n^2)$ using recursion tree method



n^2

$(3/16)\,n^2$

$(3/16)^2 n^2$

$(3/16)^3 n^2$

Series by adding all TC is a ∞ G.P

$T = n^2 + \frac{3}{16}n^2 + \left(\frac{3}{16}\right)^2 n^2 + \cdots$

$T = n^2\left(1 + \frac{3}{16} + \left(\frac{3}{16}\right)^2 + \cdots\right)$

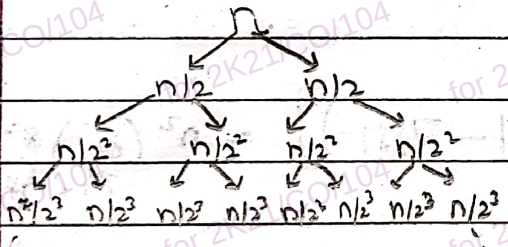
$T = n^2\left(\frac{1}{1-3/16}\right) \Rightarrow T = \frac{16}{13}n^2$

$\therefore O(n^2)$

[b] Consider the recurrence

$T(n) = 2T(n/2) + n^2$ 2012E

Solve it using recurrence tree method.



LEVEL	Work
0	n^2 = n^2
1	$(n/2)^2 \times 2^1$ = $n/2^1$
2	$(n/2^2)^2 \times 2^2$ = $n/2^2$
3	$(n/2^3)^2 \times 2^3$ = $n/2^3$
$n/2^k$	$n/2^k$ k	$(n/2^k)^2 \times 2^k = n/2^k$

$\therefore$ total work; $T(n) = n^2 + \frac{n^2}{2} + \frac{n^2}{4} + \frac{n^2}{8} - \cdots + \frac{n^2}{2^k}$

$= n^2\left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} - \cdots\right) = n^2\left(\frac{1}{1-1/2}\right) = 2n^2$

$\therefore$ TC = $O(n^2)$

(b) Solve $T(n) = 2T\left(\frac{n}{2}\right) + n$ with the help of recursion tree method. Show each and every step involved. 2018S [6 marks]

Tree is same as above $\therefore$ WORK

$\log_2 n$ terms: $n = n$; $\frac{n}{2} \times 2 = n$; $\frac{n^2}{4} \times 4 = n$

$\therefore$ TC = $n + n + n + \cdots$ ($\log n$ times)

$= n\log_2 n$

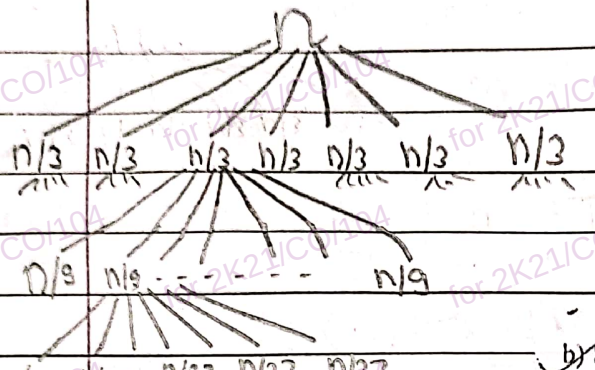
$\therefore O(n \log n)$

(b) Solve $T(n) = 7T\left(\frac{n}{3}\right) + n^2$ with the help of recursion tree method. Show each and every step involved. 2019E [4 marks]

(b) Solve $T(n) = 7T\left(\frac{n}{3}\right) + n^2$ with the help of recursion tree method. Show each and every step involved. [6 marks] 2018E

2. Solve following recurrence using recurrence tree method showing complete steps. 2018M (4)
$T(n) = 7T(n/3) + n^2$

RECURSION TREE : $T(n) = 7T(n/3) + n^2$



PARTITIONING TIME	LEVEL
n^2	0
$n^2(7/3)$	1
$n^2(7/3)^2$	2

b) Solve $T(n) = 7T\left(\frac{n}{3}\right) + n^2$ with the help of recursion tree method. Show each and every step involved 2018E (6)

$\log_2 n$ times.

$\therefore T = n^2 + n^2(7/3) + n^2(7/3)^2 + \text{-----} = \sum \left(\frac{7}{3}\right)^k n^2$

$\left[S = a\left(\frac{1-r^n}{1-r}\right)\right]$

$= n^2\left(\frac{1 - 7/3^{\log_2 n}}{8 - 4/7}\right)$

$= \frac{n^2}{6}\left(7^{\log_2 n} - 1\right)$

$= n^2\left[\left(1 - (7/81)^{\wedge \log_3 n + 1}\right) / \left(1 - 7/81\right)\right] = n^2 \left(\frac{81}{74}\right)\left(\frac{7}{81}\right)^{\wedge \log_3 n}$

$= O(n^2 \log_3 n)$

Date / /
Page No.

UNIT 2 – SEARCHING AND SORTING

Q2
a) Write quick sort algorithm. Drive worst case complexity for this algorithm. Also explain in detail what special incite does average-case gives to us? 2016E (2+2+4)

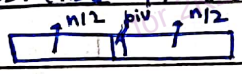
Quick Sort is an in-place comparison-based sorting algorithm. It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to ~~which~~ whether they are less than or greater than the pivot, which then becomes its final position. This work is repeated recursively on the sub-arrays, until the whole array is sorted.

ALGORITHM

Q2. (a) Write quick sort algorithm. Drive worst case and best case complexity for this algorithm. 2019E [5 marks]

```
QUICKSORT (A, p, r)
    if p<r
        then q ← PARTITION (A, p, r)
             QUICKSORT (A, p, q-1)
             QUICKSORT (A, q+1, r)
initial call is QUICKSORT (A, 1, length[A])

PARTITION (A, p, r)
    x ← A[r]
    i ← p-1
    for j ← p to r-1
        do if A[j] ≤ x
            then i ← i+1
                 exchange A[i] ↔ A[j]
    exchange A[i+1] ↔ A[r]
    return i+1
```

i) BEST CASE : pivot is median.



$\therefore T(n) = T(n/2) + T(n/2) + n$

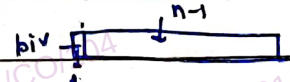
$\Rightarrow T(n) = 2T(n/2) + n$

Similarly
$T(n/2) = 2T(n/4) + n/2 \times 2$
$T(n/4) = 2T(n/8) + n/4 \times 2^1$
⋮ ⋮ ⋮ } log n
$+ \quad T(1) = 1 \qquad \times 2^n$

$T(n) = n \log n$

$\therefore$ T.C. $\Rightarrow \boxed{\Omega(n \log n)}$

ii) WORST CASE : pivot is smallest / largest ele



$\therefore T(n) = T(1) + T(n-1) + n$

$T(n-1) = T(n-2) + n-1$

$T(n-2) = T(n-3) + n-2$

⋮ ⋮

$T(1) = 1$

$T(n) = (n+1) + n + (n-1) + \dots 1$

$\Rightarrow T(n) = \frac{(n+2)(n+1)}{2} \Rightarrow \boxed{O(n^2)}$

$\leq cn^2$

Q2
a) Write down the randomized quicksort algorithm. Also explain in detail what special incite does average-case gives to us? 2017E (2+3)

(b) Write an algorithm for quick sort. Explain with an example and show the analysis for the algorithm. 2019E [5 Marks]

AVERAGE CASE

The avg case provides insights on random inputs showing that the expected runtime is O(nlogn) and matches best case runtime. It indicates that the algo will have a good chance of achieving best case for any input array. The analysis also quantifies the probability of algorithm taking more time than a fixed time to be very small. Thus, it shows its efficiency in practice and its resilient to worst-case inputs.

Q2. (a) Write the quick sort algorithm. Analyze its efficiency. Apply the algorithm to sort the list {4, 1, 6, 3, 9, 2, 7, 5} 2018S [7 marks].

piv = n-1

{4, 1, 6, 3, 9, 2, 7, 5}

PARTITION BY 5

{4, 1, 3, 2} 5 {6, 9, 7}

PB2

{1} 2 {4, 3}

PB7

{6} 7 {9}

PB1

{} 1 {}

PB3

{} 3 {4}

PB6

{} 6 {}

PB9

{} 9 {}

PB4

{} 4 {} 5

1 2 6 7 9

∴ {1, 2, 4, 5, 6, 7, 9} ← sorted array

Q4 (a) Give a divide and conquer pseudocode to find the minimum and maximum elements in an array. Also form the recurrence relation and solve the recurrence relation to obtain the complexity of the pseudo code. Also apply the pseudo code to the given below array to find the minimum and maximum of the elements?

28	60	72	56	18	69	21	95	31	17	23	90	70	44	52	24

(b) An array of n elements contains all but one of the integers from 1 to n + 1.

(i) Give the best algorithm you can for determining which number is missing if the array is sorted, and analyze its asymptotic worst-case running time.

(ii). Give the best algorithm you can for determining which number is missing if the array is not sorted, and analyze its asymptotic worst-case running time.

2018E (5+5=10)

Date / /
Page No.

a.) PSEUDO CODE ≈ C++ CODE

```
int findMinMax (int A[], int l, int r) {
    int max; min;
    if (l == r)
        max = A[l], min = A[l];
    else if (l+1 == r) {
        if (A[l] < A[r]) {
            max = A[r];
            min = A[l];
        } else {
            max = A[l];
            min = A[r];
        }
    } else {
        int mid = l + (r-l)/2;
        int left[] = findMinMax (A, l, mid);
        int right[] = findMinMax (A, mid+1, end);
        if (left[0] > right[0]) max = left[0];
        else max = right[0];
        if (left[1] < right[1]) min = left[1];
        else min = right[1];
    }
    int ans[] = {max, min};
    return ans;
}
```

```
Func MinMax (A[l...r], min, max)
    if (r = l)
        min ← A[l]; max ← A[r]
    else if (r - l = 1)
        if A[l] ≤ A[r]
            min ← A[l]; max ← A[r]
        else min ← A[r]; max ← A[l];
    else
        MinMax (A[l ... (l+r)/2], min, max);
        MinMax (A[(l+r)/2 + 1 ... r], min2, max2);
        if min2 < min  min ← min2
        if max2 > max  max ← max2
```

TIME COMPLEXITY

$T(n) = 2T(n/2) + 2$

using masters theorem

$T(n) = 3n/2 - 2$

$\therefore$ O(n)

space complexity → O(log n)

3. Given an array A which stores 0 and 1, such that each entry containing 0 appears before all those entries containing 1. In other words, it is like {0, 0, 0,..., 0, 0, 1, 1,..., 111}. Design an algorithm to find out the small index i in the array A such that A[i] = 1 with O(log n) time complexity.

2018M [5]

Date / /
Page No.

USING BINARY SEARCH METHOD ⇒ returns smallest index having 1 in A.

```
int firstOne (int[] A, int n){
if (A[0] == 1) return 0;
else if (A[n-1] == 0) return -1;
else {
    int left = 0; int right = n-1 ; bool isFound = 0;
    while (!isFound) {
        mid = (left + right) / 2;
        if (A[mid] == 0)
            left = mid + 1;
        else {
            if (A[mid-1] == 0) isFound = 1;
            else right = mid - 1;
        }
    } return mid;
}
```

2. You are given a sorted array of n elements which has been circularly shifted. For example, {35, 42, 5, 12, 23, 26} is a sorted array that has been circularly shifted by 2 positions. Give an O(log n) time algorithm to find the largest element in a circularly shifted array. (The number of positions through which it has been shifted is unknown to you.) 2019M (5)

5. Suppose you are given an array A[1..n] of sorted integers that has been circularly shifted k positions to the right. For example, [35, 42, 5, 15, 27, 29] is a sorted array that has been circularly shifted k = 2 positions, while [27, 29, 35, 42, 5, 15] has been shifted k = 4 positions. We can obviously find the largest element in A in O(n) time. Describe an O(log n) algorithm. 2020M (5)

We can observe that from any pivot element, either left or right half of the array is sorted.

```
int BinarySearch (int a[], int target, int n) {
    lo = 0 ; hi = n-1 ;
    while (lo <= hi)
        int mid = (lo + hi) / 2
        if (a[mid] == target) return mid;
        // left side is sorted
        if (a[lo] <= a[mid]) {
// check if target on left   if (target >= a[low] && target <= a[mid]
                high = mid - 1;
            else
                lo = mid + 1;
        }
    else {  // right is sorted.
        if (target >= a[mid] && target <= a[hi]
            lo = mid + 1
        else
            hi = m - 1
    } } return -1; }
```

Date / /
Page No.

(b) Using divide and conquer strategy count the number of inversions (out of order pairs) in an array. Let the elements be $x_1, \ldots, x_n$. There is an inversion if $x_i > x_j$ and $i < j$. 2019S (5+5=10)

We will use merge sort's modification, we will recursively divide the array in half and count inversions in subarrays. → log n steps n times → O(n log n)

Algo

```
COUNT-INVERSIONS (A, lo, hi)                    lo → low
                                                hi → high
    if lo >= hi
        return 0
    mid = (lo + hi)/2
    left = COUNT-INVERSIONS (A, lo, mid)
    right = COUNT-INVERSIONS (A, mid+1, hi)
    inversions = left + right + MERGE (A, lo, mid, hi)
    return inversions
```

```
MERGE (A, lo, mid, hi)
    n1 = mid - lo + 1
    n2 = hi - mid
    let L[1..n1], R[1..n2] be new arrays.
    for (i = 1 → n1)      L[i] = A[p+i-1]
    for (j = 1 → n2)      R[j] = A[q+j]
    L[n1+1] = ∞  | R[n2+1] = ∞
    i = 1  | j = 1  | inversions = 0
    for (k = lo → hi)
        if (L[i] <= R[j])
            A[k] = L[i]; i++
        else
            inversions += (n1 - i + 1)
            A[k] = R[j]; j++
    return inversions
```

Q2 Given a sorted array of non-repeated integers A [1...n], check whether there is an index i for which A[i] = i. Modify the divide and conquer algorithm/ pseudo code for binary search to do the needful. Give an example to justify your implementation. Analyze the complexity for the same? 2017M [2+2+1]

Given a sorted array of distinct integers, some of which may be negative, give an algorithm to find an index i such that $1 <= i <= n$ and A[i] = i provided such an index exists. If there are many such indices, the algorithm can return any one of them. 2019M (4)

array → sorted

Here we check if value at middle index = index itself and return if true, else we check whether the value at the middle index is greater than the index. If it is we recursively search the left half of the array. Otherwise we recursively search the right half.

```
int func (vector <int> arr, int lo, int hi) {
    if (lo > hi)  return -1;        // base case
    int mid = (lo+hi) / 2;
    if (A[mid] == mid)
            return mid;
    else if (A[mid] > mid)
            return func (A, low, mid - 1);
    else
            return func (A, mid+1, hi);
}
```

we example call func ({-10, -5, 0, 3, 7}, 0, 4)

i) MAIN CALL
mid = 2
A[mid] = 0 ≠ 2

ii) func ({-10, -5, 0, 3, 7}, 3, 4)
mid = 3
A[mid] = 3 = 3 → return 3 ✓

(b) Write down recursive binary-search algorithm and apply the same to find the value 2 in the following array. Array is as follows 1,2,3,4,5,6,7,8,9,10. 2019E 2 [5 marks]

```
BINARY-SEARCH (arr, n, lo, hi) {
    WHILE (lo < hi) {
        mid = (lo+hi)/2
        if (n == arr[mid]) return mid;
        else if (n > arr[mid]) lo = mid+1;
        else if (n < arr[mid]) hi = mid-1;
    }
}
```

1 based index

1.) arr = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
mid = 5 < 2 → hi = 4

2.) arr = 1, 2, 3, 4
mid = 2 = 2 → return 2

return binarySearch (arr, n, mid+1, hi)
return binarySearch (arr, n, lo, mid-1)

Q1. (a) Suppose that a divide and conquer algorithm solves a problem of size n by first solving three instances of size n/2 and then taking O(n) additional basic steps. If the time to create the three instances is O(n 2), what is the running time, T(n), of the algorithm. 2019S

Clearly

$T(n) = 3(T(n/2)) + n^2 + n$

by master's theorem, case 1:

$a = 3 \mid b = 2 \mid f(n) = O(n^2)$

$\therefore$

$T(n) = O(n^{\log_b(a)}) \Rightarrow \boxed{T(n) = O(n^{1.585})}$

$T(n)$ = Solving subproblem + dividing + addn basic steps

3. $T(n/2) + O(n^2) + O(n)$

Q2. (a) Write algorithm to convert random array into a Max-heap. Also write algorithms for delete-Max() and Insert() for this Max-heap. 2018E

(b) Consider following unsorted array A[]={24, 12, 8, 17, 11, 76, 32, 54, 9, 23, 44}. Convert this array in to a MAXHEAP. Also write MakeHeap() algoritm to convert this array into a Max Heap. Apply algorithm on given array and show the output in tree and array form. Now apply remaining part of heap sort algorithm and show array/tree content after every round (deleteMax). 2018S (4+6=10)

```
MAKE-HEAP(A){
    for (int i = A.size(); i >= 0; i--){
        max-heapify (arr, i);
    }
}
```

Eg) A = {24, 12, 8, 17, 11, 76, 32, 54, 9, 23, 44}

DO LIKE NEXTQ

```
MAX-HEAPIFY (A, i)
1.  largest = i
2.  l = LEFT(i)
3.  r = RIGHT(i)
4.  if (l ≤ A.heap-size and A[l] > A[largest])
5.      largest = l
6.  else if (r ≤ A.heap-size and A[r] > A[largest])
7.      largest = r
8.  if (largest ≠ i)
9.      Swap(A[largest], A[i])
10.     MAX-HEAPIFY (A, largest)
```

```
INSERT (A, key)
1.  A.heap-size = A.heap-size + 1
2.  A[A.heap-size] = -∞
3.  INCREASE-KEY (A, A.heapsize, key)
```

```
INCREASE-KEY (A, i, key)
1.  if (key < A[i])
2.      error
3.  A[i] = key
4.  while i > 1 and A[Parent(i)] < A[i]
5.      exchange A[i] with A[Parent(i)]
6.      i = PARENT(i)
```

```
DELETE-MAX (A, i)
    A[i] = A[A.heapsize]
    A.heapsize = A.heapsize + 1
    MAX-HEAPIFY (A, i
```

b) For the array: 15, 19, 10, 7, 17, 6 perform Heap-Sort operation on it. Write all the algorithms involved in it. 2016E (4+4)

[c] Explain heap sort algorithm in detail. Also analyse its complexity. 2012E 7

```
HEAP-SORT (A, n)
BUILD MAX HEAP   for (int i=n/2 ; i>=1 ; i--)
                     MAX-HEAPIFY (A, i)
to delete (not in algo)  for (i=n ; i>=1 ; i--)
                     swap (A[1], A[i])
                     MAX-HEAPIFY (A, n-1)
```

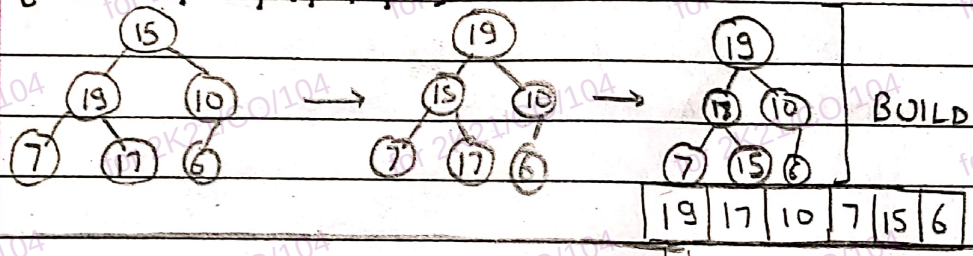
ALGORITHM

1. Set n to the length of input array A
2. Build max heap on the array A[1...n]
3. While n > 1, repeat the steps:
   a.) Exchange A[1] with A[n]
   b.) decrement n by 1
   c.) call MAX-HEAPIFY (A, 1) to ensure A[1...n-1] satisfy max-heap.
4. returns A having increasing order

TIME-COMPLEXITY

Since ~~each~~ call to build max heap takes O(n) time and each of n-1 calls to MAX-HEAPIFY takes O(log n) time, the HEAPSORT algo takes O(n log n) time.

