09/10/2021

Data Structure and Algorithms

Applied Prep Course

SATYAM SETH

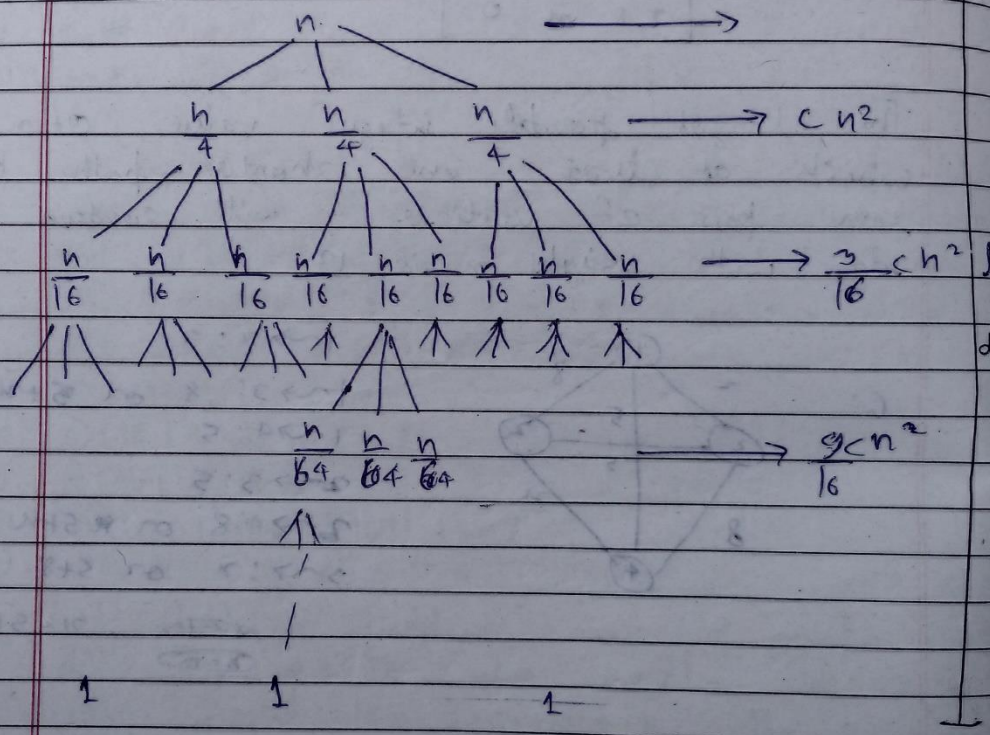
PART-4

Page No.: 352
Date: 29/11/20

Solving Recurrences -

1- Recursion Tree Method -

Eg- $T(n) = 3T(n/4) + c(n^2)$

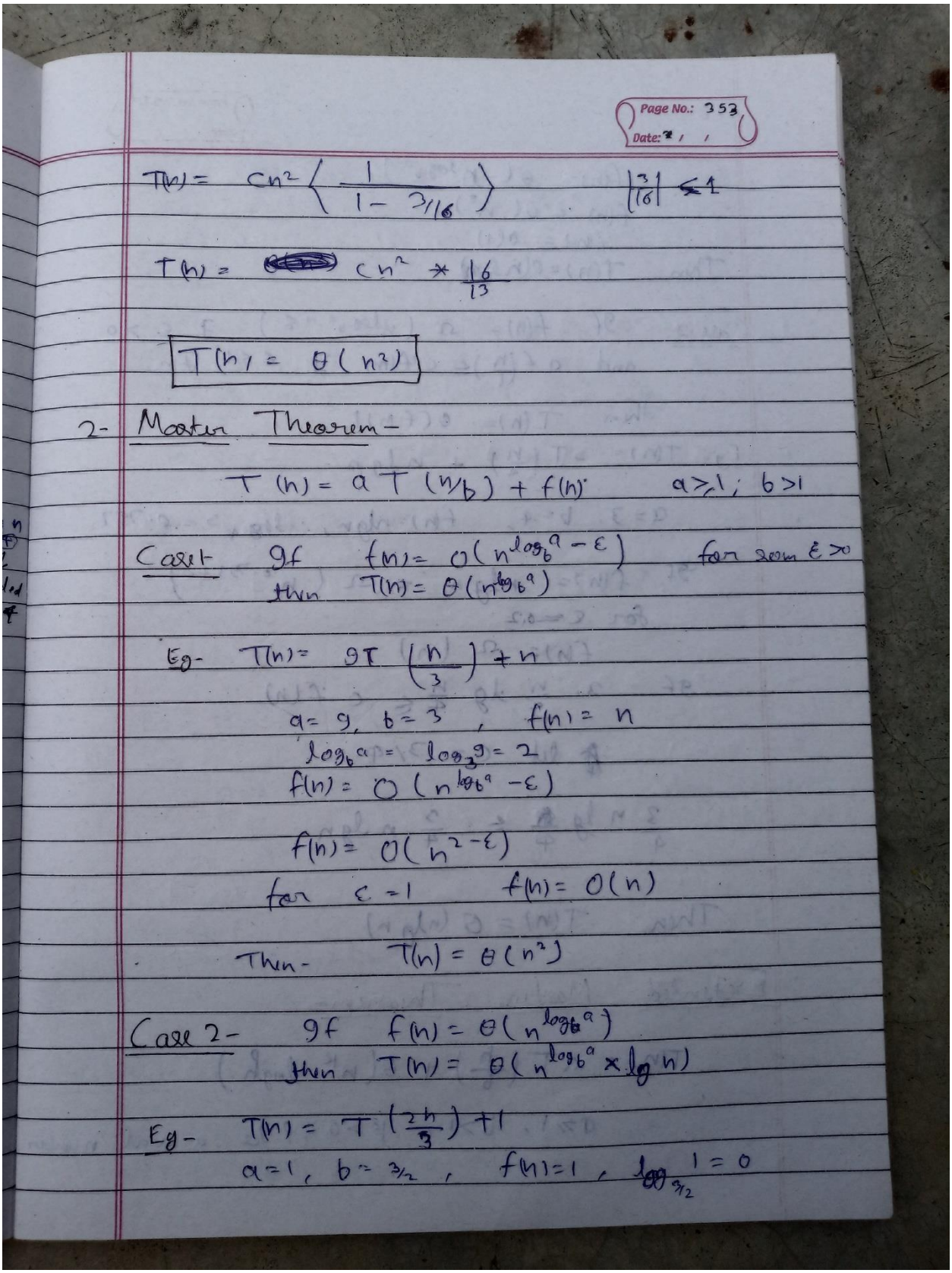


$$= cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \left(\frac{3}{16}\right)^3 cn^2 + \cdots\cdots$$

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

$$\sum_{k=0}^{\infty} ar^k = \frac{a}{1-r} \quad \text{for } |r| < 1$$

$$a = 1, \quad r = \frac{3}{16}$$



Page No.: 353
Date:

$T(n) = cn^2 \left\{ \frac{1}{1 - 3/16} \right\}$ $\left|\frac{3}{16}\right| < 1$

$T(n) = cn^2 * \frac{16}{13}$

$\boxed{T(n) = \theta(n^2)}$

2- Master Theorem -

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

Case 1- If $f(n) = O(n^{\log_b a - \varepsilon})$ for some $\varepsilon > 0$
then $T(n) = \theta(n^{\log_b a})$

Eg- $T(n) = 9T\left(\frac{n}{3}\right) + n$

$a = 9,\ b = 3,\ f(n) = n$

$\log_b a = \log_3 9 = 2$

$f(n) = O(n^{\log_b a - \varepsilon})$

$f(n) = O(n^{2-\varepsilon})$

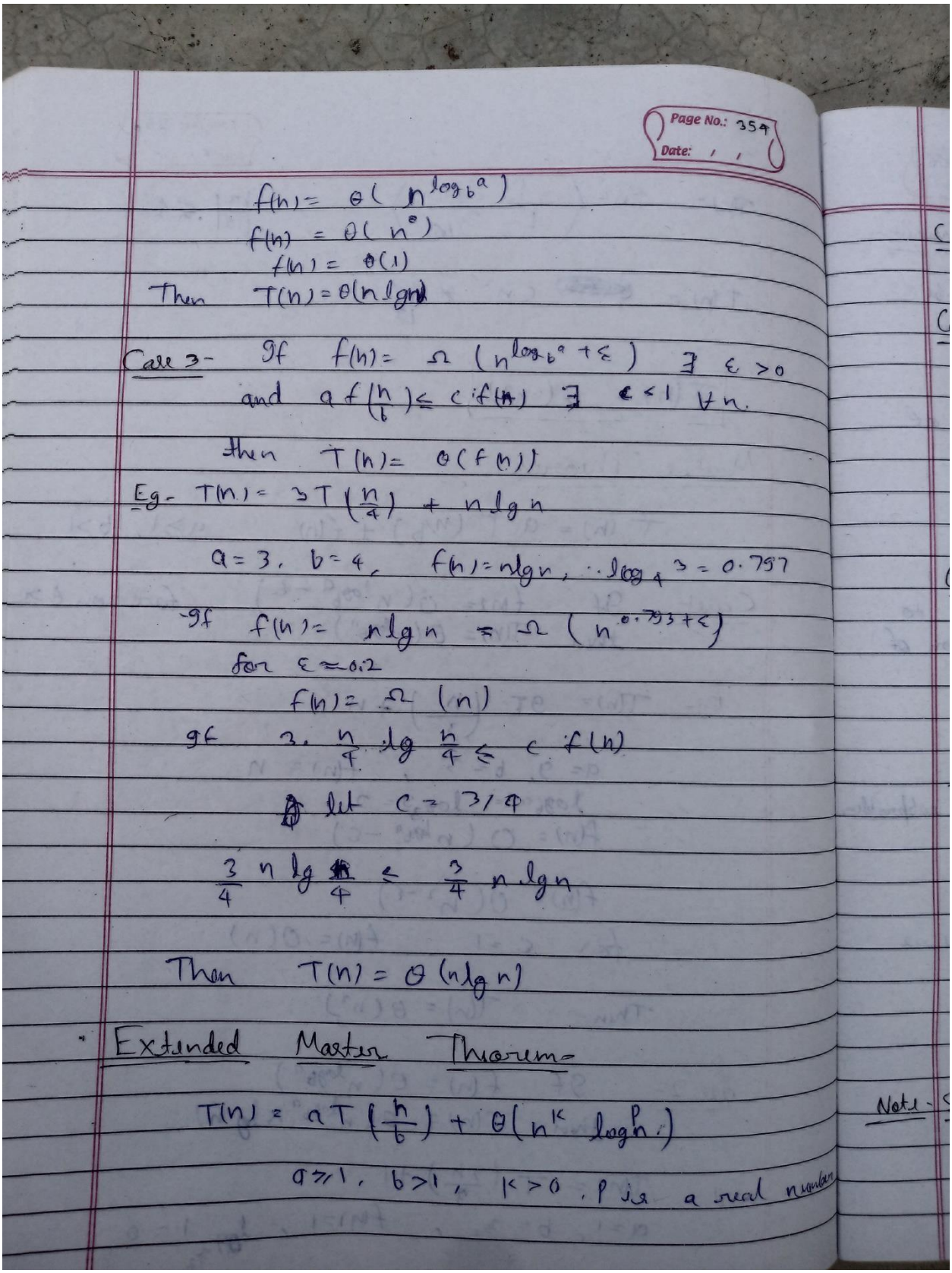
for $\varepsilon = 1$ $f(n) = O(n)$

Then- $T(n) = \theta(n^2)$

Case 2- If $f(n) = \theta(n^{\log_b a})$
then $T(n) = \theta(n^{\log_b a} \times \lg n)$

Eg- $T(n) = T\left(\frac{2n}{3}\right) + 1$

$a = 1,\ b = 3/2,\ f(n) = 1,\ \log_{3/2} 1 = 0$



Page No.: 354
Date: / /

$f(n) = \Theta(n^{\log_b a})$

$f(n) = \Theta(n^0)$

$f(n) = \Theta(1)$

Then $T(n) = \Theta(n \lg n)$

Case 3- If $f(n) = \Omega(n^{\log_b a + \epsilon})$ $\exists\ \epsilon > 0$

and $a f(\frac{n}{b}) \leq c f(n)$ $\exists\ c < 1$ $\forall n$.