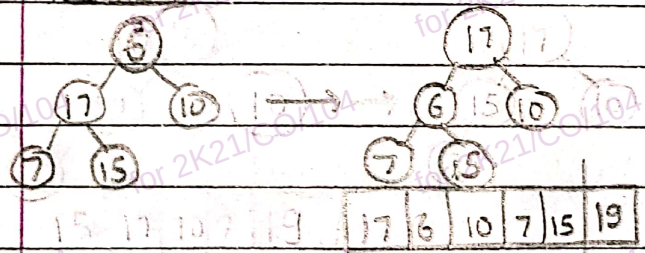
Eg.) {15, 19, 10, 7, 17, 6}



BUILD MAX HEAP.

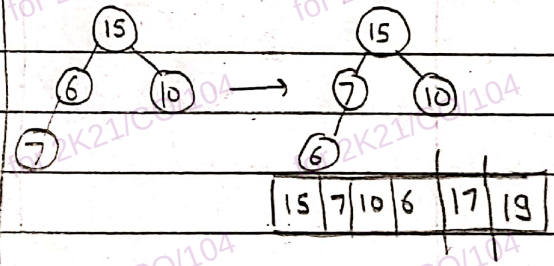
19	17	10	7	15	6

delete 19



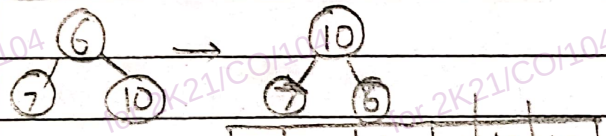
17	6	10	7	15	19

delete 17



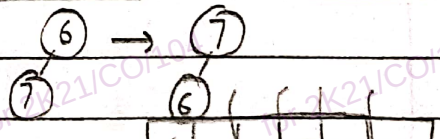
15	7	10	6	17	19

delete 15



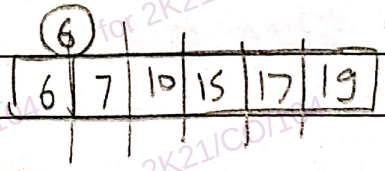
10	7	6	15	17	19

delete 10



6	7	10	15	17	19

delete 7



6	7	10	15	17	19

delete 6

6	7	10	15	17	19

Sorted array

Q. Explain ~~in detail Strassen's method~~ of matrix multiplication. Write down the Algorithm and the equations involved in algorithm, explain its complexity.
2017M

Strassen's matrix multiplication is a recursive algorithm for multiplying nxn matrices which outperforms the naive $O(n^3)$ MMA for sufficiently large values of n. It uses divide and conquer by dividing the matrices such that only 7 recursive multiplications of n/2 and n/2 and n^2 scalar additions and subtractions are required.

STEPS

1. Divide the input matrices A and B into n/2 and n/2 submatrices
2. Using $O(n^2)$ scalar additions and subtractions, compute 14 matrices each of which is of n/2 and n/2. ($A_1, A_2, \ldots B_1, B_2 \ldots$)
3. Recursively compute the seven matrix products $P \rightarrow V = A_i B_i$
4. Compute the desired submatrices $C_{11}\ C_{12}\ C_{21}\ C_{22}$ of the resultant matrix C by adding or subtracting various combinations of $P \rightarrow V$ matrices using n^2

EQUATIONS

$P = (A_{11} + A_{22})(B_{11} + B_{22})$
$Q = (A_{21} + A_{22}) \cdot B_{11}$
$R = (B_{12} - B_{22}) \cdot A_{11}$
$S = A_{22}(B_{21} - B_{11})$
$T = B_{22}(A_{11} + A_{12})$
$U = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$
$V = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$

$C_{11} = P + S - T + V$
$C_{12} = R + T$
$C_{21} = Q + S$
$C_{22} = P + R - Q + V$

TIME COMPLEXITY

$T(n) = 7T(n/2) + n^2$
$= \Theta(n^{\log_2 7})$
$= O(n^{2.81})$

(2+4+4=10)

Q2 (a) We want to merge n sorted lists $L_1, L_2, \ldots, L_n$ of sizes $l_1, l_2, \ldots, l_n$ respectively. Suppose that at any point of time, we are allowed to merge only two sorted lists, that is, simultaneously merging t sorted lists for $t > 3$ is not allowed. (For example, the sorted lists may be residing in files in a palmtop computer which has very little memory and allows only three opened file pointers at any time.) The effort associated with merging two sorted lists of sizes u and v is taken as u + v. Your task is to select the merging sequence in such a way that the total effort of merging the given n lists is minimized. Design one efficient algorithm for doing this task. 20195

Suppose that n = 4 lists are given with respective sizes 10, 20, 30, 40. Find the efforts of the following two ways of merging L1,L2,L3,L4:

merge(merge(L_1,L_2),merge(L_3,L_4)) and merge(merge(merge(L_1,L_2),L_3),L_4).

Date / /
Page No.

ALGORITHM

1. obtain the huffman tree on n symbols with weights $l_1, l_2, l_3 \ldots l_n$.

2. If d_i is the depth of the leaf l_i in a prefix tree with leaves $l_1, l_2, l_3 \ldots l_n$ Then Huffman's algo minimises $\sum_{i=1}^{n} d_i l_i$.

3. It suffices to note that this quantity is precisely the total effort of merging $L_1, L_2 \ldots L_n$ acc to prefix tree

when n=4

i.) EFFORTS (merge (merge (L_1, L_2), merge (L_3, L_4)))

= (10+20) + (30+40)
+ EFFORTS (merge (L_1+L_2, L_3+L_4))

= (30 + 40) + [(30 + 70)]

= 200

ii.) EFFORTS (merge (merge (merge (L_1, L_2) L_3), L_4))

= (10+20) + Eff (merge (merge (L_1+L_2, L_3), L_4))

= (30) + (30 + 30) + EFFORTS (merge ($L_1+L_2+L_3, L_4$))

= 30 + 60 + (60+40)

= 190

UNIT-3 (Greedy)

5. Consider following problem of frog willing to reach position n in minimum number of jumps. The frog begins at position 0 in the river. Its goal is to get to position n. There are lilypads at various positions. There is always a lilypad at position 0 and position n. The frog can jump at most r units at a time from one lilypad to another lilypad. Goal is to find the path the frog should take to minimize jumps, assuming a solution exists. Write a greedy algorithm pseudo codes for your solution. 2018M

Solve following instance of problem using your greedy algorithm with r = 3.

(5)

ALGORITHM

1. let J be an empty series of jumps
2. initial position, x = 0.
3. while x < n :
   i.) Find further lilypad l, reachable from x that is not after n. in next r steps.
   ii.) add a jump to J from x to i's position
   iii.) set x to i's location.

Result J path.

CODE:

```
vector<int> minJumps(vector<int> lilypads, int n, int r){
    vector<int> path;
    int current_position = 0;
    while(current_position != n){
        int max_possible = current_position;
        for(int i = current_position + 1; i <= current_position + r && i <= n; i++){
            if(lilypads[i] > lilypads[max_possible]){
                max_possible = i;
            }
        }path.push_back(max_possible);
        current_position = max_possible;
    }return path;
}
```

GIVEN r = 3

i.)	ii.)	iii.)	iv.)
curr = 0	curr = 2	curr = 5	curr = 8
max possible = 2	max poss = 5	max poss = 8	max poss = 10
∴ curr = 2	∴ curr = 5	∴ curr = 8	∴ curr = 10
path = {0, 2}	{0, 2, 5}	{0, 2, 5, 8}	{0, 2, 5, 8, 10}

∴ ans → {0, 2, 5, 8, 10} path.

Date / /
Page No.

3. You are given n events where each takes one unit of time. Event i will provide a profit of g_i dollars ($g_i > 0$) if started at or before time t_i where t_i is an arbitrary real number. (Note: If an event is not started by t_i then there is no benefit in scheduling it at all. All events can start as early as time 0.) Give the most efficient algorithm you can to find a schedule that maximizes the profit.

2020M (5)

Acc to Greedy Algorithm:
we will chose the most profitable event and schedule it as late as possible before its deadline.

∴ ALGORITHM

1. Sort events by profits in descending order.
2. Initialise time t to 0 and profit p to 0.
3. For each event i in the sorted order:
   i.) if time at end of event >= t_i, skip and move to next event.
   ii.) else, find latest start time for event as max (t, end time − duration)
   iii.) Schedule the event at this time and add profit to total profit.
   iv.) update curr time t to the end of the scheduled event.
4. Return total profit p.

PSEUDO CODE

```
func maxprofit (events[]) {
    events.sort (desc);
    t = 0; p = 0;
    for (int i=0; i< events.size(); i++) {
        if (end_time[i] <= ti)
            continue;
        new_start_time = max (t, end time - duration);
        p += gi
        t = end_time
    }
    return p;
}
```

2. Write algorithm for Huffman coding and find Huffman codes for following data

Character	a	b	c	d	e	f	g
Frequency	37	18	29	13	30	17	6

Compute average code length for given data. 2020M (5)

C → set of characters (n)

Frequency of char → f[c]

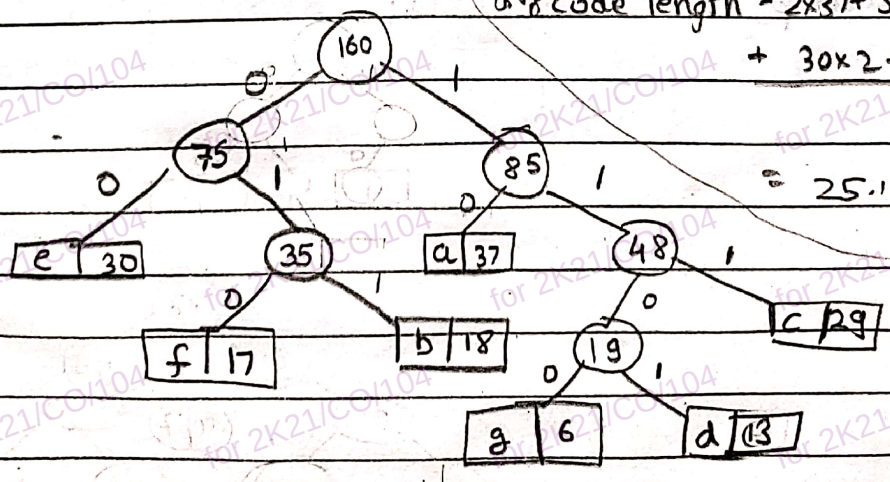
Huffman coding is a data compression algorithm. Here, the algorithm builds a tree T corresponding to the optimal code in bottom up manner. It begins with a set of |C| leaves and performs a sequence of |C| − 1 merging operations to create a final tree. A min PQ, Q based on f, is used to identify the two least frequent objects to merge together. The result of the merger of two objects is a new object whose frequency is the sum of the frequencies of the two objects that were merged.

HUFFMAN (c)

1. n ← |c|
2. Q ← c
3. for (i ← 1 to n−1)
4. do allocate a new node z.
5. left[z] ← x ← EXTRACT-MIN (Q)
6. right[z] ← y ← EXTRACT-MIN (Q)
7. f(z) ← f[x] + f[y]
8. INSERT (Q, z)

return EXTRACT-MIN (Q) // return root of tree

CHAR	a	b	c	d	e	f	g
freq	37	18	29	13	30	17	6
Code	10	011	111	1101	00	0101	1100



Avg code length = (2×37 + 3×18 + 29×3 + 13×4 + 30×2 + 17×3 + 6×4) / 160

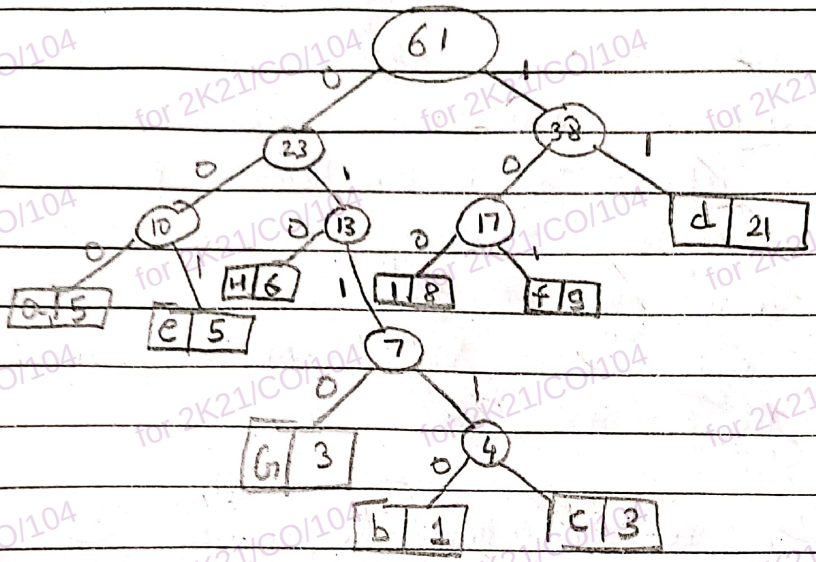
= 25.125 bits

3. Write algorithm for Huffman coding and apply it on following data:

Char	a	b	c	d	e	f	g	h	i
Frequency	5	1	3	21	5	9	3	6	8

Also compute average code size. (5)

Date / /
Page No.



CHAR	FREQUENCY	CODE
a	5	000
b	1	01110
c	3	01111
d	21	11
e	5	001
f	9	101
g	3	0110
h	6	010
i	8	100

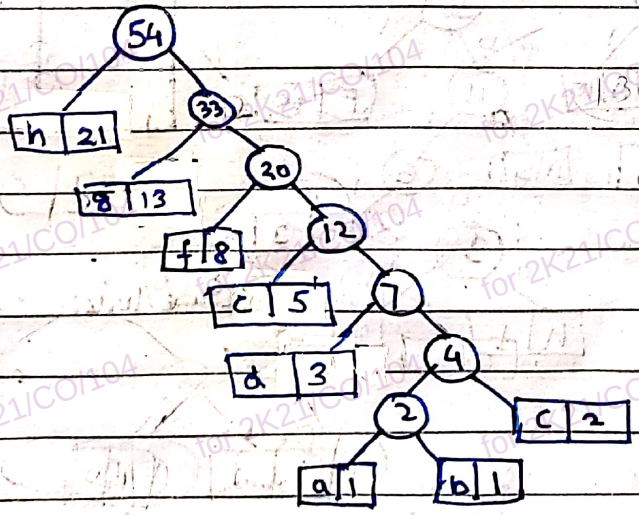
$$\text{Avg Code Size} = \frac{(5\times3 + 5\times1 + 3\times5 + 2\times21 + 5\times3 + 9\times3 + 3\times4 + 6\times3 + 8\times3)}{61}$$

$$= 2.9 \text{ bits.}$$

(b) Write down the pseudocode and obtain Huffman codes for the following instance and for the same and analyze its complexity?

Character	a	b	c	d	e	f	g	h
No of occurrences (in thousands)	1	1	2	3	5	8	13	21

(2+2+1)



CHAR	FREQ	CODE
a	1	111100
b	1	111101
c	2	11111
d	3	1110
e	5	110
f	8	10
g	13	0
h	21	

TIME COMPLEXITY

From heap, each item require $O(\log n)$ to find cheapest ot.
There are n iterations

$\therefore$ $\boxed{TC = O(n \log n)}$

[c] Greedy approach guarantees to produce optimal solution. True or false? Justify your answer. 2012E

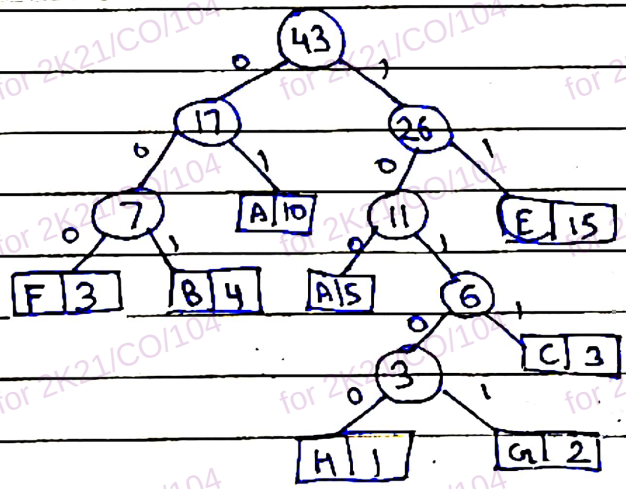
False

It is because there are scenarios where a locally optimal choice at one step may lead to a non-optimal solution in the future. Its because the algorithm does not consider the global consequences of the local optimal choice. There are many problems like travelling salesman problem, 0-1 knapsack problem, etc which dont produce optimal solution using greedy methods.

(b) Apply Huffman coding and assign binary codes to following characters. 2018E

Character	Frequency
A	10
B	4
C	3
D	5
E	15
F	3
G	2
H	1

(6+4=10)



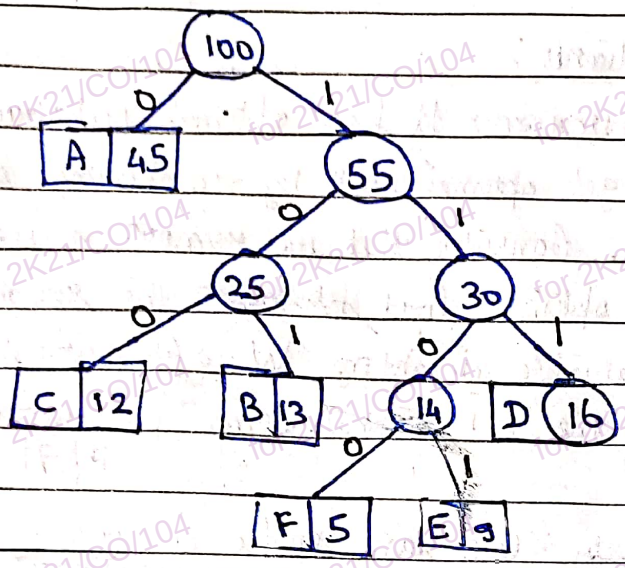
CHAR	FREQ	CODE
A	10	01
B	4	001
C	3	1011
D	5	100
E	15	11
F	3	000
G	2	10101
H	1	10100

[b] Given the characters <a, b, c, d, e, f> with the following probability of occurrence 2012E

P = <45, 13, 12, 16, 9, 5>

Build a binary tree according to greedy strategy for defining an optimal Huffman code. 8

Date / /
Page No.



CHAR	FREQ	CODE
A	45	0
B	13	101
C	12	100
D	16	111
E	9	1101
F	5	1100

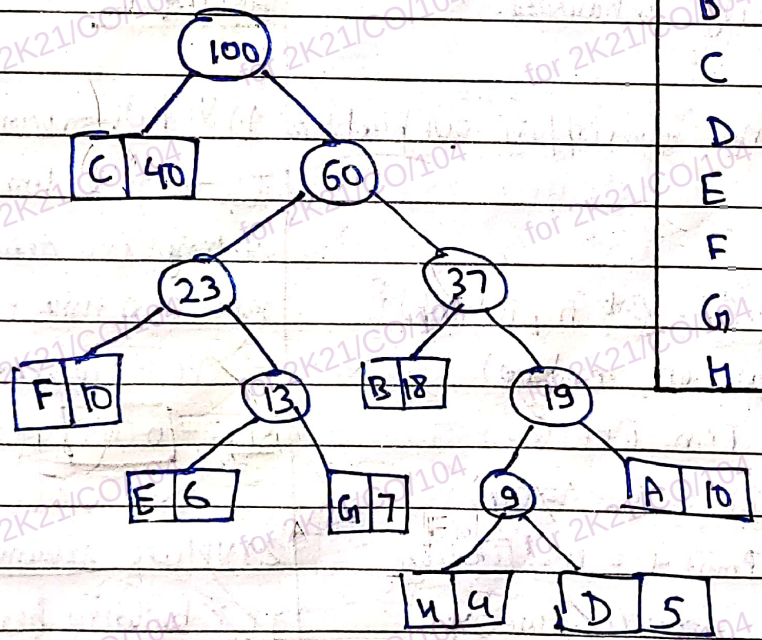
Q5. (a) Probability of occurrence of various characters in a text segment is as follows: 2019S

Character	A	B	C	D	E	F	G	H
Probability	0.10	0.18	0.40	0.05	0.06	0.10	0.07	0.04

Find Huffman encoding for this text and compute average code length. What will be saving compared to fixed length encoding for a text segment of size 1000 characters?

Convert into freq table for convenience (100 char)

CHAR	FREQ	CODE
A	10	1111
B	18	110
C	40	0
D	5	11101
E	6	1010
F	10	100
G	7	1011
H	4	11100



average code length = $(0.1\times4 + 0.18\times3 + 0.40\times1 + 0.05\times5 + 0.06\times4 + 0.1\times3 + 0.07\times4 + 0.04\times5)$

$= 2.33$ bits.

Since we used fixed length, avg = 3 bits

∴ bits saved = $(3-2.33)\times1000$

= 670 bits

Justify your reason for solving fractional knapsack problem with greedy approach by identifying ingredients of greedy strategy in it. Write the appropriate pseudo code that would have solved the fractional knapsack with greedy choice. Also, analyse your code in two scenarios viz. when v_i/w_i values is maintained as an array and when the same is maintained as a heap. 2016E

Q7. (a) Write algorithm for solving fractional Knapsack problem. 2018S

Date / /
Page No.

Identifying Ingredients:

1. **OPTIMAL SUBSTRUCTURE:** It has optimal substructure, which means we can get optimal soln by combining optimal soln to subproblems. Consider that we remove a weight w of one item j from optimal load, the remaining load must be the most valuable load weighting $W-w$ (at most) that the thief can take from the n-1 original items plus w_j-w of item j.

2. **Greedy-Choice Property:** We can make a locally optimal choice at each step and still end up with a globally optimal solution. We must compute v_i/w_i for each item. So, we can begin by taking as much as possible of an item with greatest value per weight. If exhausted, he can take next item and so on till capacity exhausted.

3. There are no overlapping subproblems

ALGO (using code)

```
int FracKnap(Item arr[], int n, int cap){
    Sort(arr, arr+n, compare)
    for(int i=0; i<n; i++){
        if(arr[i].weight <= cap){
            totalProfit += arr[i].price;
            cap -= arr[i].weight;
        }else{
            totalProfit += arr[i].price * (cap/arr[i].weight);
            break
        }
    }return totalProfit;
}
```

ANALYSIS

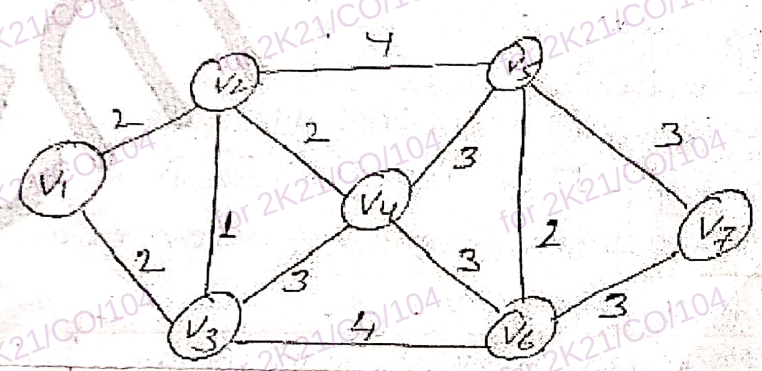
1) v_i/w_i maintained as array
TC → It is dominated by array sorting in desc order. Loop takes O(n) time.
∴ TC = nlogn + n ⇒ O(nlogn)
SC :- O(n) for array

2) v_i/w_i maintained as Heap
TC: building heap takes O(n)
loop filling knapsack takes O(k)
k → # items that fit in sack
∴ TC → O(nlogk + k)
better
SC → O(n) for heap

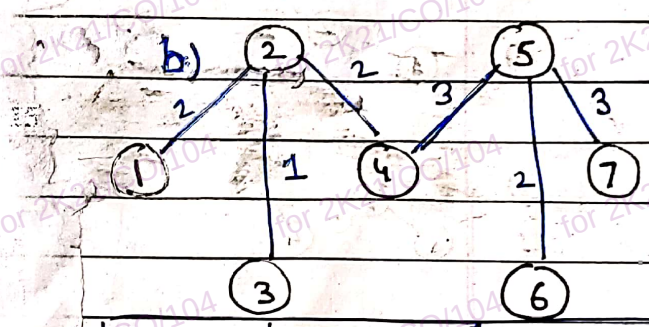
Q5. (a) What is a Spanning tree? Explain Prim's Minimum cost spanning tree algorithm with suitable example. 2018S

A spanning tree is a subgraph of an undirected graph which includes all the vertices of the original graph. Thus, it is a tree that connects all the vertices of a given graph. So, G is a spanning tree which has all the edges of a graph H with minimum no of vertices.

5 For the graph shown below obtain the following: 2012E
[a] All spanning trees
[b] Minimum spanning tree by Kruskal's method
[c] Shortest path from V_1 to V_7.



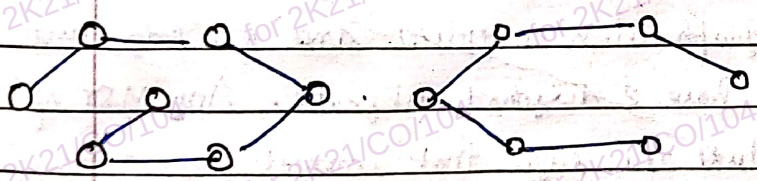
b)



CHOICE	VERTEX	WEIGHT
1	{2,3}	1
2	{1,2}	2
3	{2,4}	2
4	{5,6}	2
5	{5,7}	3
6	{4,5}	3

c) $V_1 \rightarrow V_7$: $V_1 \rightarrow V_3 \rightarrow V_6 \rightarrow V_7 = 9$
$V_1 \rightarrow V_2 \rightarrow V_5 \rightarrow V_7 = 9$

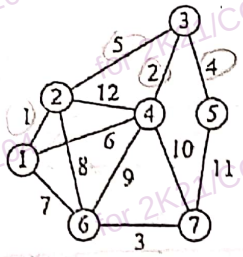
a) all spanning trees



etc.

4. Apply Prim's MST algorithm on following graph showing construction of MST at each step. Also specify how Prim's MST algorithm uses optimal substructure and greedy choice property.

2018 M



Date / /
Page No.

APPLYING PRIM'S ALGORITHM

In Prim's algorithm, we

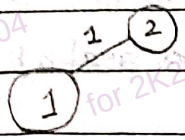
i) Select any vertex and choose the smallest weight from G.

ii) At each stage, choose the edge of smallest weight. Joining a vertex already included to vertex not yet included

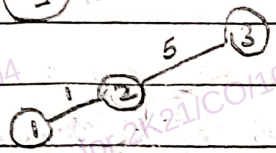
iii) Continue till all vertices are include

STEPS

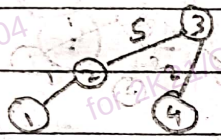
1)



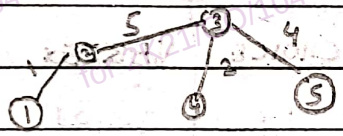
2)



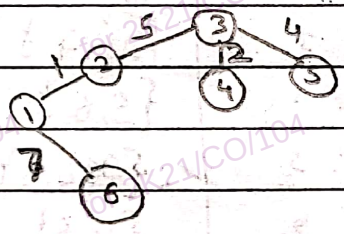
3)



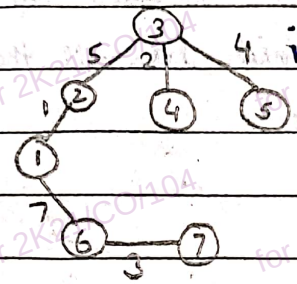
4)



5)



6)



CHOICE	EDGE	WEIGHT
1	{1,2}	1
2	{2,3}	5
3	{3,4}	2
4	{3,5}	4
5	{1,6}	7
6	{6,7}	3
7		

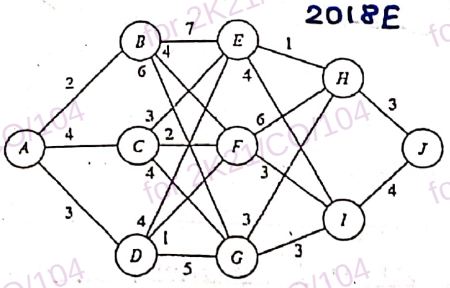
OPTIMAL SUBSTRUCTURE AND GREEDY CHOICE PROP.

1. **OPTIMAL SUBSTRUCTURE**: We can observe subtree of MST is also a MST. So if we have a connected graph with n vertices and m edges and we remove a single edge, we have 2 disjoint subgraphs. Any MST of the original graph must include the edges that connect these 2 SG. Therefore, we can build MST by incrementally adding edges one at a time to a growing subset of vertices.

2. **GREEDY CHOICE PROPERTY**: Here, we can choose the edge with the smallest weight that connects a vertex in the current MST to a vertex outside the MST. This is so cos we want to add edges such a way that total wt is minimised. By this we ensure that we are making the best possible choice at each step, which leads to the overall min wt MST

Q5. (a) Apply Kruskal's algorithm to find MST for given graph:

2018E



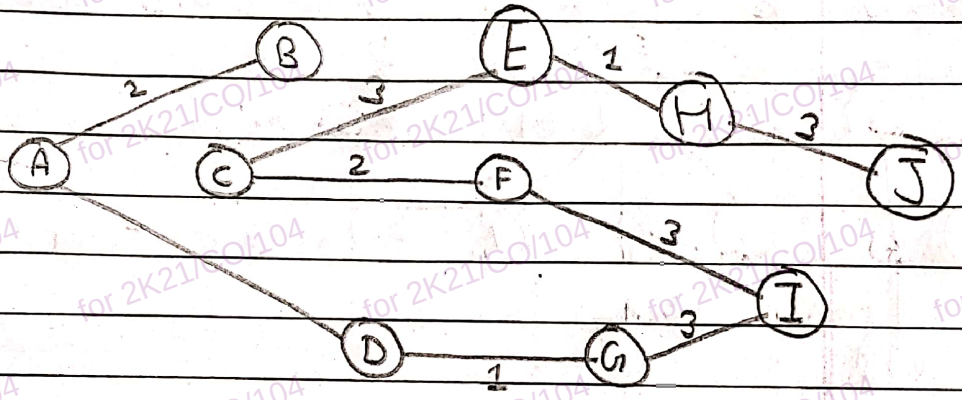
USING KRUSKAL'S ALGO

i) Choose an edge of minimal weight

ii) At each step, choose an edge whose inclusion will not create a circuit

iii) If G has n circuits, stop after (n-1) edges.

CHOICE	EDGE	WEIGHT
1	E, H	1
2	D, G	1
3	A, B	2
4	C, F	2
5	F, I	3
6	H, J	3
7	G, I	3
8	A, D	3
9	C, E	3
		21

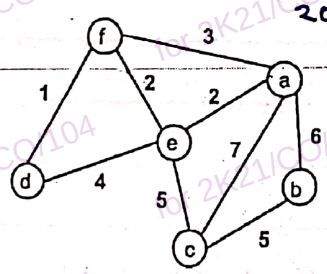


MST

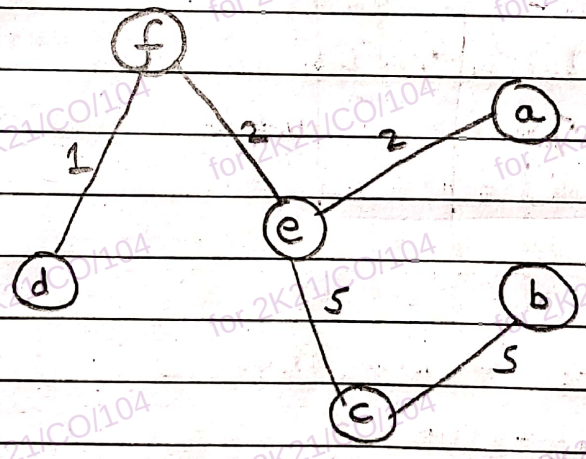
There can be multiple MSTs

Q6. (a) Apply Kruskal's algorithm to find MST for given graph:

2018S



CHOICE	EDGE	WEIGHT
1	{d, f}	1
2	{f, e}	2
3	{e, a}	2
4	{e, c}	5
5	{b, c}	5
		15



MST

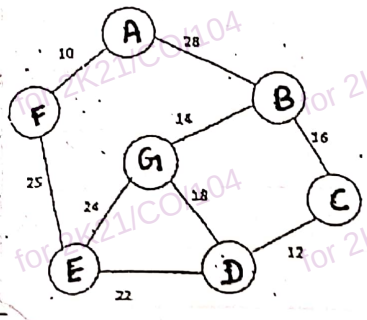
(b) Let $x = x_1x_2 \ldots x_n$ and $y = y_1y_2 \ldots y_m$ and $z = z_1z_2 \ldots z_{n+m}$ be three strings of length n, m, and n + m, respectively. We say that z is a merge of x and y if x and y can be found as two disjoint subsequences in z. Example: "algodatastrucrituthresms" is a merge of "algorithms" and "datastructures". For $0 \le I \le n$ and $0 \le J \le m$, Merge(I, J) is true if $z = z_1z_2 \ldots z_{I+J}$ is a merge of $x = x_1x_2 \ldots x_I$ and $y = y_1y_2 \ldots y_J$ ($x = x_1x_2 \ldots x_I$ is the empty string if I = 0. Similarly for y and z.). Define function Merge(I,J) and write code for this.

2019S

(5+5=10)

Q5 (a) Write down Dijkstra algorithm. Show step by step implementation of the same algorithm on the given graph with node 1 as the source node.

2018E



[7 Marks]

DIJKSTRA (G, w, s)
INITIALIZE-SINGLE-SOURCE (G, s)
S = ∅
Q = G.V
while Q ≠ 0
  u = EXTRACT-MIN (Q)
  S = S ∪ {u}
  for each vertex v ∈ G.Adj[u]
    RELAX (u, v, w)

RELAX (u, v, w)
if (d[u] + c(u,v) < d[v])
  d[v] = d[u] + c(u,v)

on a graph G = (V, E), w(u,v) ≥ 0, for each (u,v) ∈ E

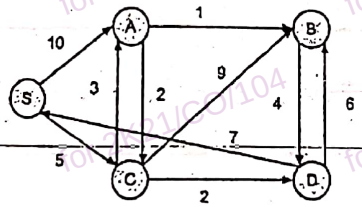
Dijkstra algorithm maintains a set S of vertices whose final shortest-path weights from source s have already been determined. The algorithm repeatedly selects the vertex u ∈ V−S with the minimum shortest-path estimate, adds u to S, and relaxes all edges leaving u. In the following implementation we use a min priority queue Q of vertices, keyed by their d values.

SELECTED VERTEX	A	B	C	D	E	F	G
A	(0)	∞	∞	∞	∞	∞	∞
F	(0)	28	∞	∞	∞	(10)	∞
B	(0)	(28)	∞	∞	35	(10)	∞
G	(0)	(28)	44	∞	(35)	(10)	42
E	(0)	(28)	44	57	(35)	(10)	(42)
C	(0)	(28)	(44)	57	(35)	(10)	(42)
D	(0)	(28)	(44)	(56)	(35)	(10)	(42)

ANS

4. Apply Dijakstra's algorithm to find Single source shortest path for following graph:

2020M



(5)

VERTEX	A	B	C	D	S
A	(0)	∞	∞	∞	∞
B	(0)	(1)	2	∞	∞
C	(0)	(1)	(2)	5	∞
D	(0)	(1)	(2)	(4)	∞
S	(0)	(1)	(2)	(4)	(11)

Starting from A shortest paths to respective edges are 0, 1, 2, 4, 11

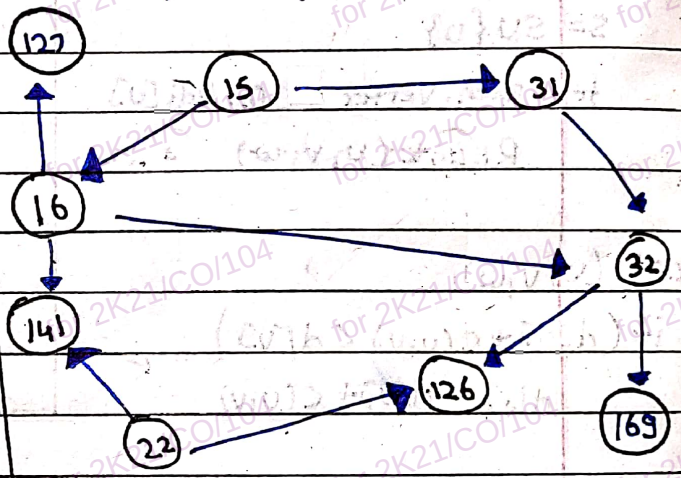
4. Bob loves foreign languages and wants to plan his course schedule to take the following nine language courses: LA15, LA16, LA22, LA31, LA32, LA126, LA127, LA141 and LA169. 2019M

The course prerequisites are: LA15: None, LA6: LA15, LA22: None, LA31: LA15, LA32: LA16 & LA31, LA126: LA22 & LA32, LA127: LA16, LA141: LA22 & LA16, LA169: LA32. Using Graphs, find a sequence of courses that allows Bob to satisfy all the prerequisites. Use proper algorithm to solve this problem. Also write the algorithm used. (5)

Date / /
Page No.