Then $T(n) = \Theta(f(n))$

Eg- $T(n) = 3T(\frac{n}{4}) + n \lg n$

$a = 3,\ b = 4,\ f(n) = n \lg n,\ \because \log_4 3 = 0.793$

If $f(n) = n \lg n = \Omega(n^{0.793 + \epsilon})$

for $\epsilon \approx 0.2$

$f(n) = \Omega(n)$

If $3 \cdot \frac{n}{4} \lg \frac{n}{4} \leq c f(n)$

Let $c = 3/4$

$\frac{3}{4} n \lg \frac{n}{4} \leq \frac{3}{4} n \lg n$

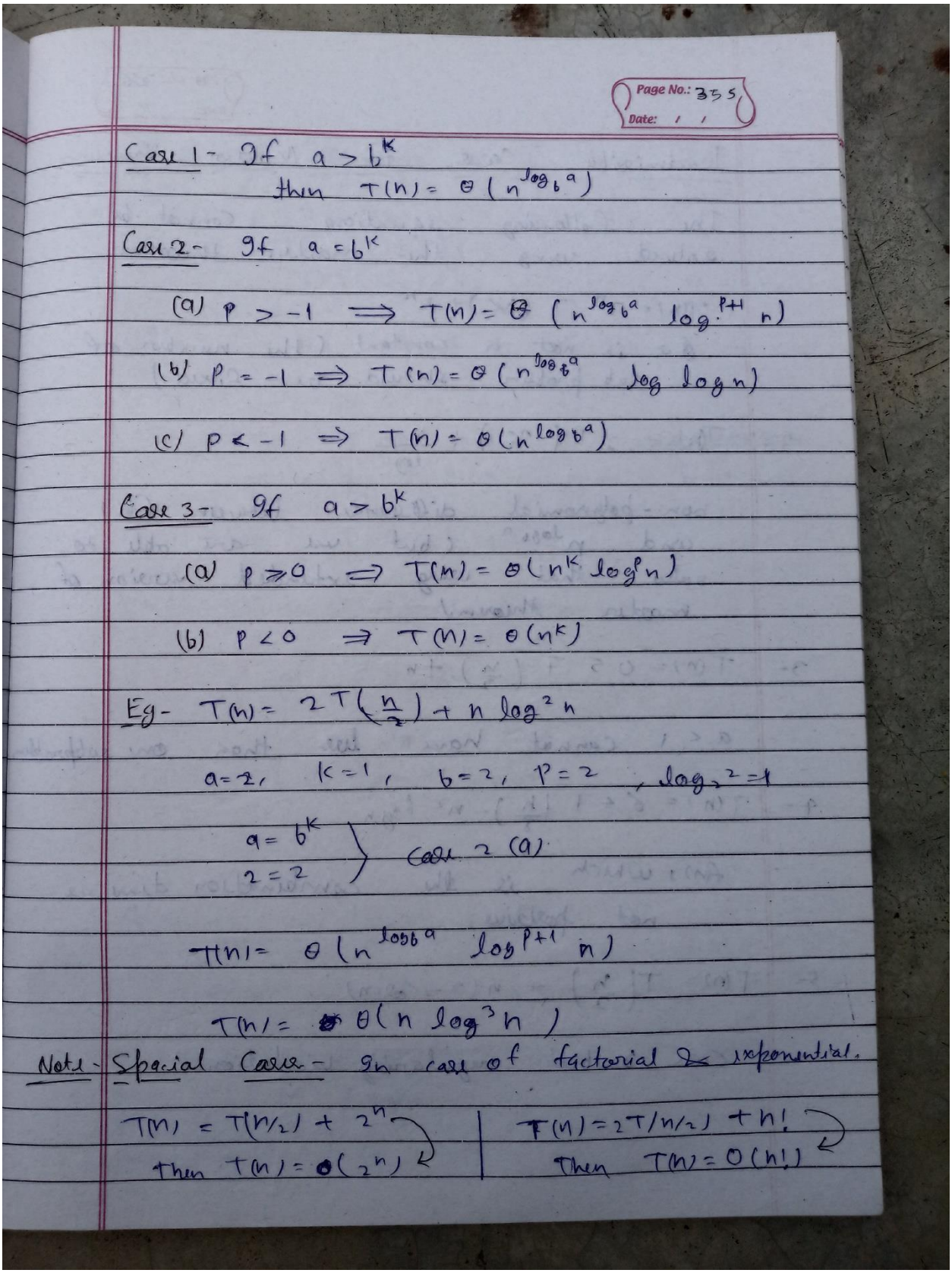
Then $T(n) = \Theta(n \lg n)$

Extended Master Theorem-

$T(n) = aT(\frac{n}{b}) + \Theta(n^k \log^p n)$

$a \geq 1,\ b > 1,\ k > 0,\ p$ is a real number

Note-



Page No.: 355
Date: / /

Case 1 - If $a > b^k$

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

Case 2 - If $a = b^k$

(a) $p > -1 \Rightarrow T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$

(b) $p = -1 \Rightarrow T(n) = \Theta(n^{\log_b a} \log \log n)$

(c) $p < -1 \Rightarrow T(n) = \Theta(n^{\log_b a})$

Case 3 - If $a > b^k$

(a) $p \geq 0 \Rightarrow T(n) = \Theta(n^k \log^p n)$

(b) $p < 0 \Rightarrow T(n) = \Theta(n^k)$

Eg - $T(n) = 2T(\frac{n}{2}) + n \log^2 n$

$a = 2, \quad k = 1, \quad b = 2, \quad p = 2, \quad \log_2 2 = 1$

$a = b^k$
$2 = 2$
} Case 2 (a)

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

$T(n) = \Theta(n \log^3 n)$

Note - Special Case - In case of factorial & exponential.

$T(n) = T(n/2) + 2^n$ ↘
Then $T(n) = \Theta(2^n)$

$T(n) = 2T(n/2) + n!$ ↘
Then $T(n) = \Theta(n!)$

Page No.: 356
Date: / /

Inadmissible Cases for Master theorem

The following equations cannot be solved using the master theorem.

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

a is not a constant (the number of sub problem should be fixed.)

2- $T(n) = 2T(n/2) + \frac{n}{\log n}$

non-polynomial difference between $f(n)$ and $n^{\log_b a}$ (but we are able to solve that using extended version of master theorem.)

3- $T(n) = 0.5\, T(\frac{n}{2}) + n$

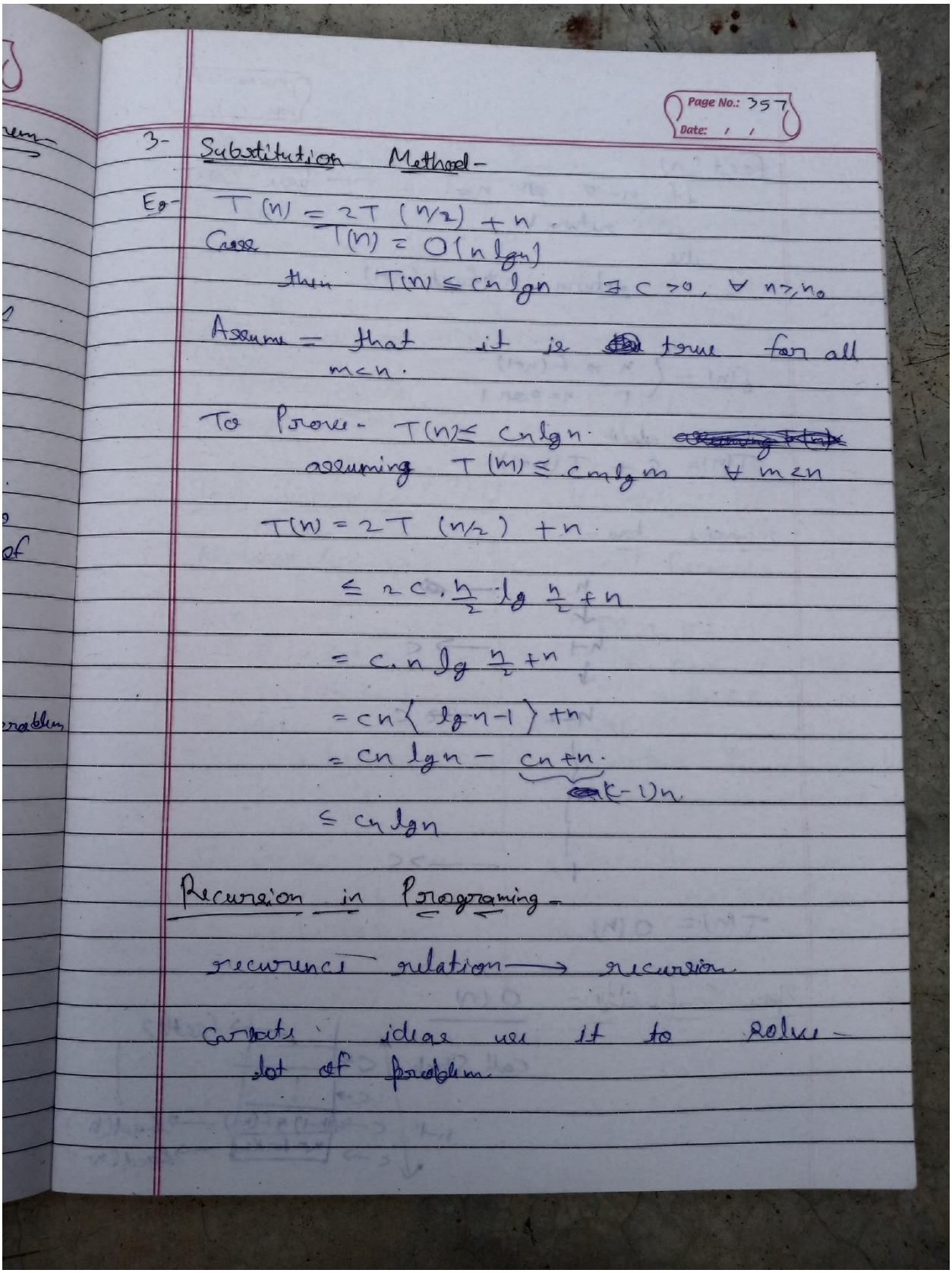
$a < 1$ cannot have less than one subproblem

4- $T(n) = 64\, T(\frac{n}{8}) - n^2 \log n$

$f(n)$ which is the combination time, is not positive

5- $T(n) = T(\frac{n}{2}) + n(2 - \cos n)$

case 3 but regularity violation.



Page No.: 357
Date: / /

3- Substitution Method-

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

Case $T(n) = O(n \lg n)$

then $T(n) \leq cn \lg n \quad \exists\, c > 0, \forall\, n \geq n_0$

Assume - that it is true for all $m < n$.

To Prove - $T(n) \leq cn \lg n$

assuming $T(m) \leq cm \lg m \quad \forall\, m < n$

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

$\leq 2c \cdot \frac{n}{2} \lg \frac{n}{2} + n$

$= c \cdot n \lg \frac{n}{2} + n$

$= cn \{ \lg n - 1 \} + n$

$= cn \lg n - cn + n$

$\leq (c-1)n$

$\leq cn \lg n$

Recursion in Programing -

recurrence relation ⟶ recursion.

Computer scientist use it to solve lot of problem.

Page No.: 358
Date: / /

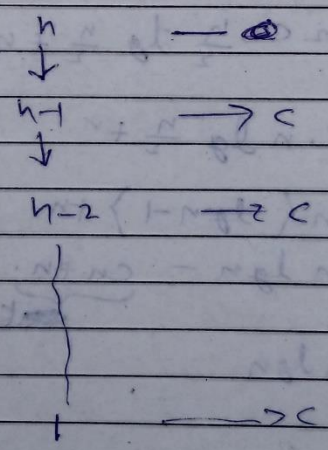
```
fact(n)
    if n=0 or n=1        ← base case
        return 1
    else
        return n*fact(n-1)
```

$$f(n) = \begin{cases} n * f(n-1) \\ 1 \quad n=0 \text{ or } 1 \end{cases}$$

Recurrence relation

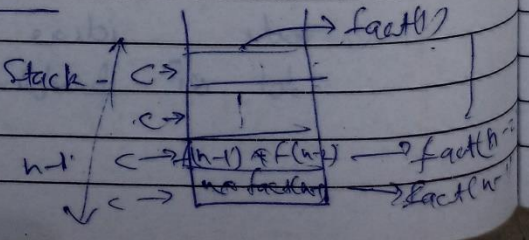
$T(n) = c + T(n-1)$

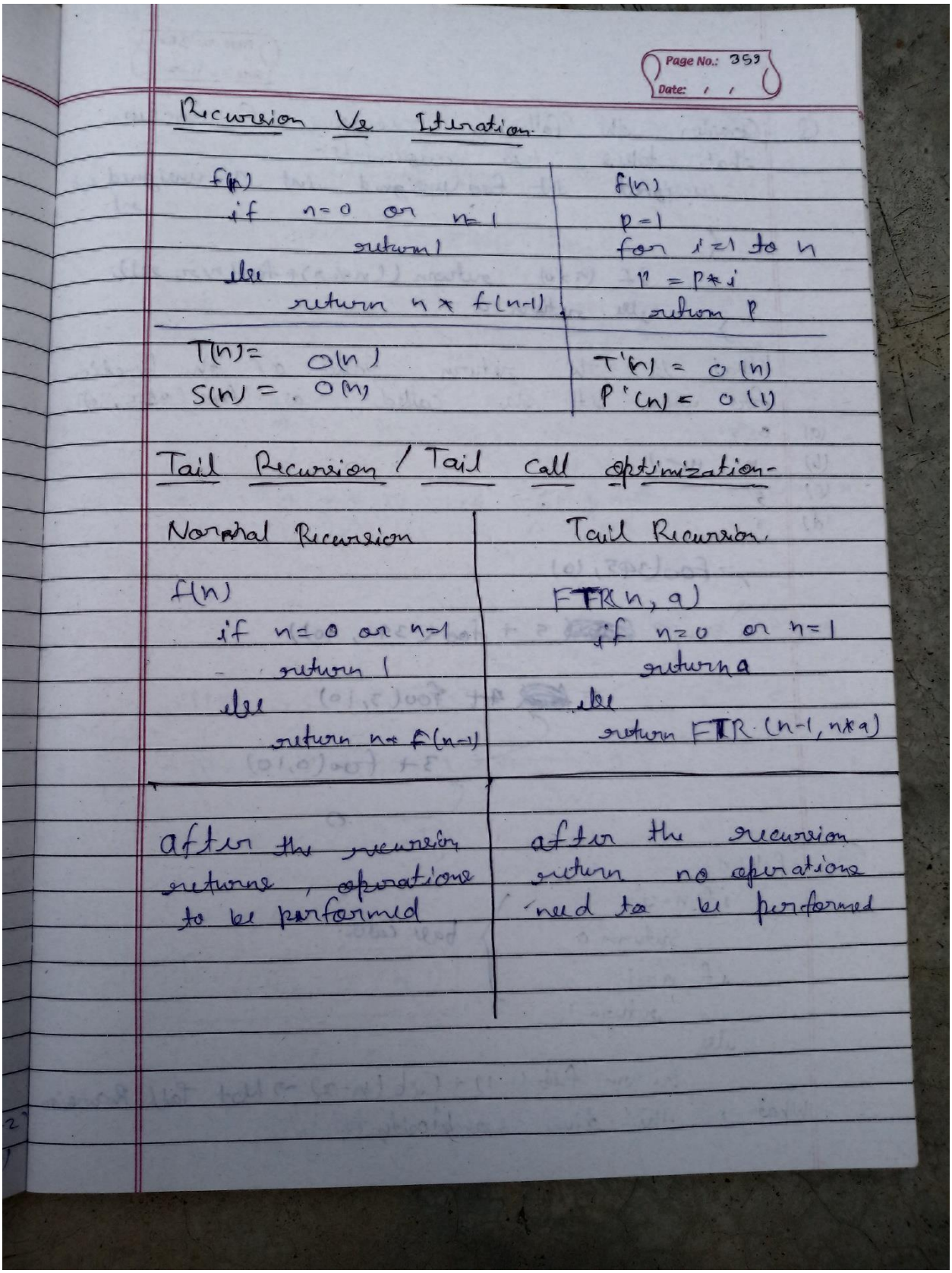
recursive tree -



$T(n) = O(n)$

Space Complexity - $O(n)$





Page No.: 359
Date: / /

Recursion Vs Iteration.

f(n)	f(n)
if n=0 or n=1 return 1 else return n * f(n-1)	p=1 for i=1 to n p = p*i return p
T(n) = O(n) S(n) = O(n)	T'(n) = O(n) P'(n) = O(1)

Tail Recursion / Tail call optimization-

Normal Recursion	Tail Recursion.
f(n) if n=0 or n=1 return 1 else return n* f(n-1)	FTR(n, a) if n=0 or n=1 return a else return FTR(n-1, n*a)
after the recursion returns, operations to be performed	after the recursion return no operations need to be performed

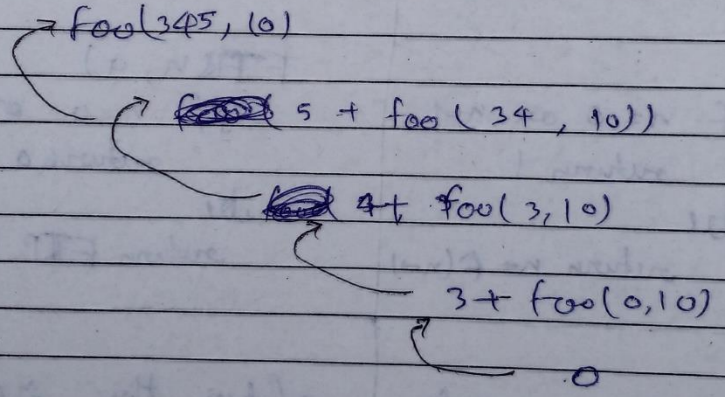
Page No.: 360
Date: 30/11/22

Q- Consider the following recursive C function that takes two arguments-

```
unsigned int foo(unsigned int n, unsigned int r)
{
    if (n>0) return ((n%r)+foo(n/r, r));
    else return 0;
}
```

What is the return value of the function foo when it is called as foo(345, 10)?

(a) 3+5
(b) 12 (✓)
(c) 5
(d) 3



Q-

```
fib(n)
    if n=0
        return 0
    if n=1
        return 1
    else
        return fib(n-1) + fib(n-2)
```

base case.

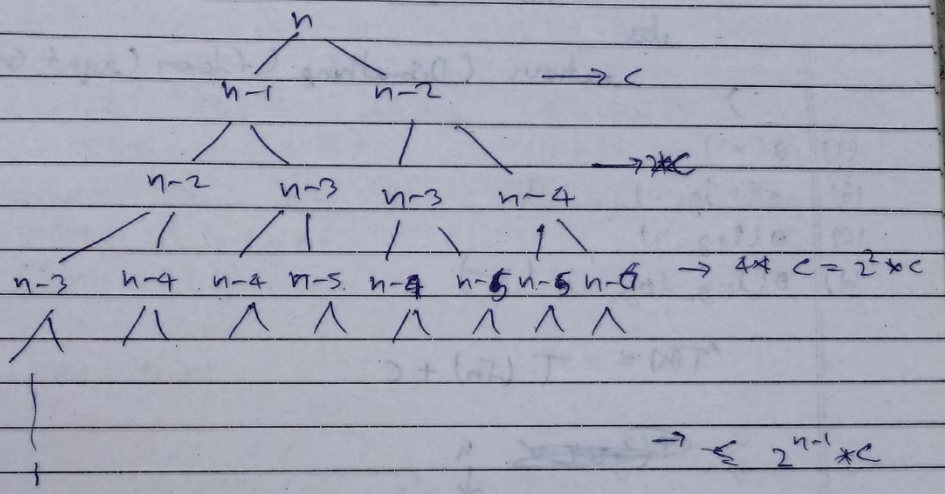
→ Not Tail Recursion

What is the time Complexity?

Page No.: 361
Date: / /

$$fib(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ fib(n-1) + fib(n-2); & \text{o/w} \end{cases}$$

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



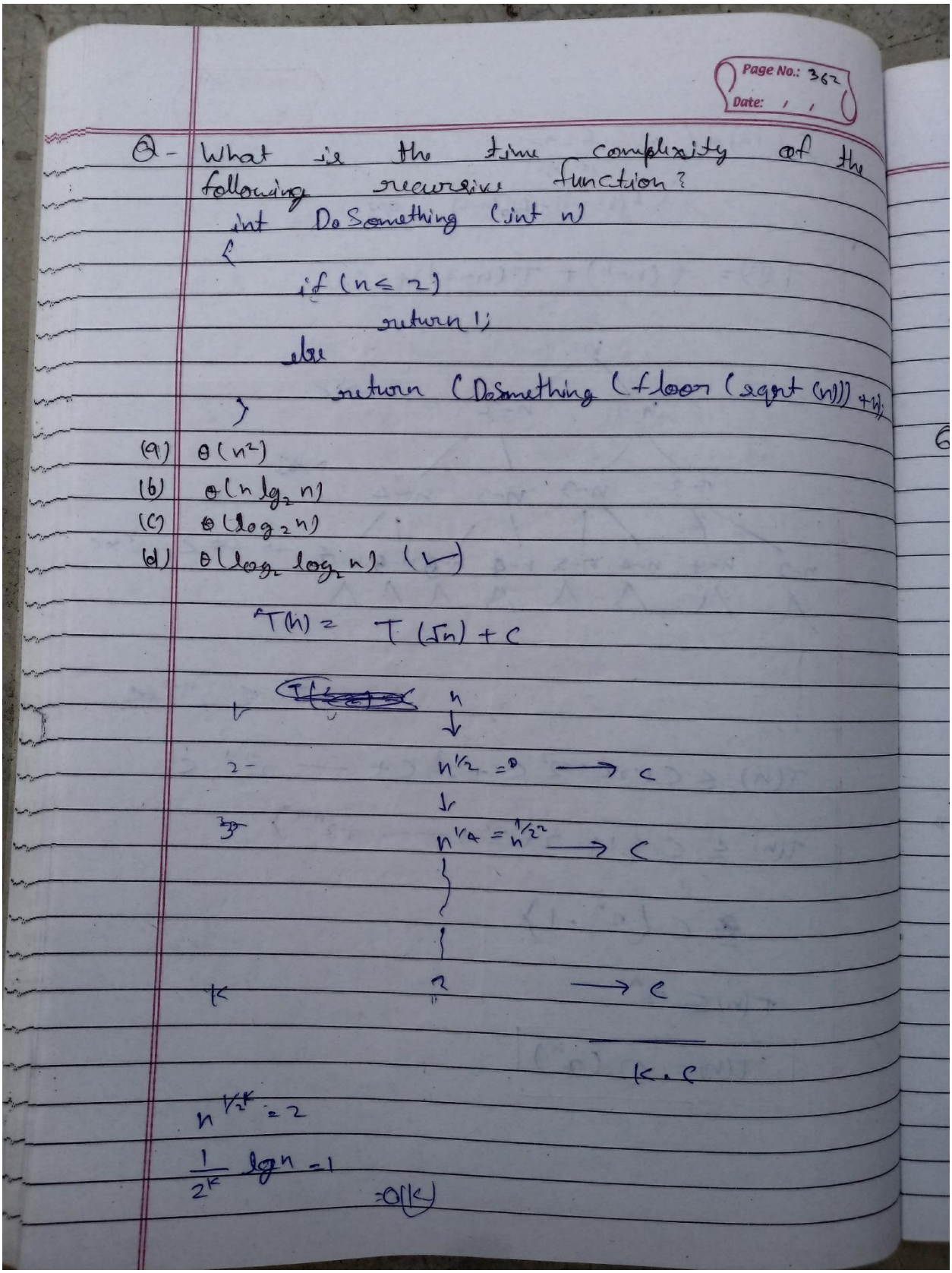
$$T(n) \leq c + 2c + 2^2 c + 2^3 c + \cdots 2^{n-1} c$$

$$T(n) \leq c\{1 + 2 + 2^2 + \cdots + 2^{n-1}\}$$

$$\leq c\{2^n - 1\}$$

$$T(n) \leq 2^n$$

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



Page No.: 362
Date: / /

Q- What is the time complexity of the following recursive function?

```
int DoSomething (int n)
{
    if (n <= 2)
        return 1;
    else
        return (DoSomething (floor (sqrt (n))) +1);
}
```

(a) $\theta(n^2)$
(b) $\theta(n \log_2 n)$
(c) $\theta(\log_2 n)$
(d) $\theta(\log_2 \log_2 n)$ (✓)

$T(n) = T(\sqrt{n}) + c$

n
↓
$n^{1/2} = 0 \longrightarrow c$
↓
$n^{1/4} = n^{1/2^2} \longrightarrow c$
}
}
2 $\longrightarrow c$

k

$k \cdot c$

$n^{1/2^k} = 2$

$\frac{1}{2^k} \log n = 1$

$= O(k)$

Page No.: 363
Date: / /

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

$-k + \lg(\lg n) = 0$

$\lg \lg n = k$

$\lg\lg n = k.$

$\lg_2 \frac{1}{2^k} = \lg_2 2^{-k}$

$= -k$

Q- Consider the following recurrence-

$T(n) = 2T(\sqrt{n}) + 1, \quad T(1) = 1$

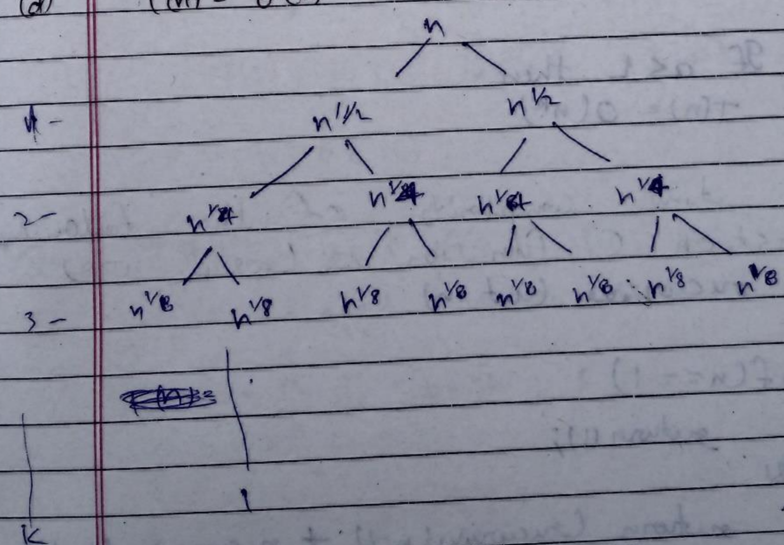
which one of the following is True?

(a) $T(n) = \Theta(\lg \lg n)$

(b) $T(n) = \Theta(\lg n)$ (✓)

(c) $T(n) = \Theta(\sqrt{n})$

(d) $T(n) = \Theta(n)$



$T(n) = 1 + 2 + 2^2 + 2^3 + \ldots\ldots 2^{k-1}$

$T(n) = 2^k - 1$

$k = \lg(\lg n)$

Page No.: 364
Date: / /

$T(n) = 2^{\lg(\lg n)} - 1$ | $2^{\lg_2 a} = a$

$\Rightarrow T(n) = \lg_2 n - 1$

$T(n) = \Theta(\lg n)$

Master Theorem Shortcuts-

$T(n) = a\,T(n-b) + O(n^k)$

$a > 0,\ b \geq 1,\ k \geq 0$

Case 1- If $a > 1$ then

$T(n) = O(n^k a^{n/b})$

Case 2- If $a = 1$ then

$T(n) = O(n^{k+1})$

Case 3- If $a < 1$ then

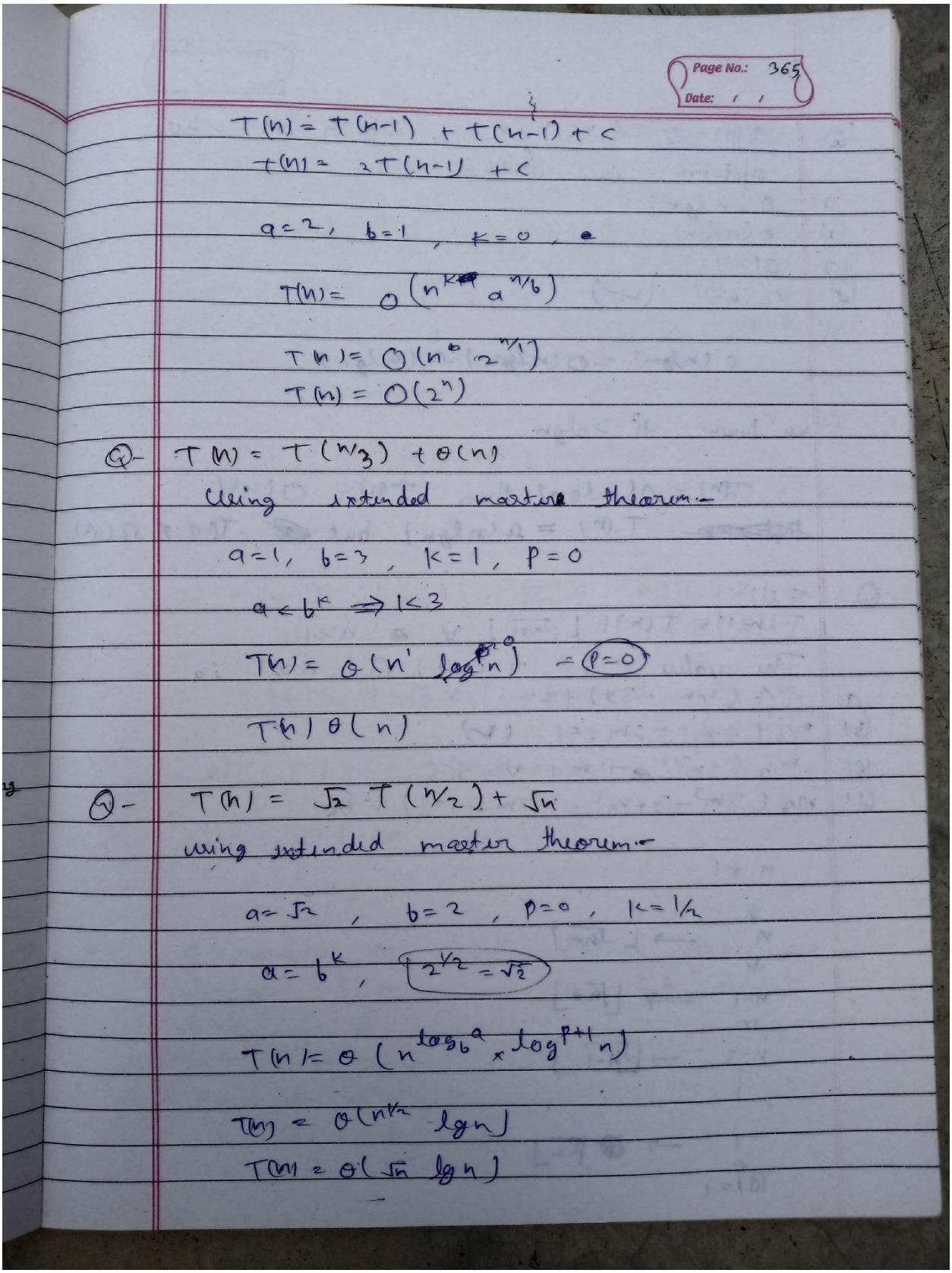
$T(n) = O(n^k)$

Eg- The time complexity of the following C function C function is (assume $n > 0$)

```
int recursive (int n)
{
    if(n==1)
        return (1);
    else
        return (recursive(n-1) + recursive(n-1));
}
```