LA15: None
LA22: None
LA16: LA15
LA141: LA16, LA22
LA127: LA16
LA31: LA15
LA32: LA16, LA31
LA126: LA22, LA32
LA169: LA32

→ = pre-requisite.



Required DAG

USING TOPOLOGICAL SORT

1. Initialise a queue and a list to store the sorted nodes
2. Compute the indegree of each node
3. Enqueue all nodes with in degree 0
4. While Queue is not empty,
   a) Dequeue a node and add to sorted list
   b) for each neighbour of the dequeued node, decrement its in-degree. If the in-degree becomes 0, enqueue it.
5. If the sorted list contains all nodes, return it. Otherwise, there is a cycle in the graph, and no valid sequence exist

∴ ORDER

LA15 → LA16 → LA22 → LA31 → LA32 → LA126 → LA127 → LA141 → LA169

or

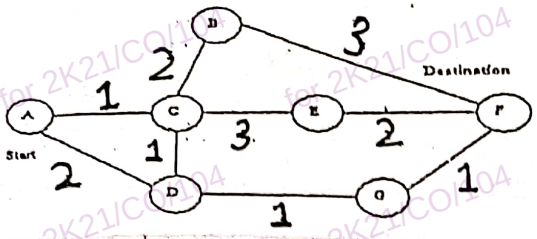
LA15 → LA31 → LA16 → LA32 → LA22 → LA141 → LA126 → LA127 → LA169 etc

So,

TOPOLOGICAL-SORT(G)

1. Call DFS(G) to compute finishing time f[v] for each edge
2. as each vertex is finished, insert it onto the front of LL.
3. return linked list of vertices

Q5 (a) Write down Dijkstra algorithm. Show step by step implementation of the same algorithm on the given graph with A as the source node. 2019E [5 Marks]

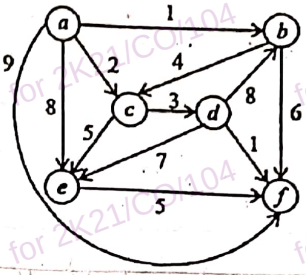


TAKING A AS SOURCE NODE by DIJKSTRA

	A	B	C	D	E	F	G
A	(0)	∞	∞	∞	∞	∞	∞
C	(0)	∞	(1)	2	∞	∞	∞
D	(0)	3	(1)	(2)	4	∞	∞
G	(0)	(3)	(1)	(2)	4	∞	3
B	(0)	(3)	(1)	(2)	4	6	(3)
G	(0)	(3)	(1)	(2)	(4)	4	(3)
E	(0)	(3)	(1)	(2)	(4)	(4)	(3)
F	(0)	(3)	(1)	(2)	(4)	(4)	(3)

∴ Shortest path is A→D→G→F with weight 4.

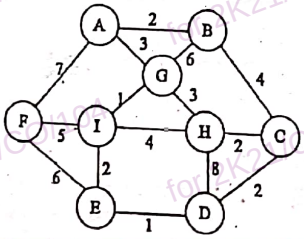
Q6. (a) Apply Dijakstra's algorithm to compute Single Source Shortest Path for vertex 'a' as source: 2019S



	a	b	c	d	e	f
a	(0)	∞	∞	∞	∞	∞
b	(0)	(1)	2	∞	8	9
c	(0)	(1)	(2)	∞	8	7
d	(0)	(1)	(2)	(5)	7	7
f	(0)	(1)	(2)	(5)	7	(6)
e	(0)	(1)	(2)	(5)	(7)	(6)

Shortest path from 'a' to other edges are 0, 1, 2, 5, 7, 6 for a, b, c, d, e, f resp.

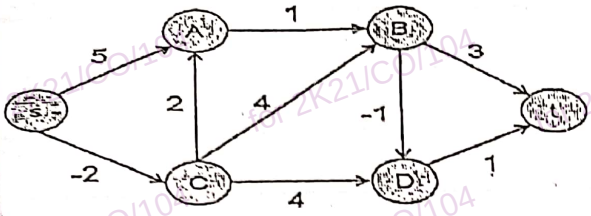
Q5 (a) Write down Dijkstra algorithm. Show step by step implementation of the same algorithm on the given graph with node A as the source node. 2018E



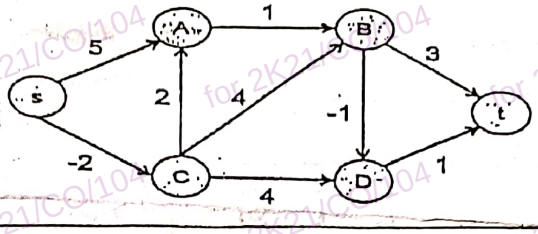
VERTICES	A	B	C	D	E	F	G	H	I
A	(0)	∞	∞	∞	∞	∞	∞	∞	∞
B	0	(2)	∞	∞	∞	7	3	∞	∞
G	(0)	(2)	6	∞	∞	7	(3)	∞	∞
I	(0)	(2)	6	∞	∞	7	(3)	6	(4)
C	(0)	(2)	(6)	∞	6	7	(3)	6	(4)
E	(0)	(2)	(6)	8	(6)	7	(3)	6	(4)
H	(0)	(2)	(6)	7	(6)	7	(3)	(6)	(4)
D	(0)	(2)	(6)	(7)	(6)	7	(3)	(6)	(4)
F	(0)	(2)	(6)	(7)	(6)	(7)	(3)	(6)	(4)

TAKING A as SOURCE and applying Dijkstra's algorithm, we find dist from A to A, B, C, D, E, F, G, H, I are 0, 2, 6, 7, 6, 7, 3, 6, 4 respectively.

(b) Write down Bellman ford algorithm. Show step by stem implementation of the same algorithm on the given graph 2018E

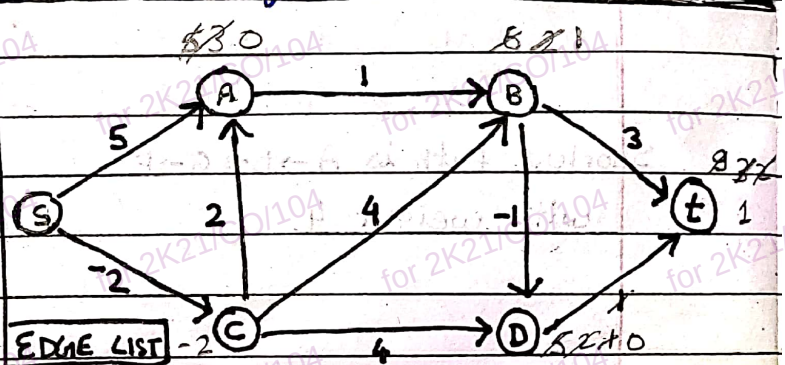


(b) Write down Bellman ford algorithm. Show step by stem implementation of the same algorithm on the given graph 2019E [5 marks]



Bellman ford algo. solves single-source shortest path problems in the general case in which edge weights may be negative. The algorithm uses relaxation, progressively decreasing an estimate d[v] on the weight of a shortest path from the source s to each vertex $v \in V$ until it achieves the actual shortest-path weight $\delta(s,v)$. The algorithm returns true iff the graph contains no-negative-weights cycles that are reachable from the source.

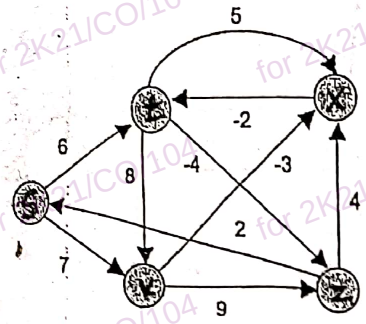
```
BELLMAN-FORD (G, w, s)
1  INITIALIZE-SINGLE-SOURCE (G, s)
2  for (i ← 1 to |V[G]| - 1)
3      do for each edge (u,v) ∈ E[G]
4          do RELAX (u,v,w)
5  for each edge (u,v) ∈ E[G]
6      do if d[v] > d[u] + w[u,v]
7          then return FALSE
8  return TRUE
```



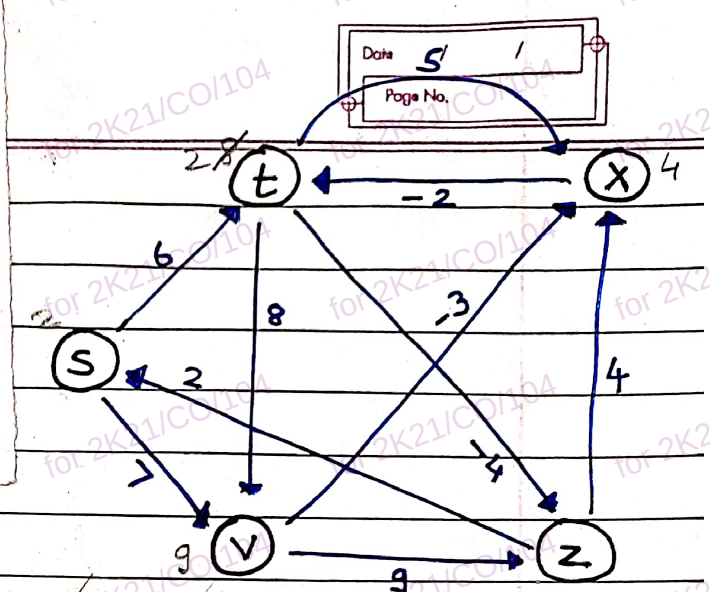
	S	A	B	C	D	t	
1	0	∞ 3	∞ 2	-2	∞ 2	∞ 3	
2	0	0	2	-2	1	2	
3	0	0	1	-2	0	1	
4	0	0	1	-2	0	1	
5	0	0	1	-2	0	1	← Ans

(b) Write down Bellman ford algorithm. Show step by stem implementation of the same algorithm on the given graph using vertex z as source vertex.

20185



[7 marks]



EDGE LIST

(z,s) (z,x) (s,t) (s,v) (t,v) (t,z) (t,x) (v,x) (v,z) (x,t)

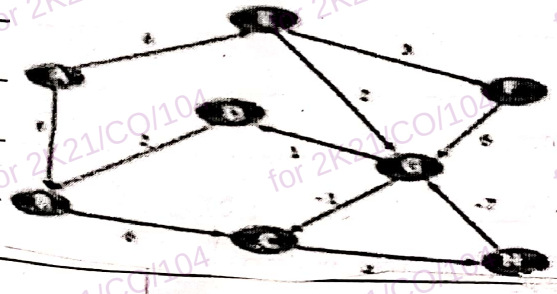
	z	s	t	v	x
1	0	2	2	9	4
2	-2	2	2	9	4
3	-4	0	0	7	2
4	-6	-2	-2	5	0

Source cycle.

NEGATIVE WEIGHT CYCLE EXIST → FAIL

Q3

a) Write down Bellman Fort algorithm. Show step by step implementation of the same algorithm on the given graph with A as the source node. 2016E (2+4)



UNCLEAR!!!

[b] Explain the greedy approach to solve the activity section problem. Give example. 2012E 8

```
Struct Activity {
    int start;
    int finish;
};
bool compare(Activity a, Activity b){
    return a.finish < b.finish;
}
```

Eg:) arr = {(1,2), (3,4), (0,6), (5,7), (5,9), (8,9)}
∴ OUTPUT = (1,2)
(3,4)
(5,7)
(8,9)

```
Void maxActivity(Activity arr[], int n){
    Sort(arr, arr+n, compare)
    int i=0; //keep track of last activity
//print(arr[i]) cout << arr[i].start << ", " << arr[i].finish << endl;
    for(int j=1; j<n; j++){
        if(arr[j].start >= arr[i].finish){
            print(arr[j]);
            i=j;
        }
    }
}
```

The activity selection problem involves selecting a maximum size set of mutually compatible activities that require use of a common resource. Each activity has a start and finish time, and the goal is to select activities in such a way that they do not overlap in time.

The greedy approach to solve this involves selecting the activity that has the earliest finish time, as it is likely to leave more time for selection of other activities

1. Sort the activities based on their finish times in ascending order
2. Select the first activity with the earliest finish time, and mark it as the current activity
3. For each remaining activity in list, if the start time of activity ≥ finish time of current activity, select that activity and set it as current activity
4. Repeat step 3 until no more activity left

b) Suppose you were to drive from St. Louis to Denver along I-70. Your gas tank, when full, holds enough gas to travel m miles, and you have a map that gives distances between gas stations along the route. Let $d_1 < d_2 < \ldots\ldots < d_n$ be the locations of all the gas stations along the route where d_i is the distance from St. Louis to the gas station. You can assume that the distance between neighboring gas stations is at most m miles. Your goal is to make as few gas stops as possible along the way. Apply greedy approach to solve the problem by listing the algorithm, optimal substructure, greedy choice property. Also discuss your algorithm with the help of example and complexity analysis? 2017 E (5)

Date / /

Page No.

(6+4=10)

Q2 (a) Describe efficient data structure for representation of collection of disjoint sets. Write Union() an Find() algorithm and their amortized cost. 2018 S

The disjoint set data structure represents efficient collection of disjoint sets. It uses the parent and rank arrays, where each element is initially the parent of itself with rank 0.

The find() operation recursively finds the root of the set containing an element and applies path compression.

The union() operation merges two sets by updating the parent array and maintaining the balance of the tree.

```
class DisjointSet {
    int *parent; int *rank;
    public:
        DisjointSet(int n) {
            parent = new int[n]; rank = new int[n];
            for (int i=0; i<n; i++) {
                parent[i] = i; rank[i] = 0;
            }
        }
int find(int v) {
    if (v == parent[v])
        return v;
    int rr = find(parent[v]);
    parent[v] = rr; //path compression
    return rr;
}
```

```
void union(int v1, int v2) {
    int re1 = find(v1);
    int re2 = find(v2);

    if (re1 != re2) {
        if (rank[re1] < rank[re2])
            parent[re1] = re2;
        else if (rank[re1] > rank[re2])
            parent[re2] = re1;
        else {
            parent[re2] = re1;
            rank[re2] += 1;
        }
    }
}
```

(b) You are given a sequence of n songs where the i^{th} song is t_i minutes long. You want to place all of the songs on an ordered series of CDs (e.g. CD_1, CD_2, CD_3, ... CD_k) where each CD can hold m minutes. Furthermore,

(1) The songs must be recorded in the given order, $song_1$, $song_2$, ..., $song_n$.

(2) All songs must be included.

2019S

(3) No song may be split across CDs. Your goal is to determine how to place them on the CDs as to minimize the number of CDs needed. Give the most efficient algorithm you can to find an optimal solution for this problem and analyze the time complexity (5+5=10)

Date / /
Page No.

We will use the greedy solution.
→ Put song 1 on CD 1
→ for song i, if space is left on current CD, put it here otherwise move to next CD

→ if no CD left then no soln

CODE:

```
FUNC (int n, int m, arr[])
    int cd_count = 1; i = 1;
    capacity_left = m
    while (i < n) {
        if (arr[i] < capacity_left)
            capacity_left -= arr[i];
        else
            cd_count++;
            capacity_left = m;
        i++;
    }
    Print (cd_count)
}
```

TIME COMPLEXITY → O(n)

Date / /
Page No.

ASYMPTOTIC ANALYSIS

1.) **Big Oh (O) : WORST CASE**

$f(n) \leq c \cdot g(n) \quad \forall \; n \geq n_0 ; c > 0$ then $f(n) = O(g(n))$

2.) **Omega (Ω) : BEST CASE**

$f(n) \geq c \cdot g(n) \quad \forall \; n \geq n_0 ; c > 0$ then $f(n) = \Omega(g(n))$

3.) **Average (Θ) : THETA CASE**

$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \quad \forall \; n \geq n_0$ and $c > 0$ then $f(n) = \Theta(g(n))$

SIEVE OF ERATHOSTHENISE (prime)

FAST EXPO

```
Pow (x,n) {
   if (n == 0) return 1;
   sp = pow (x, n/2);
   if (n%2 == 0) return sp*sp;
   else          return sp*sp*x;
}
```

MASTER'S THEOREM

$T(n) = aT(n/b) + n^k \log^p n$

and $a \geq 1 ; b > 1 ; k \geq 0 ; p$ = Real

CASE-1.) If $a > b^k$: $T(n) = \Theta(n^{\log_b a})$

CASE-2.) If $a = b^k$: a.) If $p > -1 \rightarrow T(n) = O(n^{\log} \log^{p+1} n)$

b.) If $p = -1 \rightarrow T(n) = \Theta(n^{\log} \log(\log(n)))$

c.) If $p < -1 \rightarrow T(n) = \Theta(n^{\log})$

CASE-3.) If $a < b^k$: a.) If $p \geq 0 \rightarrow T(n) = \Theta(n^k \log^p n)$

b.) If $p < 0 \rightarrow T(n) = \Theta(n^k)$

Date / /
Page No.

DIVIDE AND CONQUER

MERGE SORT

```
MERGE 2 SORTED ARR (arr[], lo, mid, hi) {
    n1 = mid - lo + 1 ;  n2 = hi - mid ;
    int * arr1 = new int [n1] ; int * arr2 = new int [n2] ;
    for (i = 0 → n1-1)  arr1[i] = arr [lo+i] ;
    for (i = 0 → n2-1)  arr2[i] = arr [mid+i+1] ;
    int a=0 ; b=0 ; c=lo ;
    while (a<n1 and b<n2)
        if (arr1[a] < arr2[b]) arr[c++] = arr1[a++] ;
        elseif (arr[a] > arr2[b]) arr[c++] = arr2[b++] ;
    while (a<n1)  arr[c++] = arr1[a++] ;
    while (b<n2)  arr[c++] = arr2[b++] ;
}
MERGE SORT (arr[], lo, hi)
    if (lo >= hi) return ;
    int mid = (lo+hi)/2 ;
    merge sort (arr, lo, mid) ;
    merge sort (arr, mid+1, hi) ;
    merge 2 sorted arr (arr, lo, mid, hi) ;
}
```

QUICK SORT

```
QUICKSORT (arr[], lo, hi)
    if (lo >= hi) return ;
    mid = (lo+hi)/2 ; piv = arr[mid] , left = lo, right = hi
    while (left ≤ right) {
        while (arr[left] < pivot) left++ ;
        while (arr[right] > pivot) right-- ;
        if (left ≤ right)
            swap (arr[left], arr[right]) ;
            left++ , right-- ;
    } }
    QUICK SORT (arr, lo, right)
    Quick SORT (arr, left, hi)
}
```

(OR)

```
QS (Arr[], l, r)
  if (l < r) {
    int pi = partition (arr, l, r)
QS (arr, l, p-1)  QS (arr, p+1, r)
}}
Partition (arr, l, r) {
  pivot = arr[r]
  i = l-1
  for (j = l → r-1)
    if (arr[j] < pivot)
      i++ ;
      swap (i, j)
  swap [i+1 , r)
  return (i+1)
}
```

BINARY SEARCH

```
BINARY SEARCH (arr[], n, x) {
    int lo=0, hi=n-1, mid=0;
    while (lo<=hi)
        mid=(lo+hi)/2;
        if (arr[mid]<x) lo=mid+1;
        else if (arr[mid]>x) hi=m-1;
        else return mid;
    } return -1;
}
```

BOOK ALLOCATION

```
Book (pages[], nos, nob)
int lo=0; hi=0; for (auto val : pages) hi+=val;
int res=0;
while (lo<=hi) {
    int mid = (lo+hi)/2;
    if (possible (pages[], nob, nos, mid) {
        res=mid;
        hi=mid-1;
    } else
        lo=mid+1
} return res;
POSSIBLE (pages[], nob, nos, max)
int students=1;
int pagesread=0; int i=0
while (i<nob) {
    if (pagesRead+pages[i]<=max) {
        pagesRead+=pages[i++];
    else {
        students++; pagesRead=0;
        if (students>nos) return false
    }
} return true }
```

1) AGGRESSIVE COWS

```
COW (Stalls[], noc, nos) {
    int lo=0;
    int hi=Stalls[nos-1]-Stalls[0];
    int res=0;
    while (lo<=hi) {
        mid=(lo+hi)/2
        if (possible (noc, nos, Stalls, mid))
            lo=mid+1
            res=mid
        } else
            hi=mid-1
} return res;
POSSIBLE (noc, nos, Stalls[], minDistance) {
    int cowplaced=1;
    int lastPos=Stalls[0];
    for (i=1→nos-1) {
        if (Stalls[i]-lastPos>=minDist)
            cowplaced++;
            lastPos=Stalls[i];
    } if (cowsPlaced==noc)
        return true;
    } return false;
}
```

Date / /
Page No.