(a) $O(n)$ (c) $O(n^2)$

(b) $O(n \lg n)$ (d) $O(2^n)$



Page No.: 365
Date: / /

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

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

$a=2, \quad b=1, \quad k=0, \quad$ e

$T(n) = O(n^k a^{n/b})$

$T(n) = O(n^0 \cdot 2^{n/1})$

$T(n) = O(2^n)$

Q- $T(n) = T(n/3) + \theta(n)$

Using intended master theorem:-

$a=1, \quad b=3, \quad k=1, \quad p=0$

$a < b^k \Rightarrow 1 < 3$

$T(n) = \theta(n^1 \log^0 n) \quad - (p=0)$

$T(n) \theta(n)$

Q- $T(n) = \sqrt{2}\, T(n/2) + \sqrt{n}$

using intended master theorem:-

$a = \sqrt{2}, \quad b=2, \quad p=0, \quad k = 1/2$

$a = b^k, \quad (2^{1/2} = \sqrt{2})$

$T(n) = \theta(n^{\log_b a} \cdot \log^{p+1} n)$

$T(n) = \theta(n^{1/2} \lg n)$

$T(n) = \theta(\sqrt{n} \lg n)$

Page No.: 366
Date: / /

Q- $T(n) = 2T(n/2) + n$, which of the options is false :

(a) $\Theta(n \lg n)$
(b) $o(n \lg n)$
(c) $O(n^2)$
(d) $\Omega(n^2)$ (✓)

$\Theta(n \lg n) = O(n \lg n) \cap \Omega(n \lg n)$

we know $n^2 > n \lg n$

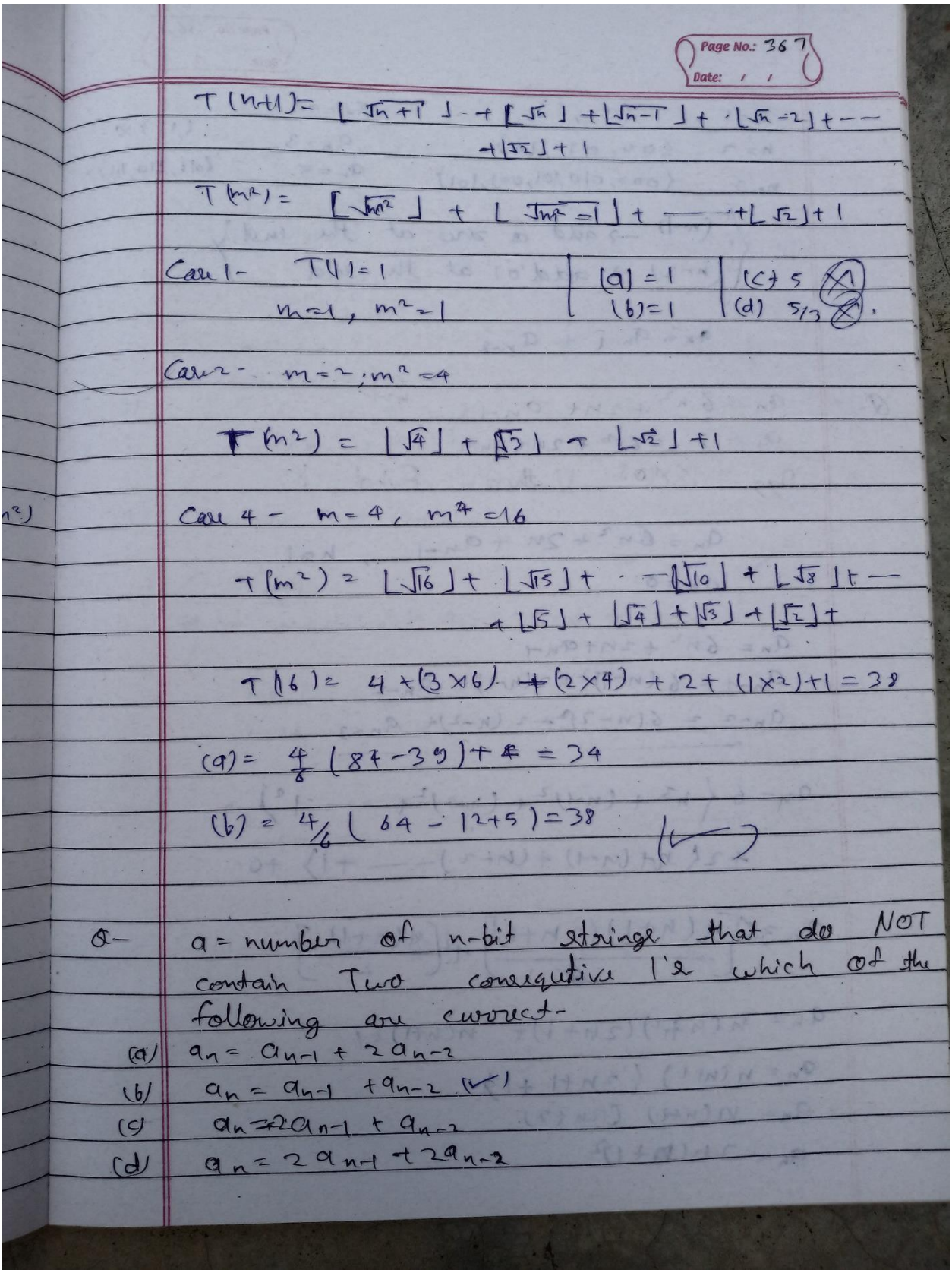
$T(n) = O(n \lg n)$ then $T(n) = O(n^2)$

$T(n) = \Omega(n \lg n)$ but $T(n) \neq \Omega(n^2)$

Q- $T(1) = 1$
$T(n+1) = T(n) + \lfloor \sqrt{n+1} \rfloor \quad \forall \quad n \geq 1$
The value of $T(m^2)$ for $m \geq 1$ is

(a) $m/6 (2/m - 3e) + 4$
(b) $m/6 (4m^2 - 3m + 5)$ (✓)
(c) $m/2 (3m^{2.5} - 11m + 20) = 5$
(d) $m/6 (5m^3 - 3 + m^2 + 137m - 104) + 5/6$

$n+1$
$n \longrightarrow \lfloor \sqrt{n+1} \rfloor$
$n-1 \longrightarrow \lfloor \sqrt{n} \rfloor$
$n-2 \longrightarrow \lfloor \sqrt{n-1} \rfloor$
$1 \longrightarrow \lfloor \sqrt{2} \rfloor$
$T(1) = 1$



Page No.: 367
Date: / /

$T(n+1) = \lfloor \sqrt{n+1} \rfloor + \lfloor \sqrt{n} \rfloor + \lfloor \sqrt{n-1} \rfloor + \lfloor \sqrt{n-2} \rfloor + \cdots + \lfloor \sqrt{2} \rfloor + 1$

$T(m^2) = \lfloor \sqrt{m^2} \rfloor + \lfloor \sqrt{m^2-1} \rfloor + \cdots + \lfloor \sqrt{2} \rfloor + 1$

Case 1 - $T(1) = 1$ | $m=1, m^2=1$ | (a) = 1 | (b) = 1 | (c) 5 | (d) 5/3

Case 2 - $m=2; m^2=4$

$T(m^2) = \lfloor \sqrt{4} \rfloor + \lfloor \sqrt{3} \rfloor + \lfloor \sqrt{2} \rfloor + 1$

Case 4 - $m=4, m^2=16$

$T(m^2) = \lfloor \sqrt{16} \rfloor + \lfloor \sqrt{15} \rfloor + \cdots + \lfloor \sqrt{10} \rfloor + \lfloor \sqrt{8} \rfloor + \cdots + \lfloor \sqrt{5} \rfloor + \lfloor \sqrt{4} \rfloor + \lfloor \sqrt{3} \rfloor + \lfloor \sqrt{2} \rfloor + 1$

$T(16) = 4 + (3 \times 6) + (2 \times 4) + 2 + (1 \times 2) + 1 = 38$

(a) $= \frac{4}{8}(84 - 39) + 4 = 34$

(b) $= \frac{4}{6}(64 - 12 + 5) = 38$ ✓

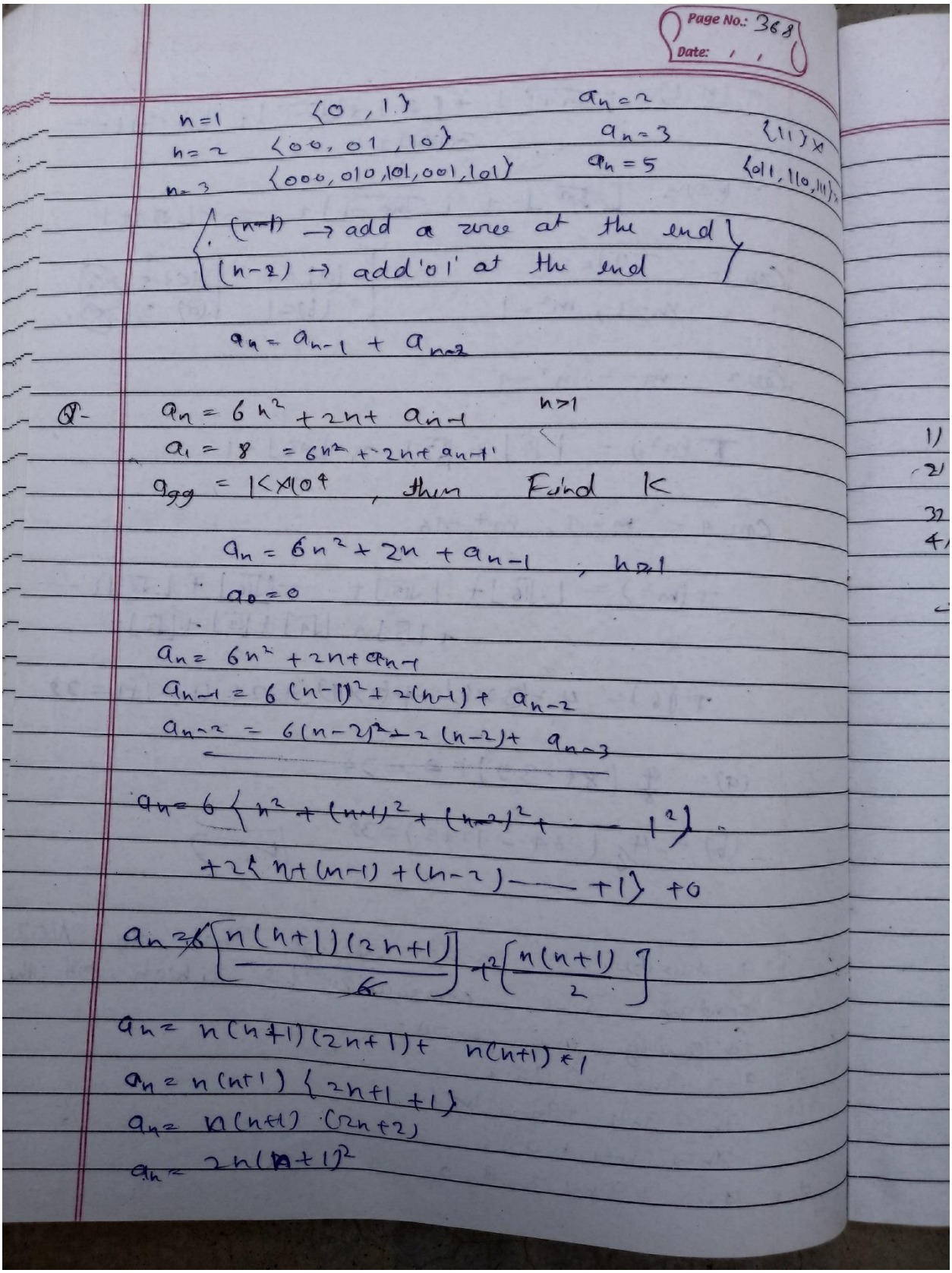
Q- a = number of n-bit strings that do NOT contain Two consecutive 1's which of the following are correct-