HEAP SORT

```
MAX-HEAPIFY (A, n, i) {
    int largest = i ;
    int l = 2 x i ;
    int r = (2 x i) + 1 ;
    while ( l ≤ n and A[l] > A[largest])
                largest = l ;
    while ( r ≤ n and A[r] > A[largest])
                largest = r ;
    if ( largest != i ) {
            swap ( A[largest], A[i])
            heapify (A, n, largest) ;
    }
}
HEAP-SORT (A, n) {
    for (i = n/2 ⟶ 1)
            max-heapify (A, n, i)
    for (i = n ⟶ 1) {
            swap ( A[1], A[i])
            Max-heapify (A, n, 1)
    }
}
```

GREEDY

a) **OPTIMAL SUBSTRUCTURE :**
Solutions to subproblems of an optimal solution are optimal

b) **GREEDY CHOICE PROPERTY :**
A global optimum solution can be arrived by making a locally optimal solution.

FRACTIONAL KNAPSACK

```
int frac (Item arr[], int n, int cap) {
    sort (arr, arr+n, compare) ; //dec
    double totalProfit = 0.0;
    for (i=0 → n-1) {
        if (arr[i].weight <= cap) {
            totalProfit += arr[i].price;
            cap -= arr[i].weight;
        } else {
            totalProfit += arr[i].price * (cap / arr[i].weight)
            break
        }
    } return totalProfit;
}
```

ACTIVITY SELECTION

```
Void maxActivity (Activity arr[], int n) {
        Sort (arr, arr+n, compare); //Asc of finish time
        int i=0; //track of last activity
        print (arr[i].start, arr[i].finish);
        for (j=1 → n-1) {
            if (arr[j].start >= arr[i].finish) {
                print (" j ", j) //Select.
                i=j;          //change for the last act. now
            }
        }
}
```

Date / /
Page No.

HUFFMAN

```
void huff (char arr[], int freq[], int n) {
    pq <Node*, vector<Node*>, compare> pq;
    for (i=0 → n-1)  pq.push (new Node (arr[i], freq[i]));
    while (pq.size()!=1)
    {
        Node* fn = pq.top(); pq.pop();
        Node* sn = pq.top(); pq.pop();        } remove least freq 2 nodes.
        Node* nn = new Node ('-', fn→freq + sn→freq);
        nn→left = fn;
        nn→right = sn;                        } create nn and link fn, sn
        pq.push(nn); push back in pq.
    }
    Node* ln = pq.top();  pq.pop();
    Output codes ;
}
```

PRIMS

```
Void prims (){
    pq <        > pq;
    bool visited[v]; fill(visited, visited+V, false);
    while (pq.size()!=0){
        Prims Node *rp = pq.top(); pq.pop();
        if (visited[rp→vname]) continue;  visited[rp→vname] = true;
        if (rp→acqvname!=-1)  print ()
        map<int, int>::iterator itr;
        for (itr = strg[rp→vname].begin(); itr != strg[rp→vname.end(); itr++)
                                                                    {
            int nbr = itr→first; int cost = itr→second;
            if (!visited[nbr])
                pq.push(new PrimsNode(nbr, rp→vname, cost));
        }
    }
}
```

KRUSKAL

```
Void Kruskal ()
{
    Sort (edgeList.begin(), edgeList.end(), compare);
    DisjointSet ds(V);
    for (auto edge : edgeList) {
        int u = edge[0];        int re-u = ds.find(u);
        int v = edge[1];        int re-v = ds.find(v);
        int cost = edge[2];
                                if (re-u != re-v) {
                                    ds.unite(u, v);
                                    print (u, v @ cost);
                                }
    }
}
```

DIJKSTRA

```
void dijkstra (int src) {
    pq <        > pq;
    pq.push (new DijkstraNode (src, to-string (src), 0));
    bool visited[V]; fill (visited, visited + V, false);
    while (pq.size() != 0) {
        DijkstraNode *rp = pq.top();   pq.pop(); // remove
        if (visited[rp->vname]) continue; // ignore if already visited
        visited[rp->vname] = true;   // visited
        Print ("rp->name' via 'rp->path' @ 'rp->cost")
        map <int, int>:: iterator itr;
        for (itr = Strg[rp->vname].begin(); itr != strg(rp->vname).end(); itr++)
            int nbr = itr->first;
            int cost = itr->second;
            if (! visited[nbr])
                pq.push (new DijkstraNode (nbr, rp->path
                    + to-string (nbr), rp->cost + cost));
        }
    }
}
```

Date / /
Page No.

BELLMAN FORD

```
void bellmanFord (int src) {
    int cost [v];
    fill (cost, cost+V, 100000);
    cost[src] = 0;
    for (i = 1 → V) {
        for (auto edge : edgeList) {
            int u = edge [0];
            int v = edge [1];
            int c = edge [2];
            int oc = cost [v];          // relax edge
            int nc = cost [u] + c;
            if (nc < oc) {
                if (i <= V-1)
                    cost [v] = nc;
                else {
                    cout << "-ve present" << endl;
                    return;
                }
            }
        }
    }
    for (i = 0 → V-1)
        cout << i << " → " << cost [i] << endl;
}
```

STRASSEN

$P = (A_{11}+A_{22}).(B_{11}+B_{22})$	$C_{11} = P+S-T+V$ PSNTV
$Q = (A_{21}+A_{22}).B_{11}$	$C_{12} = RT$ RT
$R = A_{11}(B_{12}-B_{22})$	$C_{21} = Q+S$ QS
$S = A_{22}(B_{21}-B_{11})$	$C_{22} = P+R-Q+V$ PRQV
$T = (A_{11}+A_{12}).B_{22}$	
$U = (A_{21}-A_{11}).(B_{11}+B_{12})$	
$V = (A_{12}-A_{22}).(B_{21}+B_{22})$	

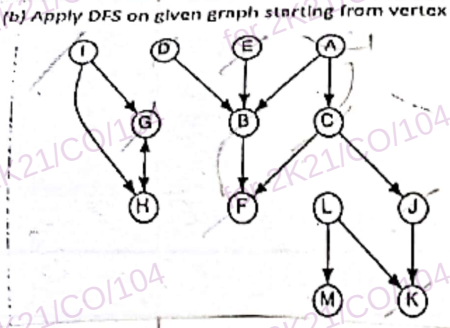
```
Void BellmanFord (Graph G, int s) {
    queue Q;
    int v, w;
    enqueue (Q, s);
    Distance [s] = 0;
    while (! isEmpty (Q)) {
        V = deQueue(Q);
        for all adjacent vertices w of v {
            compute new dist d = Distance [v] + weight [v][w];
            if (old dist to w > new distance d) {
                Distance [v] = (distance to v) + weight [v][w];
                path [w] = v;
                if (w is there in Q)
                    enqueue (Q, w)
            }
        }
    }
}

Void Kroskals (struct Graph* G) {
    S = ϕ
    for (int v=0; V < G->V ; V++)
        MakeSet (V);
    Sort edges of E by increasing value of weights.
    for each edge (U,V) in E {
        if (FIND (U) ≠ FIND (V)) {
            S = SU {(U,V)}
            UNION(U,V);
        }
    }
    return S
}
```

```
void Prims (struct Graph* G, int s) {
    pq PQ;
    int v, w;
    enqueue (PQ, s)
    Distance [s] = 0;
    while (! Empty (PQ)) {
        v = deleteMin (PQ);
        for (all adjacent vertices w of v) {
            compute new dist d = Distance [v] + weight [v][w];
            if (Distance [w] == -1) {
                Distance = weight [v][w];
                insert w in pq of priority d.
            }
        }
    }
```

(b) What are types of edges you encounter when you run DFS on a directed graph. 2018S

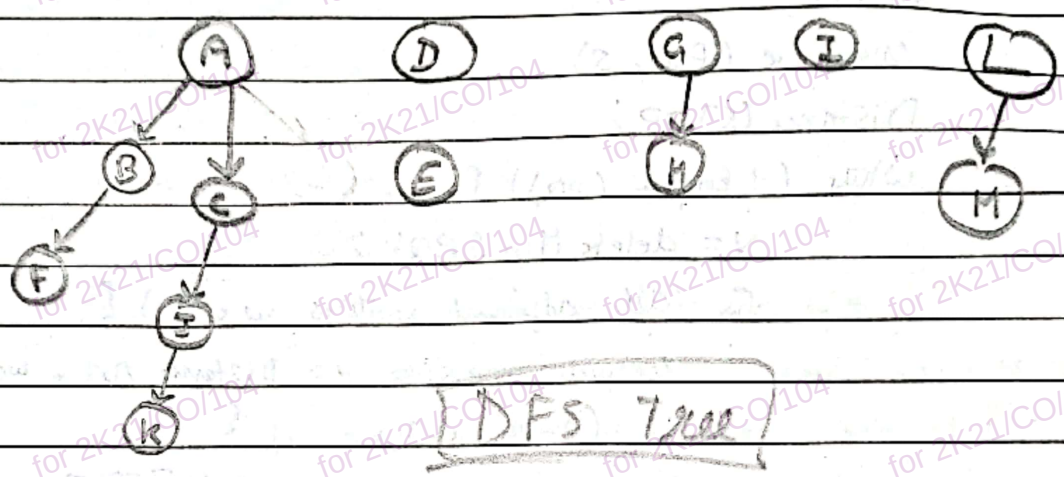
(5+5=10)

(b) Apply DFS on given graph starting from vertex A and draw DFS Tree.



2018(2)

Date / /
Page No.



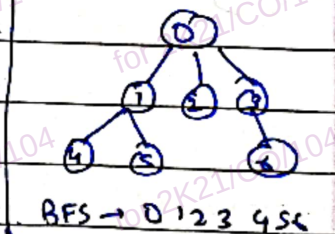
DFS Tree

[e] How does traversal occur in breadth first search.

2012E

In BFS, traversal occurs in a level-wise manner, where nodes are visited in a level-wise manner, where nodes are visited in order of their distance from the source node.

The traversal process begins at the source node and explores all of its neighbours. After visiting all of its neighbours, BFS moves on to the next level of nodes, which are neighbours of the previously visited nodes. This process continues until all nodes have been visited.



To keep track of the order in which nodes are visited, BFS typically uses a queue data structure.

Date / /
Page No.

UNIT-4 (Dynamic Programming)

Q7.(a) Explain the characteristics of a problem that can be solved efficiently using Dynamic programming technique. 2019E [5 Marks]

[b] Define principle of optimality 2012E

(b) Solve following 0/1 Knapsack problem using dynamic programming for Knapsack capacity W=16.

Item no.	weight	Value
1	4	40
2	7	42
3	5	25
4	3	12

(b) Solve following 0/1 Knapsack problem using Dynamic Programming algorithm:

Item	1	2	3	4
Weight	4	7	5	3
Value	40	42	25	12

20185

2018E

[7 marks]

USING MEMOISATION

```
int KnapsackTD(int[] weight, int[] Value, int vidx, int cap, int[][] dp) {
    if (vidx == weight.length() || cap == 0)    return 0;
    if (dp[vidx][cap] != 0)                     return dp[vidx][cap];
    //exclude
    int e = KnapsackTD (weight, Value, vidx+1, cap, dp);
    // include
    int i = 0
    if (cap >= weight[vidx])
        i = KnapsackTD (weight, price, vidx+1, cap - weight[vidx], dp) + value[vidx];

    int ans = Max (e, i);
    return dp[vidx][cap] = ans ;
}
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	42	42	42	42	42	42	42	42	42	42
2	0	0	0	0	0	25	25	42	42	42	42	42	67	67	67	67	67
1	0	0	0	12	12	25	25	42	42	42	54	54	67	67	67	79	79
0	0	0	0	12	12	25	25	42	42	42	54	54	67	67	67	79	79

Ans → 79

(b) Solve following 0/1 Knapsack problem using dynamic programming for Knapsack capacity W=5.

20185

Item no.	weight	Value
1	2	12
2	1	10
3	3	20
4	2	15

	0	1	2	3	4	5
0	0	12	17	27	32	37
1	0	10	15	25	30	35
2	0	10	10	20	30	30
3	0	10	10	10	10	10
4	0	0	0	0	0	0

Ans → 37

(c) Solve following 0/1 Knapsack problem using Dynamic Programming algorithm for m=25:

2019S

Item	1	2	3	4
Weight	9	8	12	14
Value	10	12	14	16

Date / /
Page No.

USING BOTTOM UP / TABULATION

```
int Knapsack BU (int wt [], int price [], int n, int cap) {
int Strg [] [] = new int [n+1] [cap+1];
for (int row = n-1; row >= 0; row --) {
    for (int col = 1; col <= cap; col++) {
        int exclude = Strg [row+1] [col];
        int include = 0; if (col >= wt [row]) {
            include = Strg [row+1] [col - wt [row]] + price [row];
        int ans = max (exclude, include); Strg [row] [col] = ans;
    } }
Return Strg [0] [cap];
```

wt	val	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
9	10	0	0	0	0	0	0	0	0	0	12	12	12	14	14	14	16	16	16	21	21	21	26	26	28	28	28	30
8	12	1	0	0	0	0	0	0	0	0	12	12	12	12	14	14	16	16	16	16	16	16	26	26	28	28	30	30
12	14	2	0	0	0	0	0	0	0	0	0	0	0	0	14	14	16	16	16	16	16	16	16	16	16	16	30	30
14	16	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	16	16	16	16	16	16	16	16	16	16	16
-	-	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Ans

Q4. (a) Solve following 0/1 Knapsack problem using branch and bound technique:

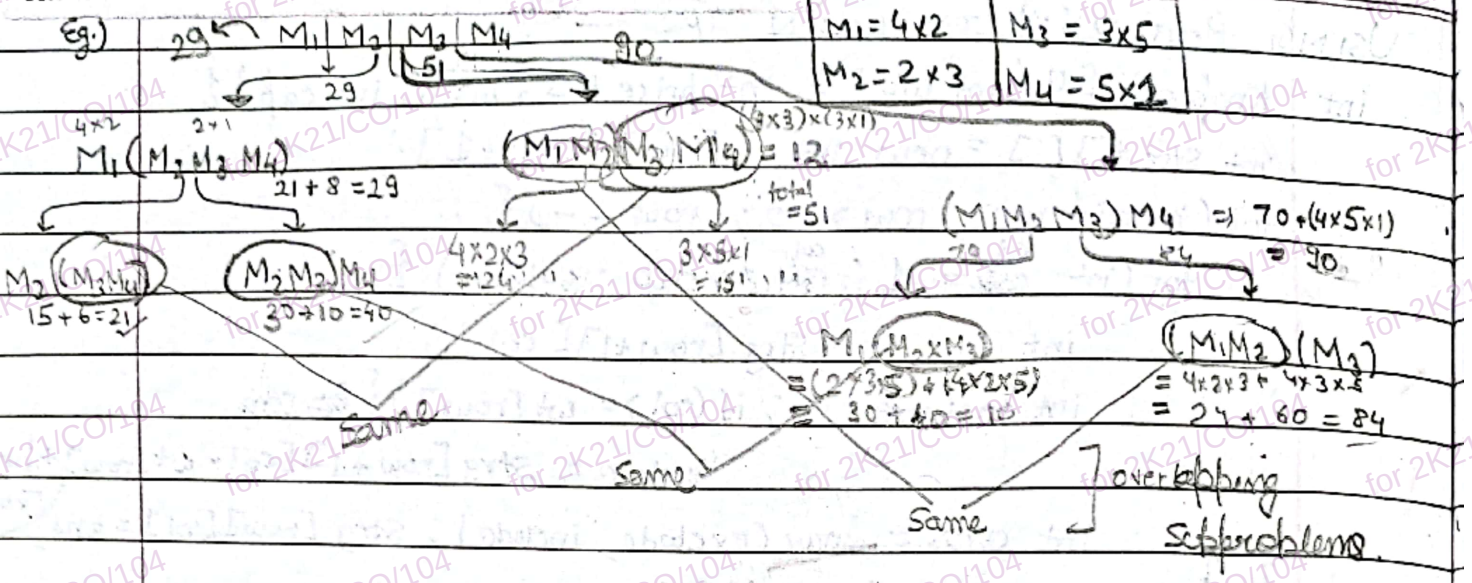
2019S

Item	1	2	3	4	5
Weight	2	3	1	5	3
Value	40	50	100	95	30

W=10

b) Discuss optimal substructure and overlapping sub problem in detail w.r.t Matrix Chain Multiplication problem. Also write the memoized solution for the same. 2016E (4+3)

$M_1 = 4 \times 2$	$M_3 = 3 \times 5$
$M_2 = 2 \times 3$	$M_4 = 5 \times 1$



MEMOISATION CODE:

```
int MCMTD (int arr[], int si, int ei, int **strg) {
if (si+1 = ei) return 0;   // base case if only one matrix
if (strg [si][ei] != 0) return strg [si][ei];   // re-use
int mini = INT_MIN;
for (int k = si+1 ; k <= ei-1 ; k++) {   // K-splits.
    int fp = MCMTD (arr, si, k, strg);   // arr[si] * arr[k]
    int sp = MCMTD (arr, k, ei , strg);   // arr [k] x arr [ei]

    int sw = arr [si] * arr [k] * arr [ei];   // Self work
    int total = fp + sp + sw;

    if (total < min)  mini = total;
}
strg [si] [ei] = mini ;   // Storage.
return mum;
}
```

Q2 (a) The Longest Increasing Subsequence (LIS) problem is to find the length of the longest subsequence of a given sequence such that all elements of the subsequence are sorted in increasing order. For example, the length of LIS for {10, 22, 9, 33, 21, 50, 41, 60, 80} is 6 and LIS is {10, 22, 33, 50, 60, 80}.

2018E

arr[]	10	22	9	33	21	50	41	60	80
LIS	1	2		3		4		5	6

Apply Dynamic programming to solve such problem. Mention the pseudo-code, complexity, Overlapping sub problems and optimal sub structure with respect to the problem.

Date / /
Page No.

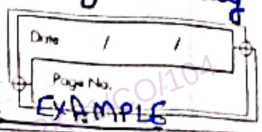
TABULATION