(a) $a_n = a_{n-1} + 2a_{n-2}$

(b) $a_n = a_{n-1} + a_{n-2}$ ✓

(c) $a_n = 2a_{n-1} + a_{n-2}$

(d) $a_n = 2a_{n-1} + 2a_{n-2}$



Page No.: 368
Date: / /

n=1 {0, 1} $a_n = 2$

n=2 {00, 01, 10} $a_n = 3$ {11} ✗

n=3 {000, 010, 101, 001, 101} $a_n = 5$ {011, 110, 111}

{(n-1) → add a zero at the end}
{(n-2) → add '01' at the end}

$a_n = a_{n-1} + a_{n-2}$

Q- $a_n = 6n^2 + 2n + a_{n-1}$ $n > 1$

$a_1 = 8 = 6n^2 + 2n + a_{n-1}$

$a_{99} = 1 \times 10^4$, then Find k

$a_n = 6n^2 + 2n + a_{n-1}$, $n \geq 1$

$a_0 = 0$

$a_n = 6n^2 + 2n + a_{n-1}$

$a_{n-1} = 6(n-1)^2 + 2(n-1) + a_{n-2}$

$a_{n-2} = 6(n-2)^2 + 2(n-2) + a_{n-3}$

$a_n = 6\{n^2 + (n-1)^2 + (n-2)^2 + \dots 1^2\}$

$+2\{n + (n-1) + (n-2) \dots + 1\} + 0$

$a_n = 6\left[\frac{n(n+1)(2n+1)}{6}\right] + 2\left[\frac{n(n+1)}{2}\right]$

$a_n = n(n+1)(2n+1) + n(n+1) \cdot 1$

$a_n = n(n+1)\{2n+1+1\}$

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

$a_n = 2n(n+1)^2$

Page No.: 389
Date: / /

$a_{99} = 2 + 99(100)^2$

$a_{99} = 198 \times 10^4$

$\boxed{k = 198}$

Linear Search –

Algorithm –

```
Linear Search (a, x)
1)  for i = 1 to a.length
2)      if a[i] = x
3)          return "found at i"
4)  return "not found"
```

↳ C Code –

```
int search(int arr[], int n, int x)
{
    int i;
    for(i = 0; i < n; i++)
        if (arr[i] == x)
            return i;
    return -1;
}
```

Time Complexity – O(n)

Space Complexity – O(1)

Page No.: 370
Date: / /

Binary Search -

- Binary Search works only on sorted arrays.
- It will not work on Linked List

Algorithm -

(i) Iterative -

```
BinarySearch(A, x, l, n)
l=1; r=n
while (l≤r)
    m = ⌊(l+r)/2⌋;
    if x < A[m]
        r = m-1
    if x > A[m]
        l = m+1
    if x = A[m]
        return "Found"
return "NOT FOUND"
```

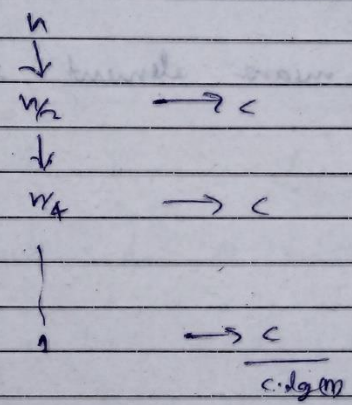
(ii) Recursive -

```
BinarySearchRecursive(A, x, l, r)
if (l>r)
    return "NOTTFOUND"
m = ⌊(l+r)/2⌋
if x = A[m]
```

Page No.: 371
Date: / /

return "FOUND"
if x<A[m]
BinarySearchRecursive(A, x, l, m-1)
if x>A[m]
BinarySearchRecursive(A, x, m+1, r)

Time Complexity- $\Theta(\lg n)$



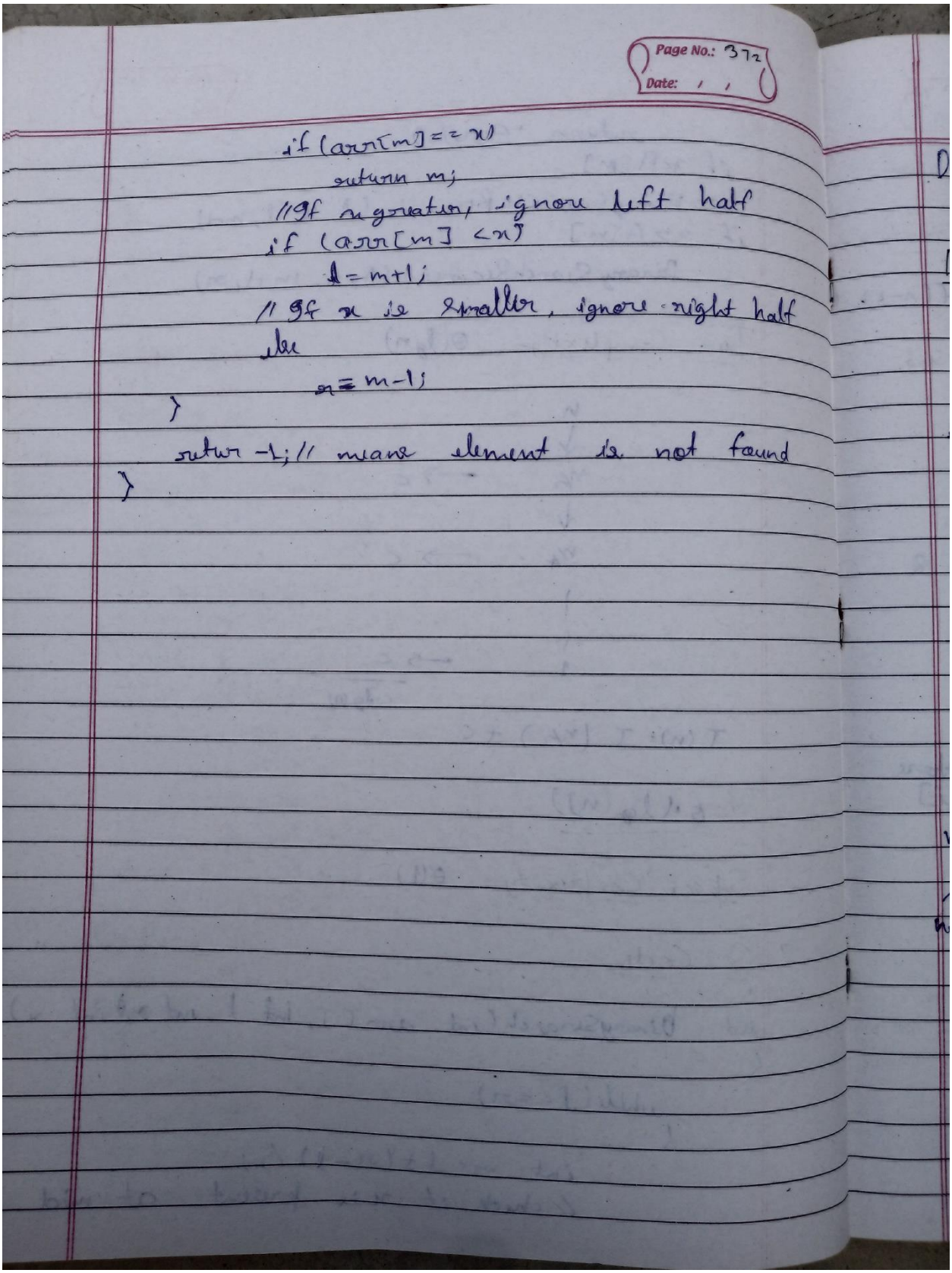
$T(n) = T(n/2) + c$

$T(n) = \Theta(\lg(n))$

Space Complexity- $\Theta(1)$

→ C Code-

```
int BinarySearch(int arr[], int l, int r, int x)
{
    while(l<=r)
    {
        int m=l+(r-l)/2;
        //check if x is present at mid
```



Page No.: 372
Date: / /

```
if (arr[m] == x)
    return m;
// If x greater, ignore left half
if (arr[m] < x)
    l = m+1;
// If x is smaller, ignore right half
else
    r = m-1;
}
return -1; // means element is not found
}
```

Dynamic Programming

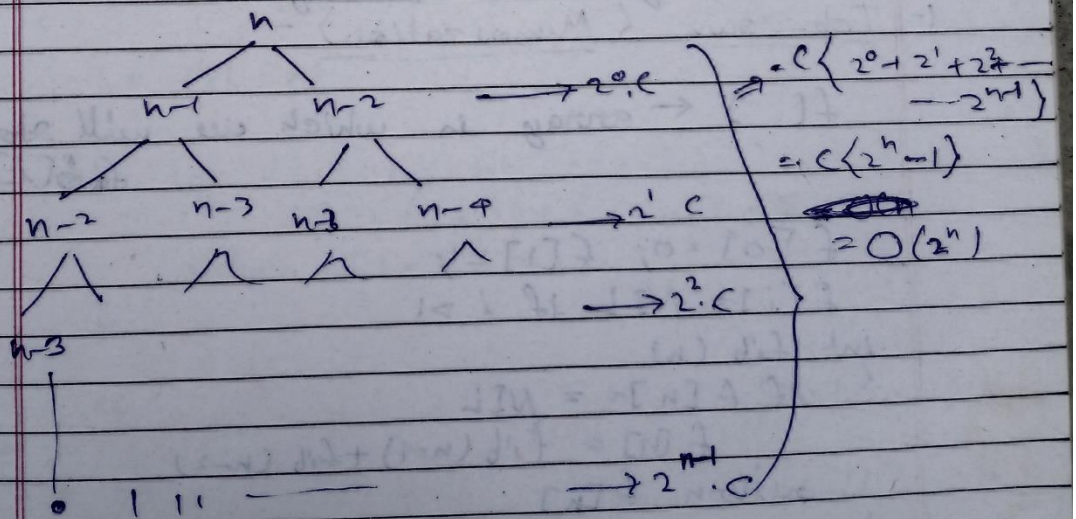
Page No.: 373
Date: 2/11/20

Dynamic Programming is algo. design to design more efficient algo.

Fibonacci numbers

$$Fib(n) = \begin{cases} n & \text{if } n=0 \text{ or } n=1 \\ Fib(n-1) + Fib(n-2) & \text{if } n>=2 \end{cases}$$

```
int Fab(n)
{
    int(n<=1)
        return n;
    else
        return fib(n-1)+fib(n-2);
}
```



Time complexity = $O(2^n)$
Space Complexity = $O(n)$

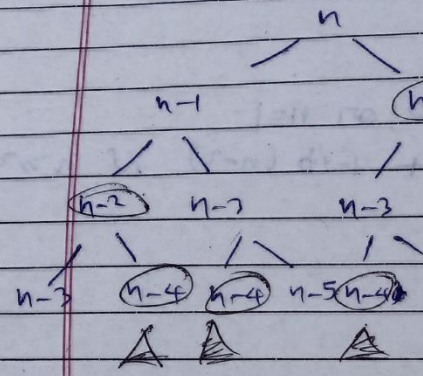
$$T(n) = \begin{cases} T(n-1) + T(n-2) \\ 1 \text{ if } n \leq 1 \end{cases}$$

Page No.: 37
Date: / /

Q. Can we do better than this from a time complexity perspective?

1- Optimal substructure -

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



2- Overlapping sub problems -

idea - Why not compute fib(n-4) once & store it → reuse it

Two ways to storing -

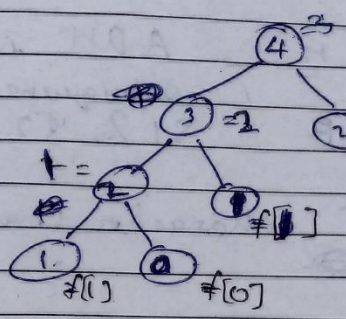
1- Top-down (Memoization) -

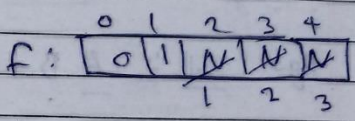
f[ ] ← array in which we will store fib[i]

```
f[0] = 0; f[1] = 1;
f[i] = NIL if i > 1
int fib(n)
{
  if f[n] == NIL
      f[n] = fib(n-1) + fib(n-2)
  return f[n]
}
```

Page No.: 375
Date: / /

Eg- Fib (4)





f[i] computing only once.

Time Complexity - O(n)

Space Complexity - stack + array
= O(n) + O(n)
= O(n)