```
int LISBU (int n, int *arr) {
    int *le = new int(n);
    le[0] = arr[0]; int len=1;
    for (int i=1; i<n; i++) {
        if (arr[i] > le[len-1]) {
            le[len] = arr[i];
            len++;
        } else {
            int idx = binarySearch(le, 0, len-1, arr[i]);
            le[idx] = arr[i];
        }
        cout << arr[i] << ":" ;
        for (auto val : le)
            cout << val << " ";
        cout << endl;
    } return len;
}
```

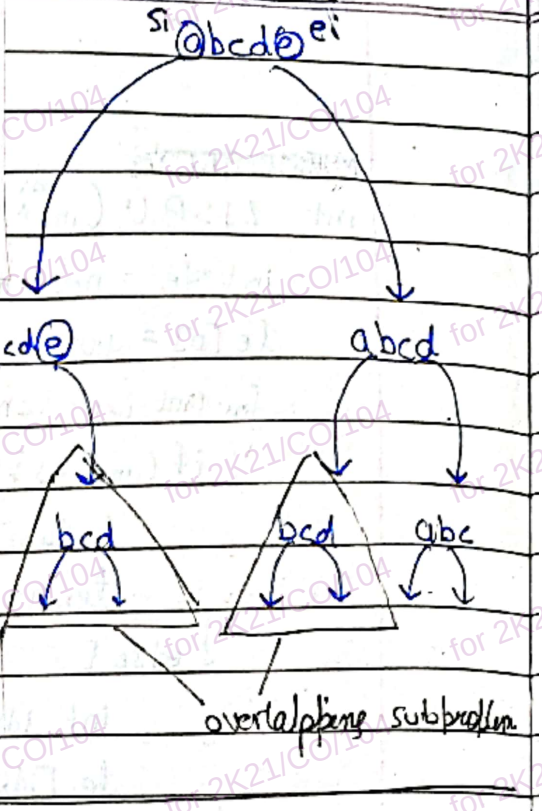
Q3 Given a problem of finding Longest Palindrome Subsequence. Find the optimal sub-structure to obtain the same, explain with the help of example. Also show graphically, with an example, how recursive implementation of this optimal sub-structure will lead to overlapping sub problems. (A sequence is a palindrome if it reads the same whether we read it left to right or right to left. For example A, C, G, G, G, G, A. Longest Palindrome Subsequence example: consider the string A, G, C, T, C, B, M, A, A, C, T, G, G, A, M has many palindrome as subsequence, for instance: A, G, T, C, M, C, T, G, A has length 9).

2017M [2.5 + 2.5]

we break storing virtually.



CONSIDER EXAMPLE




```
int LPSSRec (String str, int si, int ei){
if (si > ei)  return 0;
if (si == ei)  return 1;
int ans = 0;
if (str[si] == str[ei]) {
    ans = LPSSRec (str, si+1, ei-1) + 2;
} else {
    int o1 = LPSSRec (str, si+1, ei);
    int o2 = LPSSRec (str, si, ei-1);
    ans = max (o1, o2);
}
return ans;
```

a ---- a

viii) S1 = ABCBDABCD | S2 = BACADBCAAA

		B	A	C	A	D	B	C	A	A	–
	A	5	5	4	4	3	3	2	2	2	1	0
	B	5	4	4	3	3	3	2	1	1	1	0
	C	4	4	4	3	3	2	2	1	1	1	0
	B	4	3	3	3	3	2	1	1	1	1	0
	D	3	3	3	3	3	2	1	1	1	1	0
	A	3	3	3	3	2	2	1	1	1	1	0
	B	3	2	2	2	2	2	1	0	0	0	0
	C	2	2	2	1	1	1	1	0	0	0	0
	D	1	1	1	1	1	0	0	0	0	0	0
	–	0	0	0	0	0	0	0	0	0	0	0

A C D B A

4[a] Given two sequences S = ABAZDC and T=BACBAD, find the maximum length common subsequence of S and T using dynamic programming approach. 2012E 8

(b) Write Longest Common subsequence Algorithm. Apply the same to find the Longest Common subsequence for input Sequences "AGGTAB" and "GXTXAYB". 2016E (2+3)

Q3. (a) Determine LCS of <A,B,B,A,B,A,B,A> and <B,A,B,A,A,B,A,A,B>using dynamic programming. Write algorithm and runtime complexity for the same? [5 marks]

Q3. (a) Determine LCS of <A,B,C,B,D,B,A> and <B,D,C,A,B,A>using dynamic programming. Write algorithm and runtime complexity for the same? 2018E old [7 marks]

(b). Compute LCS for following set of sequences: X = <A, B, C, B, D, A, B, C, D> and Y = <B, A, C, A, D, B, C, A, A, A> using dynamic programming algorithm. Show mathematical formulation of the given problem using dynamic programming. 2018E

(b). Compute LCS for following set of sequences: X = "AGGTAB" and Y="GXTXAYB" using dynamic programming algorithm. Show mathematical formulation of the given problem using dynamic programming. 2018S

(b). Compute LCS for following set of sequences: X = "PMJYAUZ" and Y="MZJAWPU" using dynamic programming algorithm. Show mathematical formulation of the given problem using dynamic programming. 2019S (5+5=10)

Q3. (a) Determine LCS of <A,C,G,G,T,T,A> and <C,G,T,A,T>using dynamic programming. Write algorithm and runtime complexity for the same? 2018S [7 marks]

MEMOISATION

```
int LCSTD(string S1, string S2, int vidx1, int vidx2, int** strg){
    if(vidx1==S1.length() || vidx2 == S2.length)
        return 0;
    if (strg[vidx1][vidx2]!= -1)
        return strg[vidx1][vidx2];
    int ans;
    if (S1[vidx1] == S2[vidx2])
        ans = LCSTD(S1, S2, vidx1+1, vidx2+1, strg) +1;
    else{
        int o1 = LCSTD(S1, S2, vidx1, vidx2+1, strg);
        int o2 = LCSTD(S1, S2, vidx1+1, vidx2, strg);
        ans = max(o1, o2);
    }strg[vidx1][vidx2] = ans;
    return ans;
}
```

TABULATION

```
int LCSBU(string S1, string S2){
    int row = S1.length(); col = S2.length();
    int strg[row+1][col+1];
    for (int j=0; j<=col; j++)
        strg[row][j]=0;
    for (int i=0; i<=row; i++)
        strg[i][col]=0;
    for(int i=row-1; i>=0; i--){
        for(int j=col-1; j>=0; j--){
            if (S1[i]==S2[j])
                strg[i][j] = strg[i+1][j+1]+1;
            else
                strg[i][j]= max(strg[i][j+1], strg[i+1][j]);
        }
    }
}
```

T.C. = O(m×n)

i) S1 = ABAZDC | S2 = BACBAD

		B	A	C	B	A	D	
	A	4	4	3	3	2	1	0
	B	4	3	3	3	2	1	0
	A	3	3	2	2	2	1	0
	Z	2	2	2	1	1	1	0
	D	2	2	2	1	1	1	0
	C	1	1	1	1	1	1	0
		0	0	0	0	0	0	0

A B D

Date / /
Page No.

ii) S1 = AGGTAB | S2 = GXTXAYB

		G	X	T	X	A	Y	B	
	4	3	3	2	2	1	1	0
A	4	3	3	2	2	1	1	0
G	4	3	3	2	2	1	1	0
G	4	3	3	2	2	1	1	0
T	3	3	3	2	2	1	1	0
A	2	2	2	2	2	1	1	0
B	1	1	1	1	1	1	1	0
	0	0	0	0	0	0	0	0

G T A B

iii) S1 = ABBABABA | S2 = BABAABAB

		B	A	B	A	A	B	A	B	0
	6	5	5	4	4	3	3	2	1	0
A	6	5	5	4	4	3	3	2	1	0
B	6	5	5	4	4	3	3	2	1	0
B	6	5	5	4	4	3	3	2	1	0
A	5	5	4	4	4	3	3	2	1	0
B	4	4	4	3	3	3	2	2	1	0
A	3	3	3	3	3	2	2	2	1	0
B	2	2	2	2	2	2	1	1	1	0
A	1	1	1	1	1	1	1	1	0	0
	0	0	0	0	0	0	0	0	0	0

A B A A B

iv) S1 = ABCBDBA | S2 = BDCABA

		B	D	C	A	B	A	–
	4	3	3	3	2	1	0
A	4	3	3	3	2	1	0
B	4	3	3	2	2	1	0
C	4	3	3	2	2	1	0
B	4	3	2	2	2	1	0
D	3	3	2	2	2	1	0
B	2	2	2	2	2	1	0
A	1	1	1	1	1	1	0
	0	0	0	0	0	0	0

B C B A

v) S1 = AGGGTTA | S2 = CGTAT

		C	G	T	A	T	–
	4	3	2	2	1	0
A	4	3	2	2	1	0
C	4	3	2	1	1	0
G	3	3	2	1	1	0
G	3	3	2	1	1	0
T	2	2	2	1	1	0
T	2	2	2	1	1	0
A	1	1	1	1	0	0
–	0	0	0	0	0	0

C G T T

vi) S1 = AGGTAB | S2 = GXTXAYB

		G	X	T	X	A	Y	B	–
	4	3	3	2	2	1	1	0
A	4	3	3	2	2	1	1	0
G	4	3	3	2	2	1	1	0
G	4	3	3	2	2	1	1	0
T	3	3	3	2	2	1	1	0
A	2	2	2	2	2	1	1	0
B	1	1	1	1	1	1	1	0
–	0	0	0	0	0	0	0	0

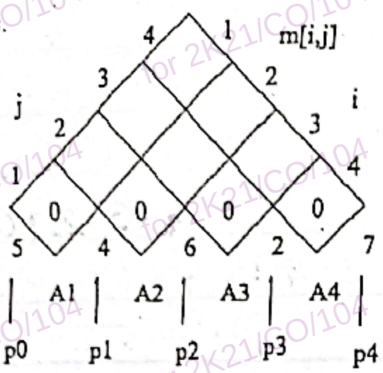
G T A B

vii) S1 = PMJYUAX | S2 = MZJAWPU

		M	Z	J	A	W	P	U	–
	3	2	2	2	2	2	1	0
P	3	2	2	2	2	2	1	0
M	3	2	2	1	1	1	1	0
J	2	2	2	1	1	1	1	0
Y	1	1	1	1	1	1	1	0
U	1	1	1	1	1	1	1	0
A	1	1	1	1	0	0	0	0
X	0	0	0	0	0	0	0	0
–	0	0	0	0	0	0	0	0

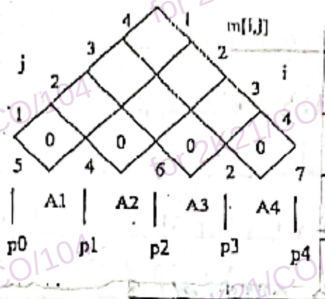
MJU

iven a chain of four matrices A_1, A_2, A_3, and A_4, with dimensions (15x4), (4x6), (6x12), and 2x5) respectively. Fill the following table in bottom-up fashion and give solution to the oblem:



2018S

(b) Given a chain of four matrices A_1, A_2, A_3, and A_4, with dimensions (5x4), (4x6), (6x2), and (2x7) respectively. Fill the following table in bottom-up fashion and give solution to the problem:



(5+5=10)

Date / /

Page No.

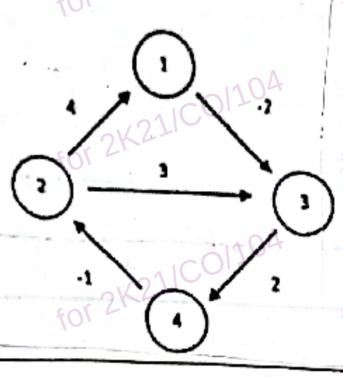
2018E

Q3

a) Write down Floyd Warshall algorithm and its complexity. Show step by step implementation of the same algorithm on the given graph. 2017E

6[a] Illustrate an algorithm that efficiently solves all pair shortest path problem. 2012E 7

(b) Apply Floyd Warshall all pair shortest path algorithm on following graph: 2018E



ALGORITHM

1. Initialize the distance matrix
   i) create 2D array of size n×n n → # vertices
   ii) Set diagonal dist[i][i] = 0
   iii) for each edge [i][j], set dist[i][j] and dist[j][i] to given value

2. Update distance matrix
   i) for each vertex k, consider all pairs of (i,j) and update dist as:
      i) if dist[i][j] through k is shorter than curr then update it as dist[i][k] + dist[k][j].
      ii) otherwise skip.

3. Return dist matrix with APSP

Q5 (=) For bit strings $X = x_1 \ldots x_m$, $Y = y_1 \ldots y_n$ and $Z = z_1 \ldots z_{m+n}$, we say that Z is an interleaving of X and Y if it can be obtained by interleaving the bits in X and Y in a way that maintains the left-to-right order of the bits in X and Y. For example if X = 101 and Y = 01 then $x_1x_2y_1x_3y_2$ = 10011 is an interleaving of X and Y, whereas 11010 is not. Give the most efficient algorithm you can to determine if Z is an interleaving of X and Y. Prove your algorithm is correct and analyze its time complexity as a function m and n.

2018E

Let $C[i,j]$ be true if $z_1 \ldots z_{i+j}$ is an interleaving of $X = x_1 \ldots x_i$, $Y = y_1 \ldots y_j$

Now subproblem is recursively:

$$C[i,j] = \begin{cases} \text{true} & \text{if } i=j=0 \\ \text{false} & \text{if } x_i \neq z_{i+j} \text{ and } y_j \neq z_{i+j} \\ C[i-1,j] & \text{if } x_i = z_{i+j} \text{ and } y_j \neq z_{i+j} \\ C[i,j-1] & \text{if } x_i \neq z_{i+j} \text{ and } y_j = z_{i+j} \\ C[i-1,j] \vee C[i,j-1] & \text{if } x_i = z_{i+j} \text{ and } y_j = z_{i+j} \end{cases}$$

∴ $C[m,n]$ contains answer.

ALGORITHM

```
STRING-INTERLEAVING (X[1...m], Y[1...n], Z[1...p])
C[0...m, 0...n] is a 2-dimentional array (default entry : false)
if m=n=p=0
    return true
if p ≠ m+n
    return false
if m==0 and Y==Z
    return true
else
    return false
if n==0 and X==Z
    return true
else
    return false
C[0,0] = true;
for (i=0 → n)
    for (j=1 → n)
        C[i,j] = (X[i]==Z[i+j] ∧ C[i-1,j]) ∨ (Y[j]==Z[i+j] ∧ C[i,j-1])
return C[m,n]
```

PROOF:

CASE 1: if i=j=0, both X and Y are empty and then by def^n Z is also empty and therefore, Z is valid.

CASE 2: if $x_i \neq z_{i+j}$ and $y_j \neq z_{i+j}$, then there is no interleaving possible and therefore return false.

CASE 3: if $x_i = z_{i+j}$ and $z_{i+j} \neq y_i$ then there exists a valid interleaving where x_i appears last iff $C[i-1,j]$.

CASE 4: Symmetric to case 3.

CASE 5: when both case 3 and 4 are satisfied, we can find a valid interleaving of X and Y, by extending either the third or fourth case.

Clearly, optimal substructure holds, because a correct interleaving of X and Y must contain in it correct interleaving of subproblem of Y_i and Y_j, for $0 \leq i \leq m$ and $0 \leq j \leq n$.

RUNTIME

$O(m \cdot n)$.

Q1 (a) How is the dynamic programming solution of the travelling salesman problem, better than the naïve algorithm? Discuss with the help of example. Also derive and compare their complexity? 2017E (5)

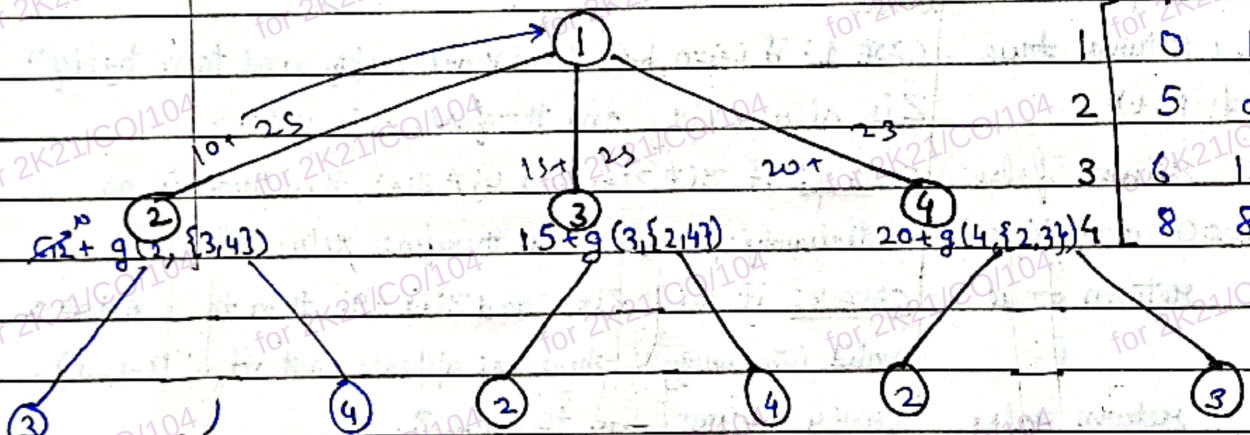
The naive algorithm for TSP involves trying every possible permutation of choice of cities to find shortest route. For n cities there are n! possible permutations, which is unfeasable for even moderately sized problems. Thus, it has a TC of $O(n!)$.

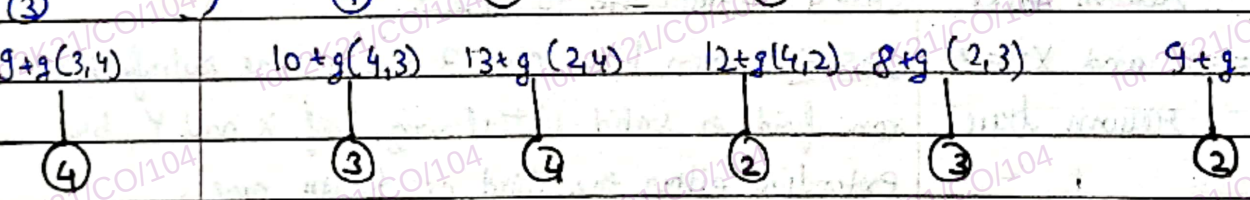
The DP solution involves a bottom up approach to build up the solution by solving smaller subproblems and combining to solve larger problem. This increases efficiency with a complexity of $O(n^2 * 2^n)$ which is better than naive algo.

(b) Solve travelling salesman by applying dynamic programming for the given graph? 2019E [5 marks]

Eg: Using DP: $g(i,S) = \min_{k \in S} \{C_{ik} + g(k, S-\{k\})\}$

	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0





12+g(4,ϕ)	9+g(3,ϕ)	10+g(4,ϕ)	8+g(2,ϕ)	9+g(3,ϕ)	13+g(2,ϕ)
= 20	= 15	= 20	= 13	= 15	= 18

∴ It can be written as.

g(2, ϕ) = 5	g(2,{3}) = 15	g(2,{3,4}) = 25
g(3, ϕ) = 6	g(2,{4}) = 18	g(3,{2,4}) = 25
g(4, ϕ) = 8	g(3,{2}) = 18	g(4,{2,3}) = (23)
	g(3,{4}) = 20	
	g(4,{2}) = 13	g(1,{2,3,4}) = 35
	g(4,{3}) = 15	

g(-,ϕ) = g(-,1)

(b) Consider a two team game where team A and B, play a series of games until one of the teams win 'n' games. Assume that the probability of team A winning a game is 'p', and losing it is q=(1-p). We also assume that there is no tie in a game. Let p(i,j) be the probability of team A winning the series if A needs 'i' more games to win the series and B needs 'j' more games to win the series. After setting a recurrence relation for p (i, j), write the pseudocode for solving this problem using dynamic programming. What is the complexity of the algorithm. 2017E (2+2+1)

Date / /
Page No.

The question will lead to this recurrance relation.

$$P(i,j) = pP(i-1,j) + qP(i,j-1) \quad \text{for all } i,j > 0$$

base conditions,

$$P(0,j) = 1 \ \forall\ j>0 \quad | \quad P(i,0) = 0 \ \forall\ i>0$$

PSEUDO CODE

```
WORLD SERIES (n,p){
    q = 1-p;
    for(int j = 1 to i = n) {
        P[0,j] = 1 ;
    }for ( int i = 1 to i = n) {
        P[i,0] = 0 ;
        for (int j = 1 to j = n) {
            P[i,j] = p * [P[i-1,j]] + q * [P[i,j-1]] ;
        }
    } return P[n,n];
}
```

using rec table.

COMPLEXITY

SC = $O(n^2)$ | TC = (n^2)

∴ table of (n+1) × (n+1) is computed

b) Consider the below given recursive algorithm for calculating factorial of any integer n.