2- Bottom-up DP [Tabulation] -

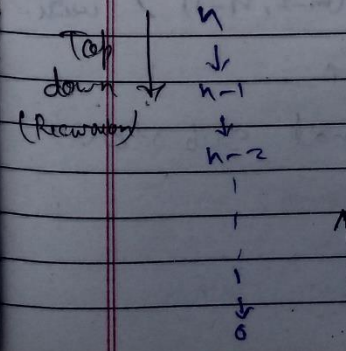
f[] : array

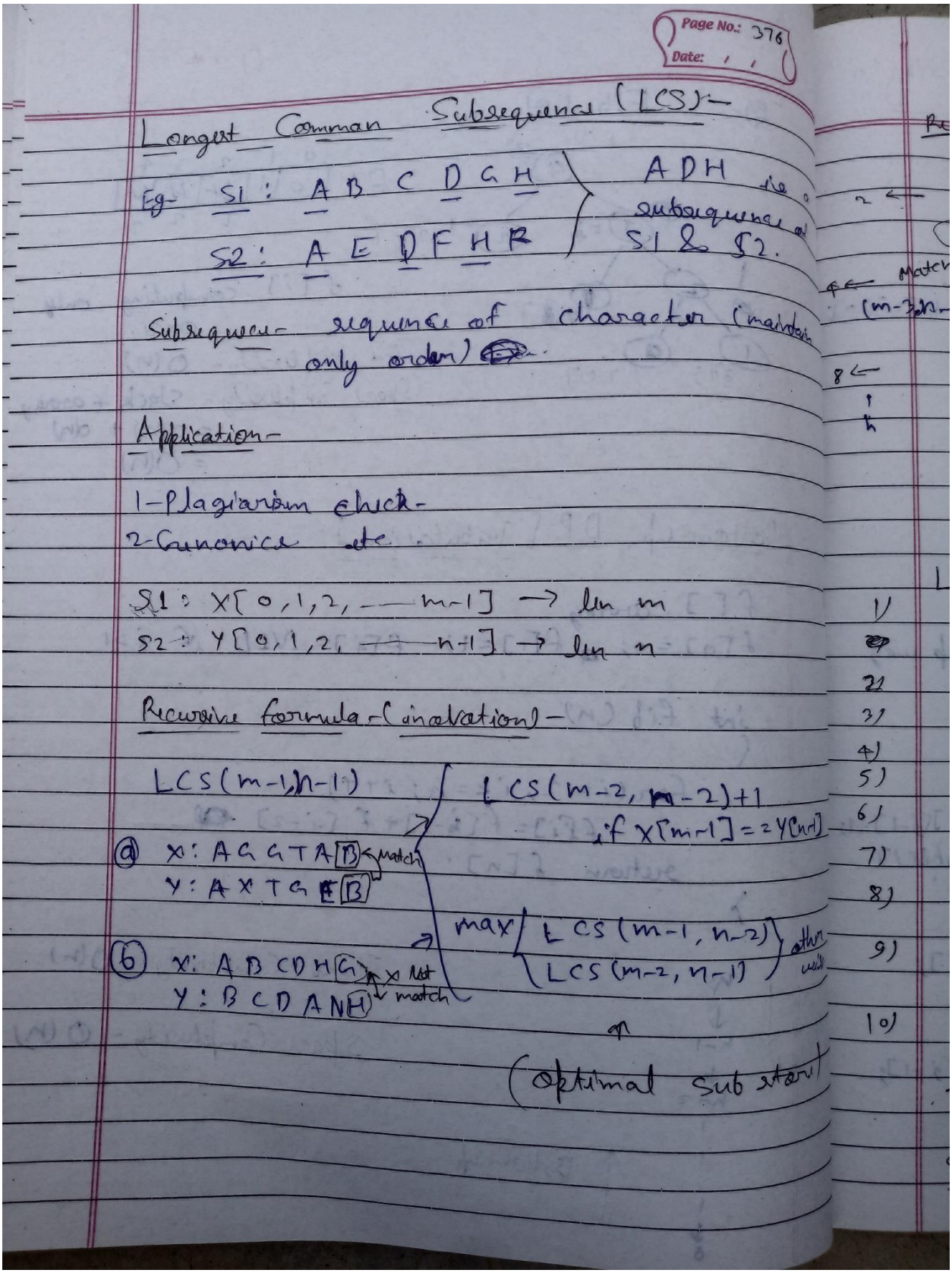
f[0] = 0; f[1] = 1; f[i] = NIL if i > 1

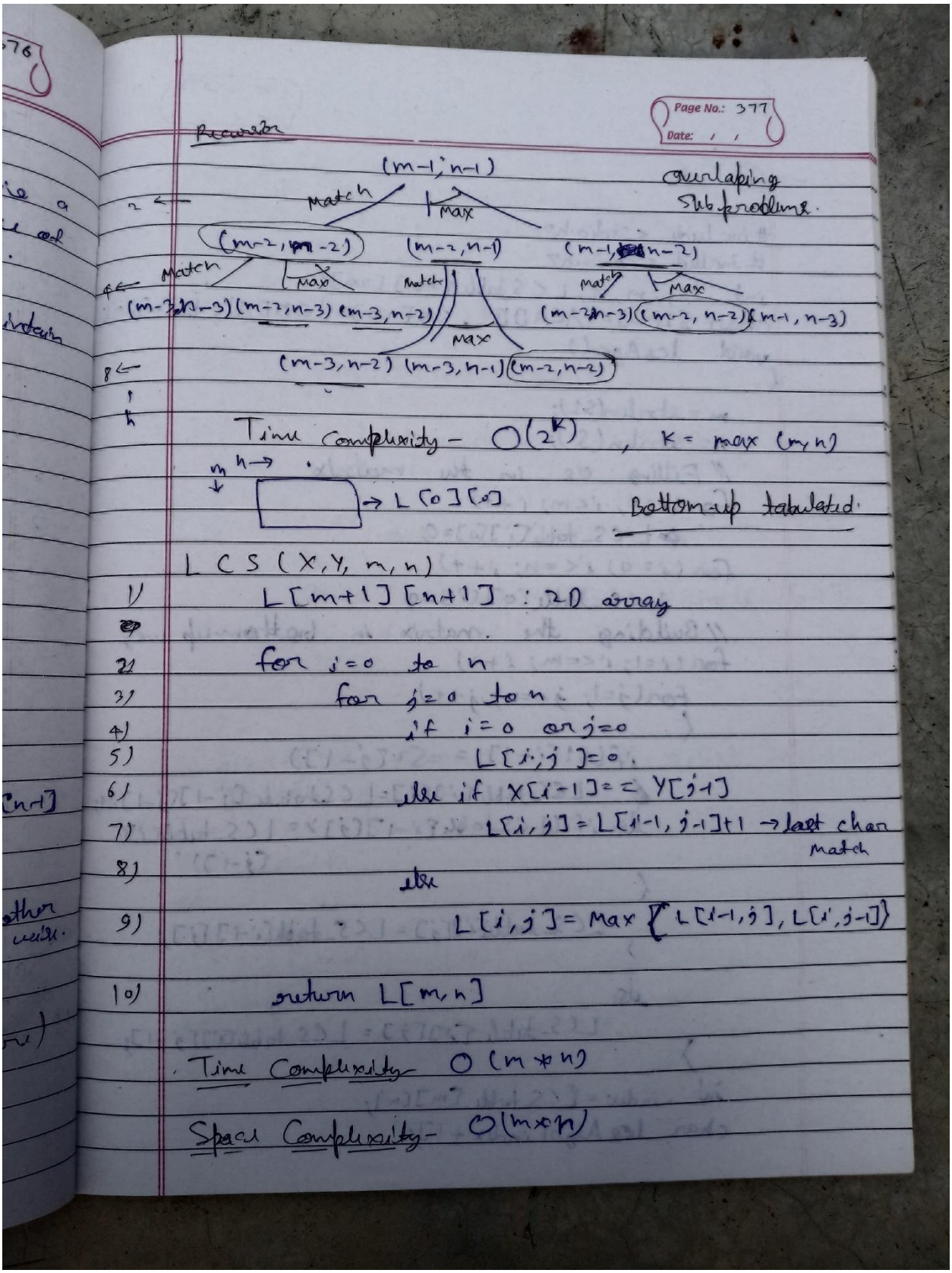
```
int fib(n)
{
    for(i = 2; i <= n; i++)
        f[i] = f[i-1] + f[i-2]
    return f[n]
}
```

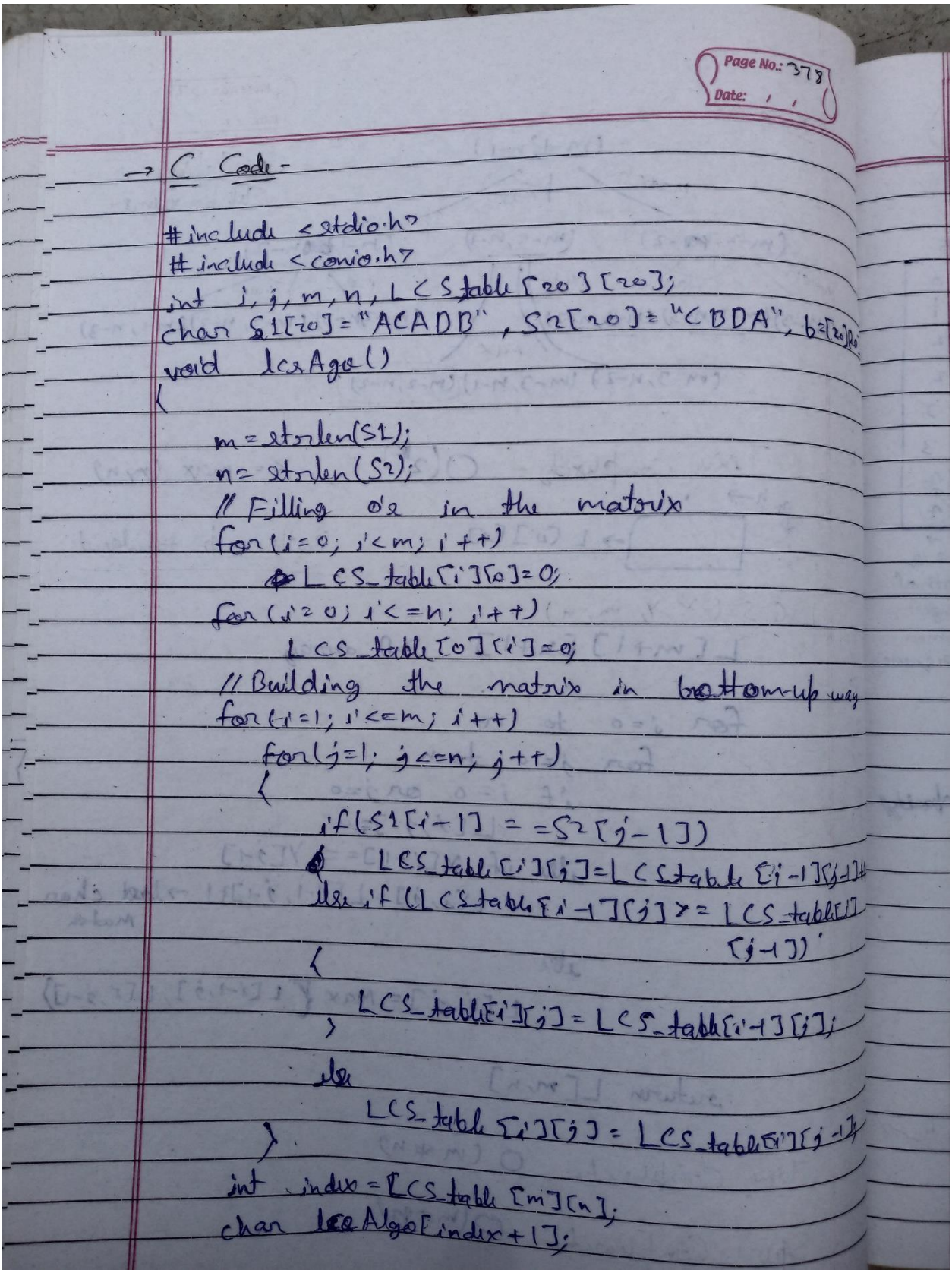
Time Complexity - O(n)

Space Complexity - O(n)



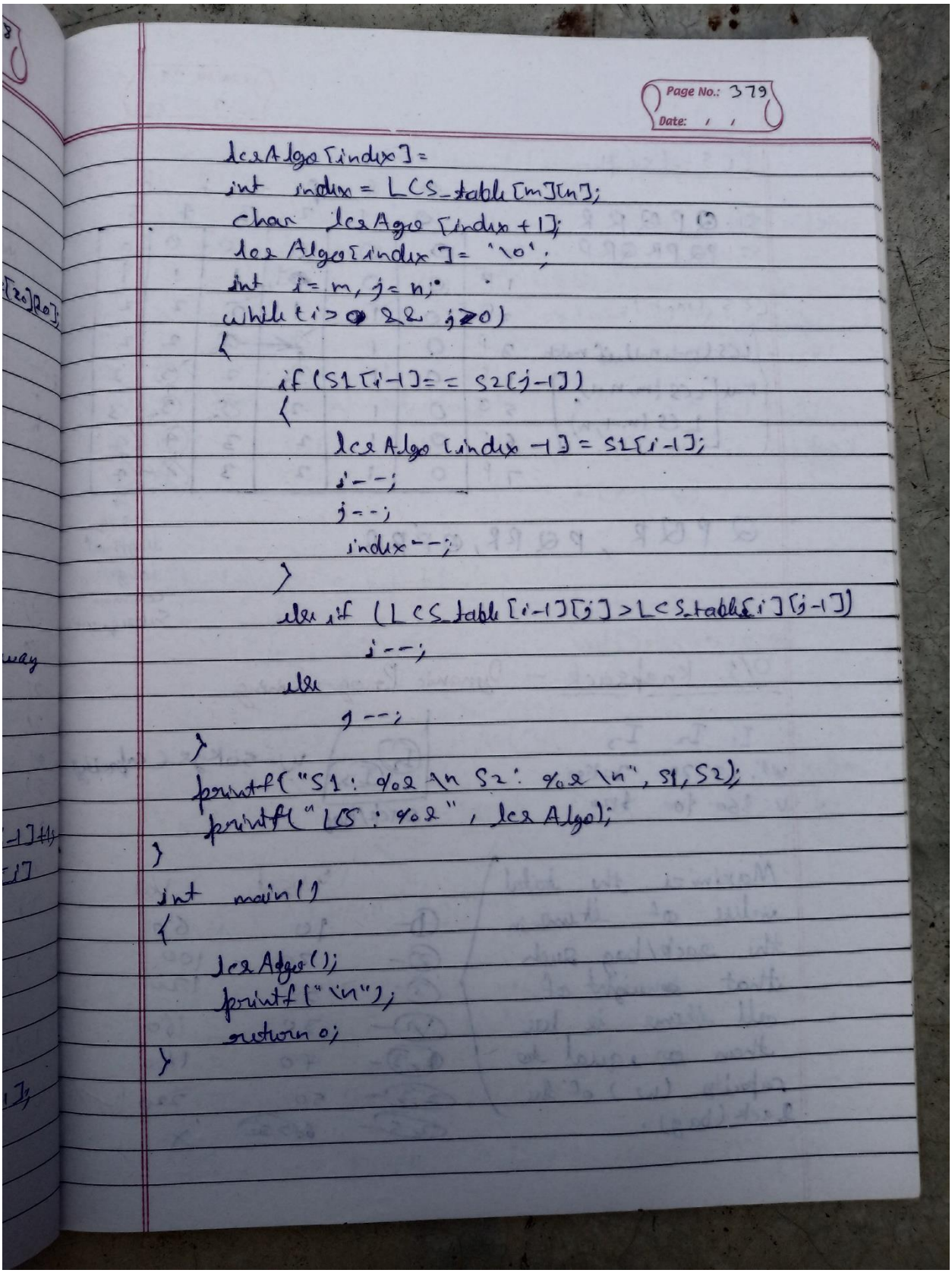

Page No.: 376
Date: / /
Longest Comman Subsequence (LCS):-
Eg- S1 : A B C D G H
S2 : A E D F H R
ADH is a subsequence of S1 & S2.
Subsequence- sequence of character (maintain only order)
Application-
1-Plagiarism check-
2-Genomics etc.
S1 : X[0,1,2,---m-1] → len m
S2 : Y[0,1,2,---n-1] → len n
Recursive formula-(Derivation)-
LCS(m-1,n-1)
LCS(m-2, n-2)+1 if X[m-1] = Y[n-1]
a) X: AGGTAB match
Y: AXTGXB
b) X: ABCDHG not match
Y: BCDANH
max { LCS(m-1, n-2), LCS(m-2, n-1) } either will
(optimal sub struct