```
int fact( int n){
if(n==1)
        return 1;
else if(n==0) return 1;
else
    return n * fact(n-1);
}
```

First analyze the time complexity for the given algorithm. Then modify the same algorithm to calculate the series 1! + 2! + ... + n! when n is fed as the input without disturbing its divide and conquer nature. Now transform the modified algorithm to make it a bottom up dynamic programming algorithm. Compare the two algorithms in terms of complexity and overlapping sub problems. (1+1+1+3).

The function fact (n) has a TC of O(n) as it gives n recursive calls.

MODIFIED ALGO

```
int sumfact (int n) {
    if(n==1) return 1; if(n==0) return 1;
    else{
        temp = fact(n-1) x n
        return temp + sumfact (n-1);
    }
```

Date / /
Page No.

int DP[n+1]

BOTTOMUP (diff way)

```
int SumfactDP (int n) {
    DP[0] = 1
    for (int i=0; i<=n; i++)
        DP[i] = i * DP[i-1];
    int sum = 0;
    for (int i=0; i<=n; i++)
        sum += DP[i];
    return sum;
}
```

Overlapping Subproblems

we had to compute factorials of all preve terms for each term in the sequence which is avoided using DP

TC: old → O(n!)
new → O(n)

[b] Given a sequence of 3 matrices A_1, A_2, A_3 whose dimensions are $P_0=3$, $P_1=1$ $P_2=2$, $P_3=1$. what are the m array and S array for these inputs. Also construct optimal parenthesization. 2012E 7

(MCM) dp=m

here, (with 1 based indexing)

```
int MCMBU (int arr[], int n) {
    int dp[n][n];
    for (int slide = 1 → n-1) {
        for (int si = 0 → n-slide-1) {
            int ei = si + slide;
            if ((si+1) == ei) dp[si][ei] = 0;
            else {
                int min = INT_MAX;
                for (k = si+1 → ei-1) {
                    int fp = dp[si][k];
                    int sp = dp[k][ei];
                    int sw = arr[si] * arr[k] * arr[ei];
                    int total = fp + sp + sw;
                    if (total < min) min = total;
                }
                Strg[si][ei] = min;
            }
        }
    }
    return Strg[0][n-1];
}
```

m(1,1) = m(2,2) = m(3,3) = 0

m(1,2) = $p_0 p_1 p_2$ = 6

m(2,3) = $p_1 p_2 p_3$ = 2

∴ m(1,3) = min{ m(1,1) + m(2,3) + $p_0 p_1 p_3$, m(1,2) + m(3,3) + $p_0 p_2 p_3$ }

= min{0+2+3, 6+0+6}

m(1,3) = 5

S ARRAY → stores optimal cost of k giving m(i,j) for (S(i,j))

S	1	2	3
1	-	1	1
2	-	-	2
3	-	-	-

m	1	2	3
1	0	6	5
2	-	0	2
3	-	-	0

b) Define backtracking phenomenon? Write the pseudo code for solving n-queen problem with the help of back tracking 2016E (2+3)

06

Q4: (a) Define backtracking phenomenon? Write the pseudo code for solving n-queen problem with the help of back tracking 2019E [5 marks]

Backtracking is a common problem solving technique which is used to explore all possible solutions by incrementally building a solution and discarding it if its determined not to find the solution and proceeding to a different path.
It is done by backtracking to the last choice point and trying a different solution. It continues until a solution is found or all possible choices have been explored.

N-QUEENS : Queen's perspective.

```
bool isItSafe(bool board[][n], int row, int col){
    int r=row-1; int c=col; // vertically up
    while(r>=0){
        if(board[r][c]==1) return false;
        r--;
    }
    r=row-1; c=col-1; // diagonally left
    while(r>=0 && c>=0){
        if(board[r][c]==1) return false;
        r--; c--;
    }
    r=row-1; c=col+1; // diagonally right
    while(r>=0 && c<n){
        if(board[r][c]==1) return false;
        r--; c++
    }
    return true;
}
```

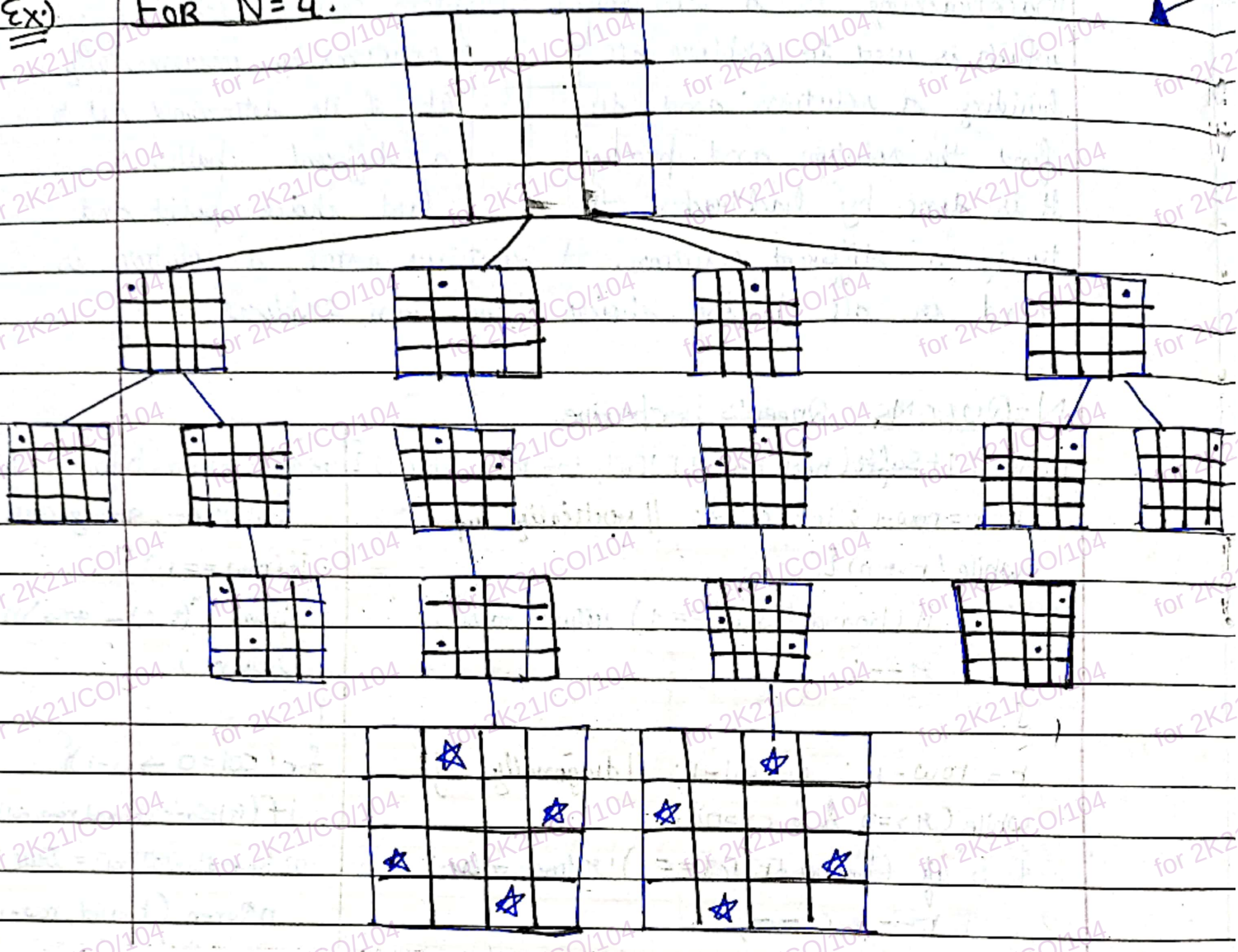
```
void nQueen(bool board[][n],
    int row, string ans){
    if(row==n){
    print("("+c)- ans\n");
    return;
    }
    for(col=0->n-1){
    if(isItSafe(board,row,col)){
    board[row][col]=true;
    nQueen(board,row+1
    ,ans+'{'+row+'-'+col
    +'}');
    board[row][col]=false;
    }
    }
```

Q7 (a) Write algorithm for solving 8-queen's problem. 2018E

8-queens problem is a specific instance of N queens' problem with N=8. It is infact specially about placing 8 queens on a chessboard of 8x8 size.

3[a] What is back tracking approach of problem solving? Write the backtracking algorithm for solution to well known n-queen problem with example. 2012E 7

Ex) FOR N=4:



(10)

Q4. (a) What is a backtracking? Give the explicit and implicit constraints in 8 queen's problem 2018S

EXPLICIT

$S_i = \{1, 2, 3, \ldots 8\}$ $1 \le i \le 8$. The soln space consists of 8^8-8tuples

IMPLICIT

It is that no two x_i's can be the same (as queens must be on different columns) and not two queens can be on same diagonal.

This implies that all solutions are permutations of 8-tuple (1,2,...8) and reduces the soln space from 8^8 to 8! tuples

j) Define backtracking phenomenon? Cut the state space for 4- queen problem by applying back tracking? Also mention how you are going to take care of diagonal constraint in its implementation?
2017E (2+2+1)

Diagonal constraints are

Q4. (a) Define backtracking phenomenon? Write the pseudo code for solving subset sum problem with the help of back tracking
2018E(odd) [7 marks]

SUBSET SUM PROBLEM

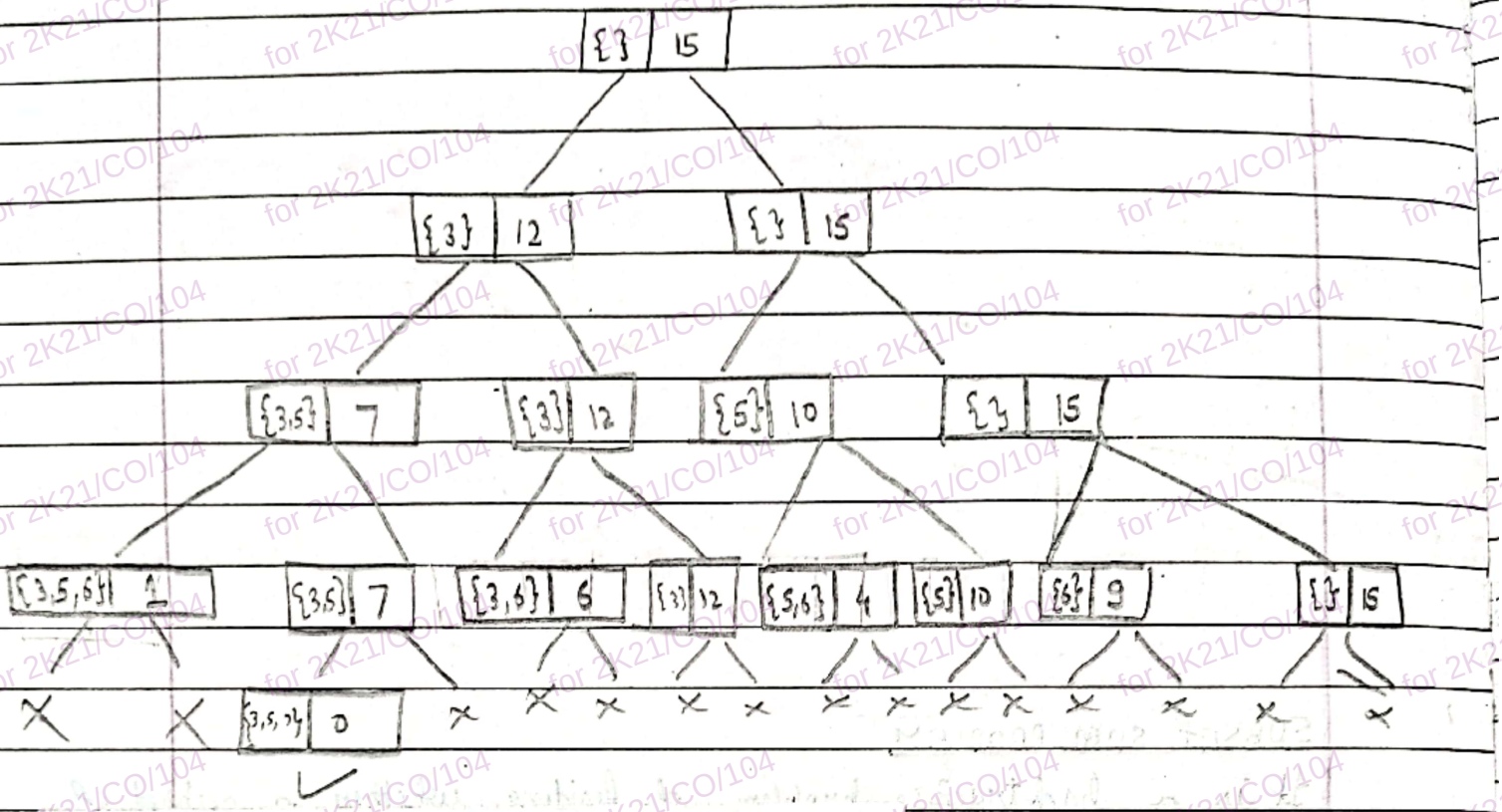
It is a backtracking problem of finding whether a subset of a given set of integers has a sum equal to given target number

```
void SubsetSum(vector<int>& S, vector<int>& path, int target, int start, int sum)
{   if (sum == target)
        // print the path in vector path
    if (sum > target || start = S.size())
        return ; // no path found.
    path.push_back(S[start]);                    // include
    subsetsum (S, path, target, start+1, sum + S[start]);
    }
    path.pop_back();                             // exclude
    SubsetSum (S, path, target, start+1, sum);
}
```

Q.8. (a) Solve the following subset problem using backtracking. $S = (3, 5, 6, 7)$ and $d = 15$. 2019S

Q.8. (a) Solve the following subset problem using backtracking. S={3,5,6,7}, d=15. 2018S

Given, $S = \{3, 5, 6, 7\}$ $d = 15$.



(b) What is a Hamiltonian Cycle? Explain how to find Hamiltonian path and cycle using backtracking algorithm. 2018S (5+5=10)

A hamiltonian cycle is cycle / path in a graph which passes through through every vertex exactly once, forming a closed path and returns to its starting point.