Page No.: 377
Date:
Recursion
(m-1, n-1)
match
Max
overlaping subproblems.
(m-2, m-2)
(m-2, n-1)
(m-1, n-2)
Match
Max
Match
match
Max
(m-3, n-3) (m-2, n-3) (m-3, n-2)
Max
(m-2, n-3) (m-2, n-2) (m-1, n-3)
(m-3, n-2) (m-3, n-1) (m-2, n-2)
h
Time complexity - O(2^k) , k = max (m, n)
m
n →
→ L[0][0]
Bottom-up tabulated
LCS (X, Y, m, n)
L[m+1][n+1] : 2D array
for i=0 to n
for j=0 to n
if i=0 or j=0
L[i,j]=0.
else if X[i-1] == Y[j-1]
L[i,j] = L[i-1, j-1]+1 → last char match
else
L[i,j] = max {L[i-1,j], L[i,j-1]}
return L[m,n]
Time Complexity- O(m*n)
Space Complexity- O(m×n)

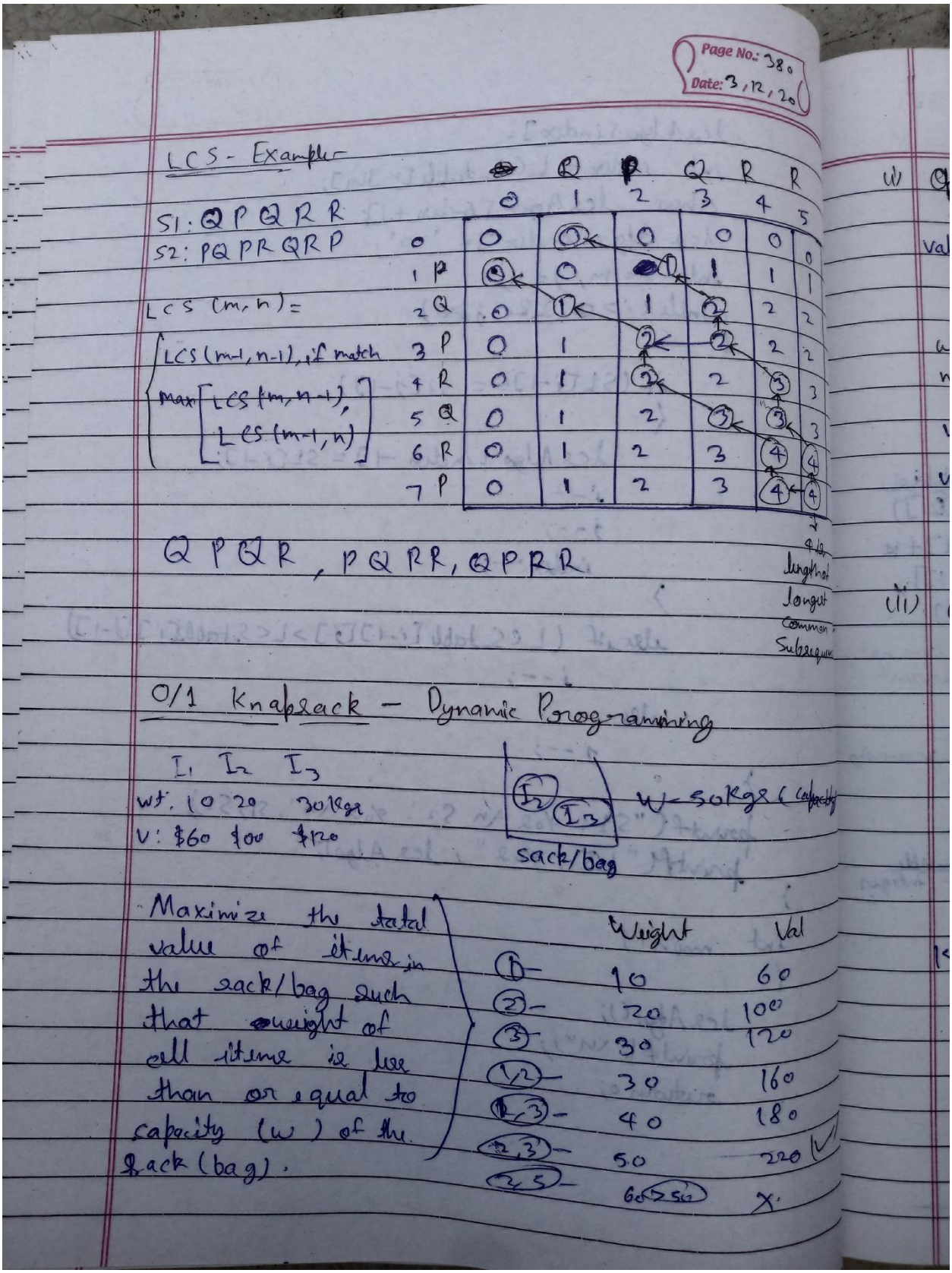


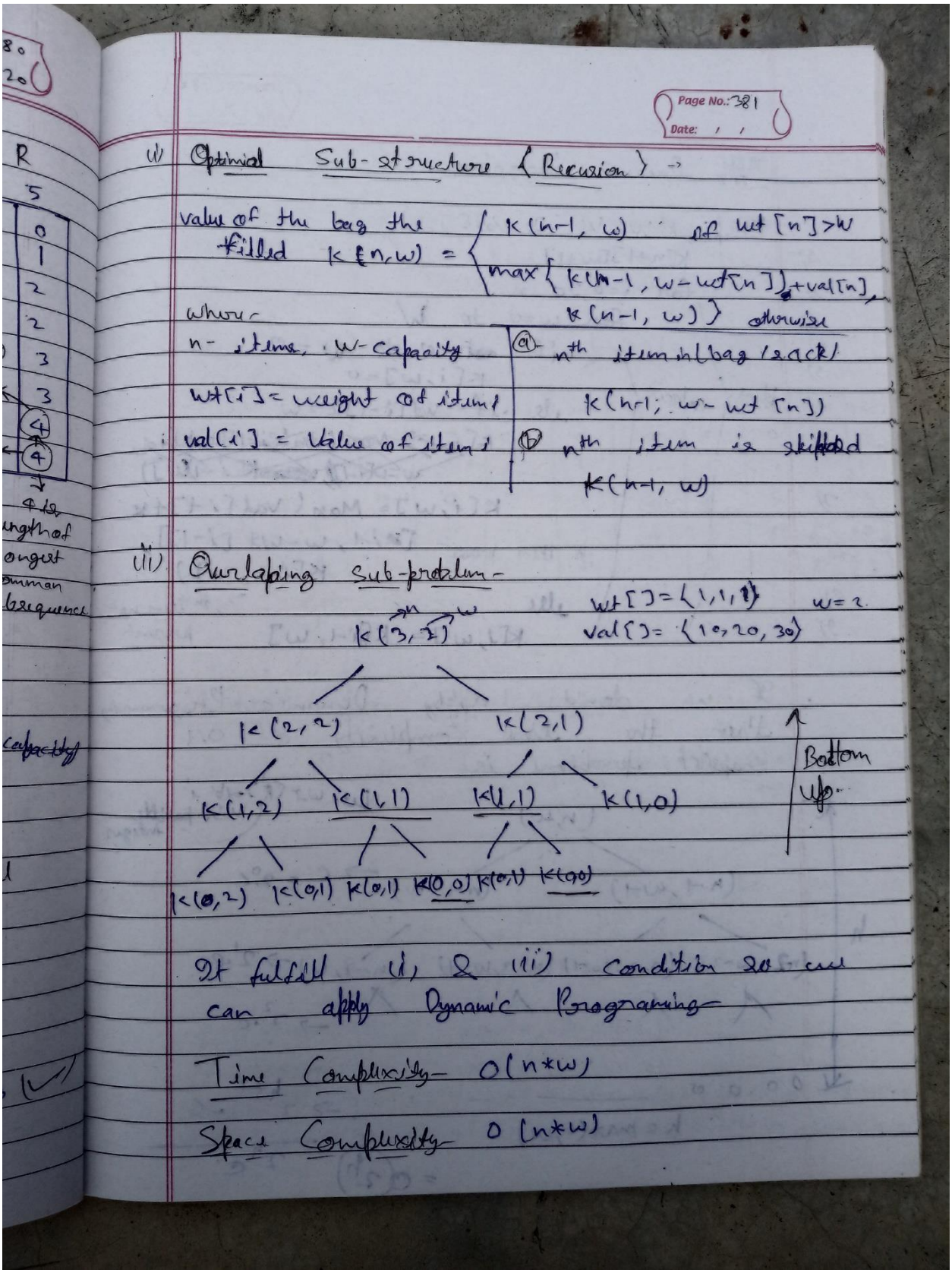
→ C Code -

```
#include <stdio.h>
#include <conio.h>
int i, j, m, n, LCS_table[20][20];
char S1[20] = "ACADB", S2[20] = "CBDA", b[20][20];
void lcsAlgo()
{
    m = strlen(S1);
    n = strlen(S2);
    // Filling 0's in the matrix
    for(i=0; i<m; i++)
        LCS_table[i][0] = 0;
    for(i=0; i<=n; i++)
        LCS_table[0][i] = 0;
    // Building the matrix in bottom-up way
    for(i=1; i<=m; i++)
        for(j=1; j<=n; j++)
        {
            if(S1[i-1] == S2[j-1])
                LCS_table[i][j] = LCS_table[i-1][j-1]+
            else if (LCS_table[i-1][j] >= LCS_table[i]
                                          [j-1])
            {
                LCS_table[i][j] = LCS_table[i-1][j];
            }
            else
                LCS_table[i][j] = LCS_table[i][j-1];
        }
    int index = LCS_table[m][n];
    char lcsAlgo[index+1];
```



```
lcsAlgo[index] =
int index = LCS_table[m][n];
char lcsAgo[index+1];
lcsAlgo[index] = '\0';
int i=m, j=n;
while(i>0 && j>0)
{
    if(S1[i-1] == S2[j-1])
    {
        lcsAlgo[index-1] = S1[i-1];
        i--;
        j--;
        index--;
    }
    else if (LCS_table[i-1][j] > LCS_table[i][j-1])
        i--;
    else
        j--;
}
printf("S1: %s \n S2: %s \n", S1, S2);
printf("LCS: %s", lcsAlgo);
}
int main()
{
    lcsAlgo();
    printf("\n");
    return 0;
}
```


Page No.: 380
Date: 3/12/20
LCS - Example
S1: Q P Q R R
S2: P Q P R Q R P
LCS (m, n) =
{ LCS(m-1, n-1), if match
{ max [LCS(m, n-1), LCS(m-1, n)]
Q P Q R , P Q R R , Q P R R
Longest Common Subsequence
0/1 Knapsack - Dynamic Programming
I1 I2 I3
wt. 10 20 30 Kgs
V: $60 $100 $120
W = 50 Kgs (Capacity)
sack/bag
Maximize the total value of items in the sack/bag such that weight of all items is less than or equal to capacity (w) of the sack (bag).
Weight Val
1 - 10 60
2 - 20 100
3 - 30 120
1,2 - 30 160
1,3 - 40 180
2,3 - 50 220
2,5 - 60 > 50 X



(i) Optimal Sub-structure {Recursion} →

value of the bag the filled $k(n,w)$ =
- $k(n-1, w)$ if $wt[n] > w$
- $\max\{k(n-1, w-wt[n]) + val[n],\ k(n-1, w)\}$ otherwise

where -
n - items, w - capacity
wt[i] = weight of item i
val[i] = value of item i