```
bool hamiltonian (int src, int curr, int path[], int idx) {
    if (idx == V) {
        if (dp[src].count(curr) != 0) {
            for (int i = 0 → V-1)
                cout << path[i] << " ";
            cout << endl;
            return true;
        } return false;
    }

    int ans = false;
    map<int, int> :: iterator itr;

    for (itr = dp[curr].begin(); itr != dp[curr].end(); itr++) {
        int nbr = itr → first;
        if (isItSafe(path, nbr)) {
            path[idx] = nbr;
            ans = hamiltonian(src, nbr, path, idx+1) || ans;
            path[idx] = -1;
        }
    } return ans;
}
```

UNIT 5: Branch and Bound

(b) Define the following: [5 marks] 2019E
1. FIFO Branch and Bound
2. LIFO Branch and Bound
3. Least cost search

1) FIFO BB is a strategy used in BB algorithms to explore the tree such that nodes are explored in the order they are added to the queue. The algorithm starts with an initial node and then generates child nodes from it. ~~These child nodes are~~ These child nodes are then added to a queue, and the next node to be explored is popped from queue.

This approach uses the BFS method to implement the algorithm using a queue data structure

2. LIFO BB is a strategy in which nodes are explored in reverse order. they were added in the queue. The algorithm starts from an initial node and then generates child nodes from it. These nodes are pushed to a stack, and the next node to be explored is is popped out from stack.

This approach uses the DFS method to implement the algorithm using a stack data structure.

3. Least Cost Search. is an algorithm used to find minimal path in graph to a goal node based on least cost. It starts with an initial node and examines all its adjacent nodes to find the node with least cost.
The algorithm then generates child nodes from selected node and adds them to a priority queue based on their cost. The priority queue ensures that the node with the least cost are explored first

(b) Differentiate between "backtracking" and "branch and bound" strategies.

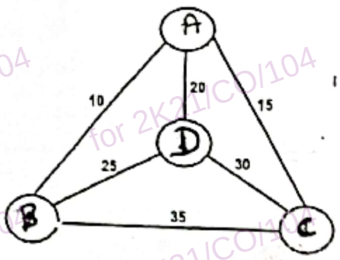
2019S (5+5=10)

Date / /
Page No.

BACKTRACKING	BRANCH AND BOUND
1. In Backtracking, DFS is used for tracing the solution for the given problem.	In Branch and Bound, BFS is used for tracing the solution for the given problem.
2. Typically decision problems can be solved using backtracking.	Typically optimization problems can be solved using branch and bound.
3. While finding the solutions, wrong choices may be made in process.	It proceeds only on optimal or better solution.
4. The state space tree is searched until the solution is obtained.	The SST needs to be searched completely as optimal soln may be anywhere
Eg) It is used to solve: N-Queen's Problem Graph Coloring Problem Hamiltonian cycle problem	It is applications includs 0-1 knapsack Problem Travelling Salesman Problem.

(b) Solve travelling salesman by applying branch and bound for the given graph?

2018S (old)



[7 marks]

Matrix

	A	B	C	D	
A	∞	10	15	20	10
B	10	∞	35	25	10
C	15	35	∞	30	15
D	20	25	30	∞	20 ⇒ 55

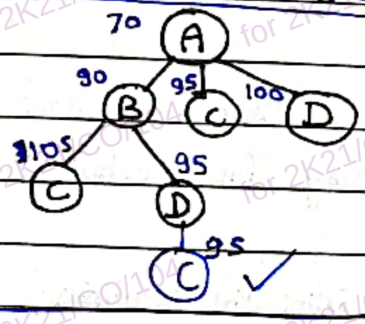
after row reduction

	A	B	C	D	
A	∞	0	5	10	
B	0	∞	25	15	
C	0	20	∞	15	
D	0	5	10	∞	
	0	0	5	10 → 15	

after column redⁿ

	A	B	C	D
A	∞	0	0	0
B	0	∞	20	5
C	0	20	∞	5
D	0	5	5	∞

parent Matrix (70)



for A → B

	A	B	C	D	
A	∞	∞	∞	∞	
B	∞	∞	20	5	
C	0	∞	∞	5	
D	0	∞	5	∞	

redⁿ ⇒

∞	∞	∞	∞	
∞	∞	10	0	
0	∞	∞	5	
0	∞	0	∞	

Reduction = 5 + 5 = 10

∴ T.C = 70 + 10 + 10 = 90

for A → B → C

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	∞	∞
C	∞	∞	∞	5
D	0	∞	∞	∞

Redⁿ = 5

∞	∞	∞	∞	
∞	∞	∞	∞	
∞	∞	∞	0	
0	∞	∞	∞	

T.C = 90 + 10 + 5 = 105

for A → B → D

	A	B	C	D
A	∞	∞	∞	∞
B	∞	∞	∞	∞
C	0	∞	∞	∞
D	0	∞	0	∞

T.C = 90 + 0 + 5 = 95 ✓

for A → C Redⁿ = 10

	A	B	C	D	
A	∞	∞	∞	∞	
B	0	∞	∞	5	
C	∞	20	∞	5	5
D	0	5	∞	∞	
		5			

→

∞	∞	∞	∞	
0	∞	∞	5	
∞	15	∞	0	
0	0	∞	∞	

T.C = 70 + 10 + 15 = 95

for A → D Redⁿ = 10

	A	B	C	D	
A	∞	∞	∞	∞	
B	0	∞	20	∞	
C	0	20	∞	∞	
D	0	5	5	∞	
		5	5		

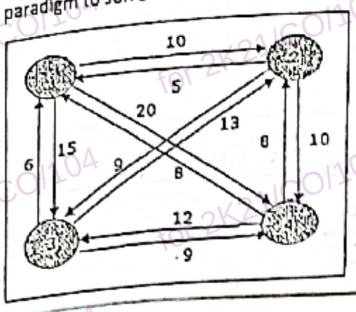
→

∞	∞	∞	∞	
0	∞	15	∞	
0	15	∞	∞	
0	0	0	∞	

T.C = 70 + 10 + 20 = 100

Cost = 95 A → B → D → C → A

Q3 A traveler needs to visit all the cities from a list, where distances between all the cities are known and each city should be visited just once. What is the shortest possible route that he visits each city exactly once and returns to the origin city? Apply branch and bound paradigm to solve TSP for graph given below. 2018E

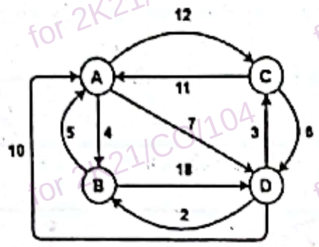


(10)

Date / /
Page No.

Q3. A traveler needs to visit all the cities from a list, where distances between all the cities are known and each city should be visited just once. What is the shortest possible route that he visits each city exactly once and returns to the origin city? Apply branch and bound paradigm to solve TSP for graph given below.

20195

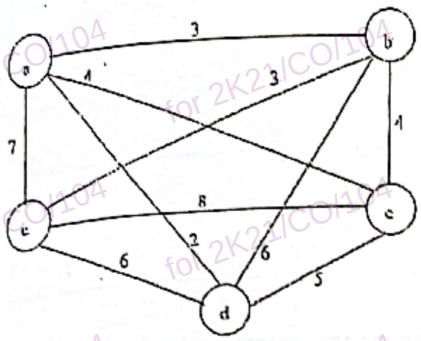


(10)

Date / /
Page No.

Solve travelling salesman by applying branch and bound for the given graph?

2018E (old)



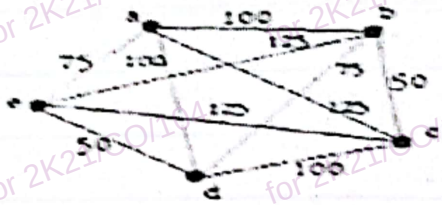
[7 marks]

Date / /

Page No.

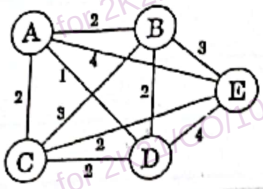
Q2 a) State and describe Travelling salesman problem. Solve TSP using branch and bound by constructing state space diagram for the given graph.

2016E (2-7)



Q3. A traveler needs to visit all the cities from a list, where distances between all the cities are known and each city should be visited just once. What is the shortest possible route that he visits each city exactly once and returns to the origin city? Apply branch and bound paradigm to solve TSP for graph given below.

2018S



Date : / /

Page No.

a) What is LC branch and bound? Apply LC search on famous 0/1 knapsack problem for the given instance of the problem. Capacity of knapsack is 12 kg? (2+3)

2017E

Item	Weight	Value
1	4	$10
2	6	$15
3	3	$6
4	5	$8
5	2	$4

12 kg

Date : / /
Page No.

$P_1P_2P_3P_4P_5 = (10, 15, 6, 8, 4)$
$w_1w_2w_3w_4w_5 = (4, 6, 3, 5, 2)$

(1) $U = 10 + 15$

2

2016E

Q5 a) Draw the full state space for 4-queen problem. Explain each of the below mentioned terminologies and then draw the new state space for the same problem in each of the mentioned case. Don't bother about the feasibility of bound function. A bound function for each case is given and just show its effect if it would have been there. (3X3 = 9)

(i) LIFO branch and bound, For an node Y which is child of node X let the bound function is maximization function and the bound value of any Y is always greater than the bound value its corresponding X

(ii) FIFO branch and bound, For an node x it could be the level number of node X

2016E

(iii) LC branch and bound, For any node X it could be the number of levels the nearest answer node (in the sub tree X) is from X

Date : / /

Page No.

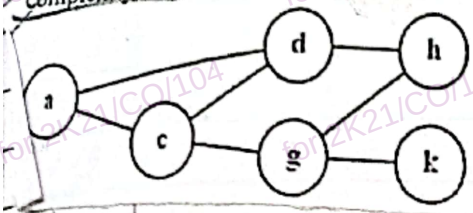
b) Describe vertex cover problem? Prove that the vertex cover problem is NP-complete. Also design an approximate solution for the same? (2+5+3)

(vii) Vertex Cover Problem 2018E

Q7. (a) Prove that vertex cover problem is NP-complete. 2019S

[b] Explain the vertex cover problem and travelling sales man problem. 2012E 8

b) Write down an approximate solution for vertex cover problem? Also apply the same on the given graph and also explain its complexity? (1+2+1)



2017E

UNIT 6: Computational Complexity

VERTEX COVER PROBLEM

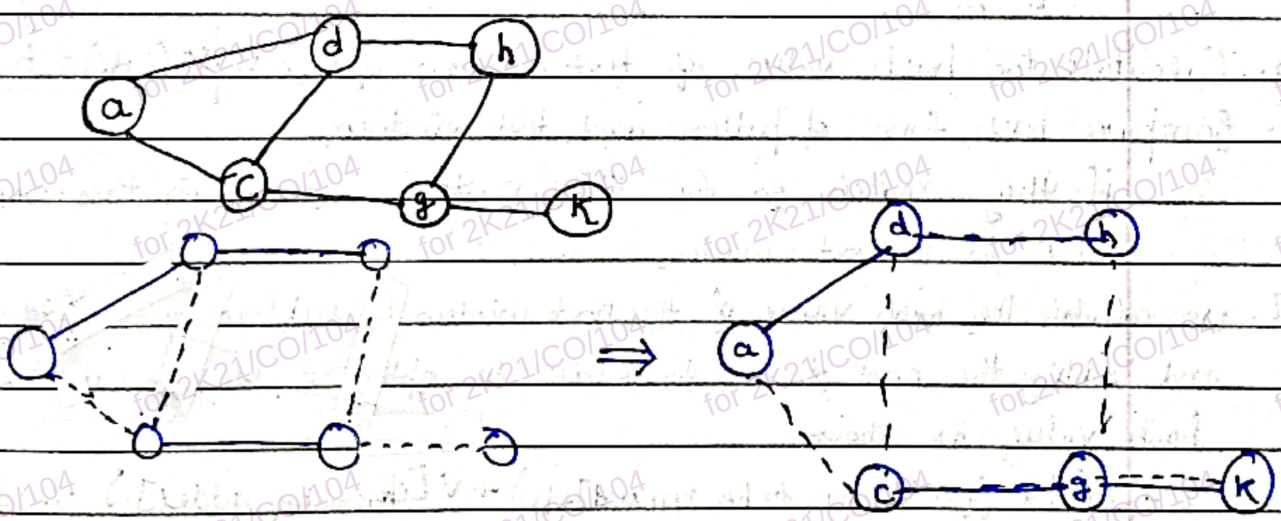
It is an NP-complete problem where we need to find the minimal no of vertex to cover the entire graph. A vertex cover of a graph is subset of vertices which cover every edge.

SOLUTION

APPROX-COVER-PROB (G)

1. $C \leftarrow \phi$
2. $E' \leftarrow E(G)$
3. while ($E' \neq \phi$)
4. do let (u,v) an arbitrary edge in E'
5. $C \leftarrow C \cup \{u,v\}$

   Remove from E' every edge incident either u, or v.
6. Return C.



∴ we get 2 vertex

Q6 (a) Explain Rabin-karp algorithm for string matching using an example? Analyze its run time complexity 2017E (3+2)

It is an algorithm used for searching/matching patterns in text using a hash function. It filters the characters that do not match and then perfams the comparisons on remaing.

∴ A sequence of char is taken and checked for the possibility of the presence of the req string. If possibility is found then matching is done.

Text	Pattern
1. T = ABCCDDAEFG	P = CDD

2. Weights are assigned (A → J)

A B C D E F G H I J
1 2 3 4 5 6 7 8 9 10

3. n = 10 (text length) | m = 3 (pattern length)

4. Value of hash C D D
3 4 4

∴ #(P) = Σ (V * d^(m-1)) mod 13
= ((3×10²) + (4×10¹) + (4×10⁰)) % 13
= 6

5. Calculate the hash value of text window of size m | #(ABC) = 6

6. Compare hash func of pattern and text windows,
if they match go for string match (not in this case)
else go to next window

7. We calculate the hash value of the next window by subtracting the first term and adding the next term. In order to optimize, we can use the prev. hash value as shown

t = ((d * (t - V[char to be removed] * h) + V[char to be added]) % 13

8. We search like this till we find CDD at S = 3 ✓

```
RABIN-KARP-MATCHER (T, P, d, q)
1.  n = T.length
2.  m = P.length
3.  h = d^(m-1) mod q
4.  p = 0
5.  t0 = 0
6.  for (i = 1 → m) // preprocessing
7.      p = (dp + P[i]) mod q
8.      t0 = (dt0 + T[i]) mod q
9.  for (s = 0 → n-m) // matching
10.     if (p == ts)
11.         if (P[1...m] == T[s+1...s+m])
12.             print ("pattern at shift" s)
13.     if (s < n-m)
14.         ts+1 = (d(ts - T[s+1]h)
                + T[s+m+1]) mod q
```

(b) Explain concept of Approximation algorithms. Write approximation algorithm for NP-Hard problem. 2018 (S) old [7 marks]

(b) Explain concept of Approximation algorithms. Write approximation algorithm for NP-Hard problem. 2018 E [7 marks]

2018 S (old)
Q6. Write short notes on
~~(i) P and NP class~~
(ii) NP-Hard problems

ii) A decision problem is NP-hard if every problem in NP can be reduced to it in polynomial time. So, if an algorithm could be developed to solve a NP hard problem in polynomial time, then it would be able to solve any problem in polynomial time. Thus, a NP hard problem is atleast as hard as the hardest problem in NP.

Eg) Graph Coloring, knapsack

APPROXIMATION ALGORITHMS

Approximation algorithms are a class of algorithms that find approximate solutions to optimization problems, typically in cases where finding an exact solution is either impractical or impossible. These algorithms trade off solution quality for computational efficiency, and their primary goal is to find a solution that is guaranteed to be within a certain factor of the optimal solution. It deals with NP completeness for optimization problem.

The goal of approximation algo is to come close to optimal soln in polynomial time

Refer Vertex Cover Problem (Proof of NP complete)

C → cost of soln
C^* → cost of optimal soln
P(n) → Approximation ratio
n → input size

1) Maximization: $\frac{C^*}{C} \leq P(n)$

2) Minimization: $\frac{C^*}{C} \leq P(n)$

(b) What are NP-Complete problems. Give example of one NP-Complete problem. 2018E

[d] What are NP complete problems? 2012E

7[a] Describe the strategy that is used to show that a given problem is NP hard problem. 2012E 7

To show that a given problem is NP hard, we typically use a reduction proof, which involves reducing a known NP-hard problem to the problem in question.

To prove that a problem is hard, one must describe an efficient algorithm to solve a different problem that is already known to be hard, using a hypothetical efficient algo for the problem as a black box subroutine. This reduction implies that if the problem in question were easy, then the other prob would be easy which it isn't (proof by contradiction)

Eg.) Satisfiability problem is proven to be NP hard by giving reduction from a known NP hard problem such as Boolean Satisfaction Problem (SAT)

a) Write short notes on 2016E
(i) P class
(ii) NP class
(iii) NP compete class

b) Write short notes on:- 2017E
(i) P class
(ii) NP class
(iii) NP complete class

c) Write short notes on (3x2=6)
(i) NP class 2018E
(ii) NP compete class 2018E
(iii) Circuit Satisfiability 2018E

(iii) Reducibility

i) P class → Polynomial Time

P class refers to a set of decision problems in CS that can be solved by a deterministic turing machine in polynomial time. ($O(n^k)$)

Thus, P class problems be can solved in polynomial time and verified in polynomial time on a regular computer.

It is a subset of NP class problems.

Such problems can therefore be solved efficiently in practice.

Eg:) Sorting, Shortest Path, Linear Programming Problems.

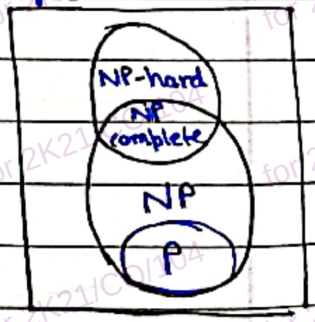
ii:) NP Class → Non deterministic Polynomial Time.

NP class problems refer to a set of decision problems that can be solved by a non-deterministic turing machine in polynomial time. $O(k^n)$

Thus, NP class problems can be solved in exponential time and verified in polynomial time on a regular computer.

Such problems can only be solved efficiently in polynomial time using a non-deterministic algorithm (not ~~DFA~~RL).

Eg:) Travelling Salesman problem (TSP)



iii:) NP Complete Class →

A decision problem is NP-complete if it is both NP-hard and NP. So, it is the one of most difficult problems in NP, and that any problem in NP can be reduced to it in polynomial time.

NP complete class is a class of computational problems for which no efficient solution algorithm has been found.

Eg:) Boolean Satisfiability Problem (SAT)

iii) Reducability

Reducability is a concept used in computational complexity to compare the difficulty of different problems and establish relations among them. It involves transforming (reducing) one problem into another problem in a way that preserves the difficulty / complexity of the problem. Two types of reducability are: polynomial-time and logarithmic-space reducability.

If a problem Q can be reduced to Q' then $\longrightarrow Q \leq_P Q'$

Eg.) Solving linear equations can be reduced to solving quadratic eqns

Reducability is a powerful tool for analysing comp. complexity of problems, it is extensively used in the study of NP-completeness.

∴ L_1 is reducible to L_2 ($L_1 \leq_P L_2$), if there exists a polynomial time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that $\forall x \in \{0,1\}^*$, $x \in L_1$ if and only if $f(x) \in L_2$.

We call f as reduction function and the polynomial time algorithm F that computes f is called a reduction algorithm.

(b) Prove that CNF- satisfiability reduces to clique decision problem.

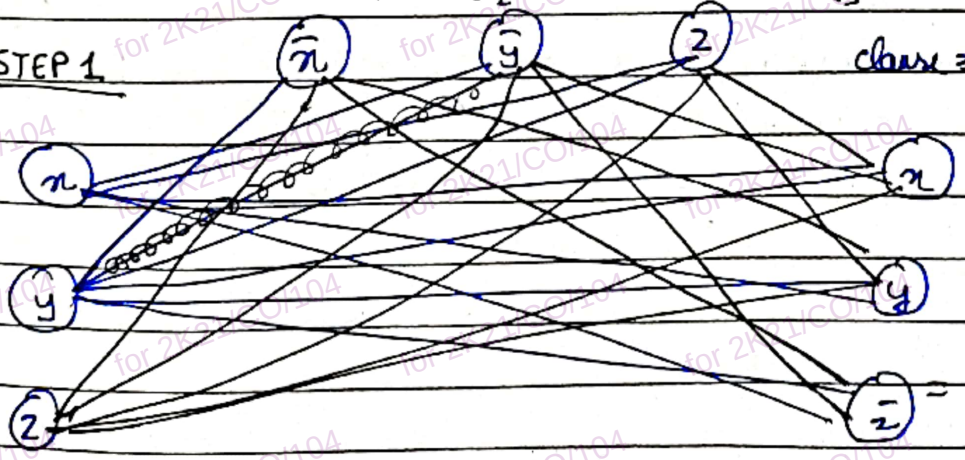
2019

3CNF- clique

$\phi = (x \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z) \wedge (x \vee y \vee \bar{z})$

$c_1 \quad = 1 \quad c_2 \quad c_3$

STEP 1



So, we can observe

$y, \bar{x}, \bar{z} = 1$ which is a clique ($k = 3$)

x	y	z
0	1	0

STEP 2 : substitute in boolean variable ϕ

$$\phi = (x \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z) \wedge (x \vee y \vee \bar{z})$$

$c_1 = 1$ $\quad c_2 = 1$ $\quad c_3 = 2$

$\therefore \phi = 1 \wedge 1 \wedge 1 = 1$ (gets satisfied) ✓

So, the reduction is done in polynomial time → NP complete