(a) n^{th} item in bag / sack
$k(n-1, w-wt[n])$

(b) n^{th} item is skipped
$k(n-1, w)$

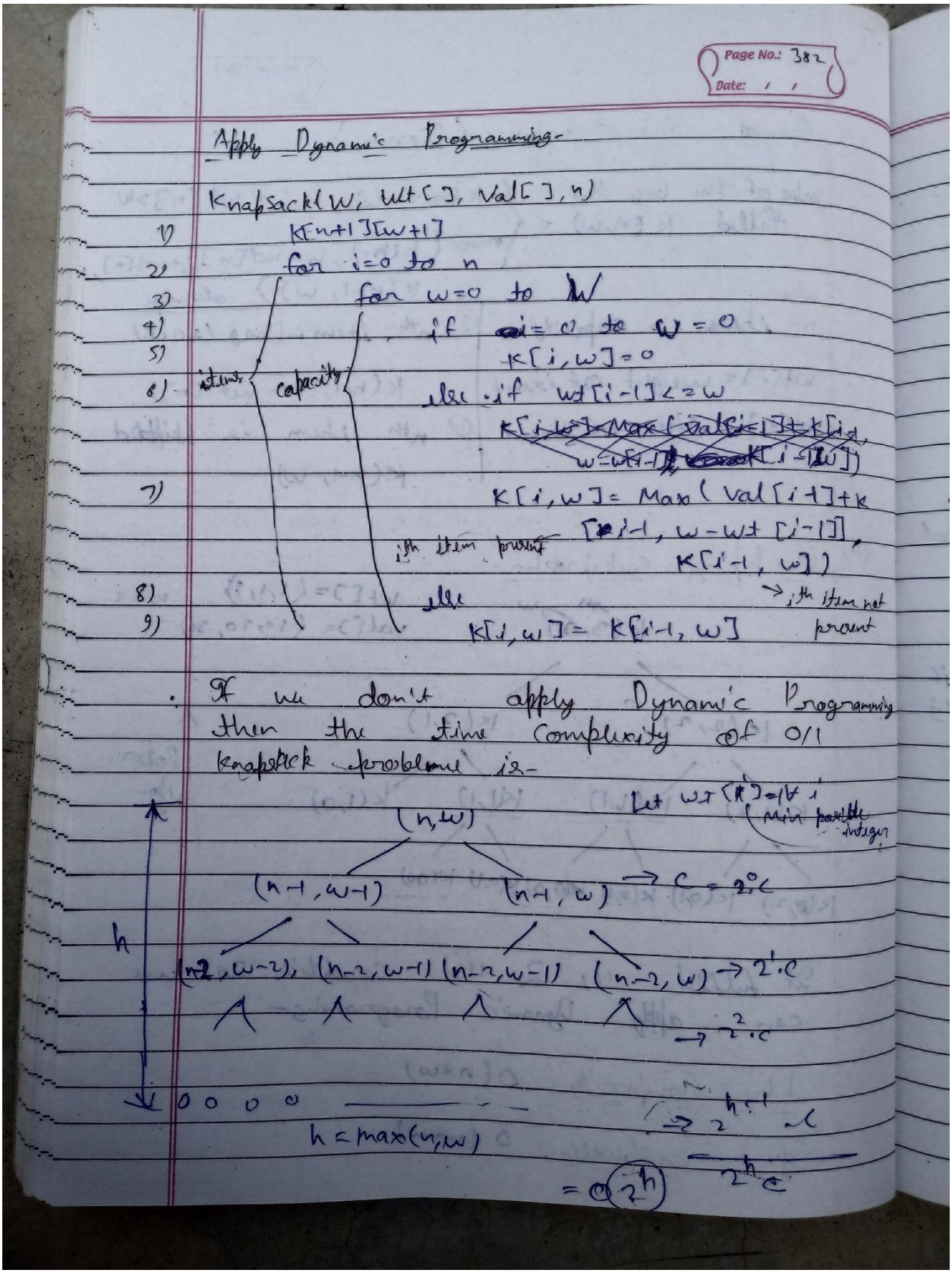
(ii) Overlaping Sub-problem -

wt[] = {1, 1, 1} w = 2
val[] = {10, 20, 30}

It fulfill (i) & (ii) condition so we can apply Dynamic Programming

Time Complexity - $O(n*w)$

Space Complexity - $O(n*w)$


Page No.: 382
Date: / /
Apply Dynamic Programming-
Knapsack(W, wt[], val[], n)
1) K[n+1][w+1]
2) for i=0 to n
3) for w=0 to W
4) if
5) K[i,w]=0
6) items
capacity
else if wt[i-1]<=w
K[i,w]=Max(val[i-1]+K
7) [i-1, w-wt[i-1]],
ith item present
K[i-1, w])
8) else
ith item not present
9) K[i,w]=K[i-1,w]
If we don't apply Dynamic Programming then the time complexity of 0/1 knapsack problem is-
Let wt[i]=1 ∀ i
(min possible integer)
(n,w)
(n-1,w-1)
(n-1,w)
→ c = 2^0 c
h
(n-2,w-2), (n-2,w-1) (n-2,w-1) (n-2,w)
→ 2^1 c
→ 2^2 c
h = max(n,w)
→ 2^(h+1) c
= O(2^h)
2^h c

Page No.: 383
Date: / /

Example of 0/1 Knapsack –

$$K(n,w) = \begin{cases} K(n-1,w) & \text{if } wt[n] > w \\ \max\{K(n-1,w),\ K(n-1,w-wt[n])+val[n]\} & \text{otherwise} \end{cases}$$

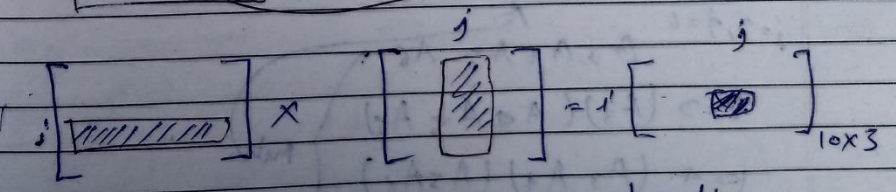
W = 7, Capacity of knapsack.

item	wt	val	w=0	w=1	w=2	w=3	w=4	w=5	w=6	w=7
n=0	0	0	0	0	0	0	0	0	0	0
n=1	1	1	0	1	1	1	1	1	1	1
n=2	3	4	0	1	1	4	5	5	5	5
n=3	4	5	0	1	1	4	5	6	6	9
n=4	5	7	0	1	1	4	5	7	8	9

input

Matrix Chain Multiplication

$A_{10\times30}\ B_{30\times5} = T_{10\times5}$ ⇒ number of operations = $10\times30\times5$



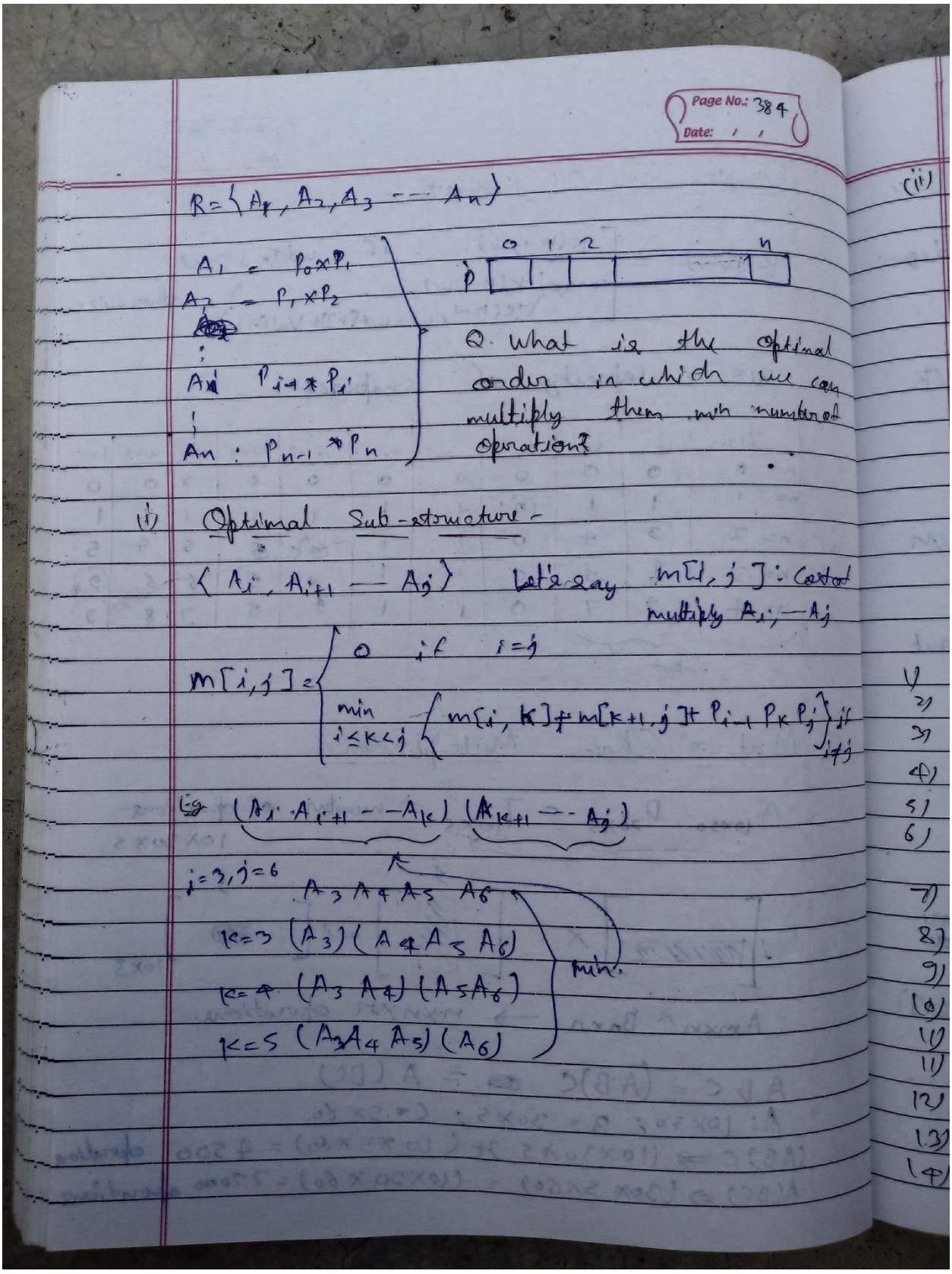
$A_{m\times n} \times B_{n\times r} \rightarrow m\times n\times r$ operations.

$ABC = (AB)C = A(BC)$

A: 10×30; B: 30×5; C: 5×60

$(AB)C \Rightarrow (10\times30\times5) + (10\times5\times60) = 4500$ operations

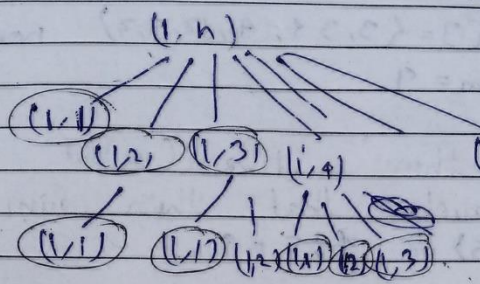
$A(BC) \Rightarrow (30\times5\times60) + (10\times30\times60) = 27000$ operations


Page No.: 384
Date: / /
R = {A1, A2, A3 -- An}
A1 = P0 x P1
A2 = P1 x P2
Ai : Pi-1 * Pi
An : Pn-1 * Pn
0 1 2 n
p
Q. What is the optimal order in which we can multiply them with minimal operations?
(i) Optimal Sub-structure -
{Ai, Ai+1 -- Aj} Let's say m[i, j] : Cost of multiply Ai -- Aj
m[i,j] = 0 if i=j
min i≤k<j { m[i,k] + m[k+1,j] + Pi-1 Pk Pj } if i≠j
Eg (A1 A2 -- Ak) (Ak+1 -- Aj)
i=3, j=6
A3 A4 A5 A6
k=3 (A3)(A4 A5 A6)
k=4 (A3 A4)(A5 A6)
k=5 (A3 A4 A5)(A6)
min.

Page No.: 385
Date: / /

(ii) Overlapping Sub-problems

m[1, n]

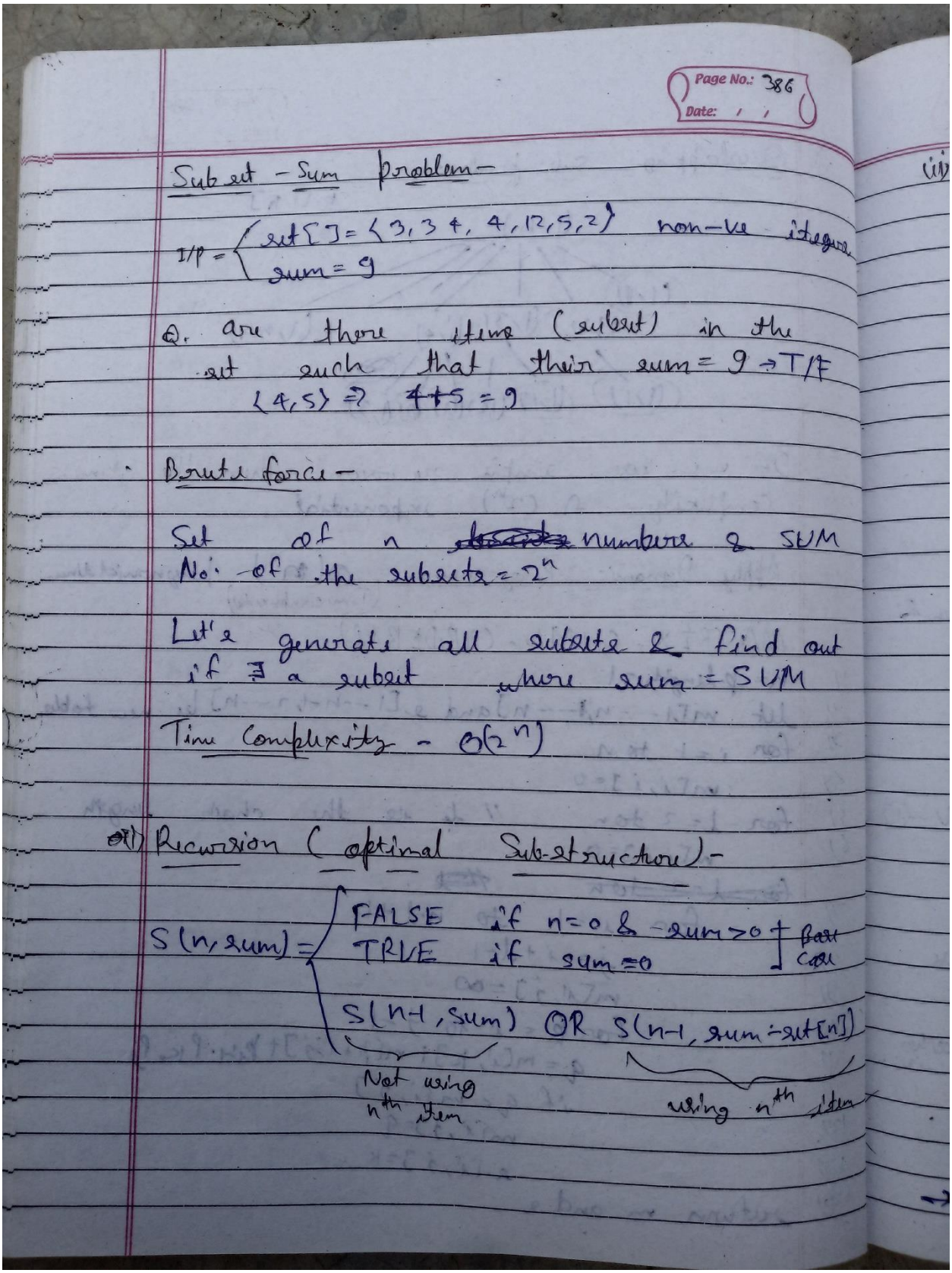


If we use simple recursion then the time-complexity - $\Omega(2^n)$ exponential

Apply Dynamic Programming $O(n^3)$ polynomial time (Time complexity)

MATRIX-CHAIN-ORDER(P)

```
1)  n = p.length - 1
2)  let m[1...n, 1...n] and s[1...n-1, 2...n] be new tables
3)  for i = 1 to n
4)      m[i, i] = 0
5)  for l = 2 to n        // l is the chain length
6)      m[i, i] = 0
    ~~for l = 2 to n~~ ~~do~~
7)      for i = 1 to n-l+1
8)          j = i + l - 1
9)          m[i, j] = ∞
10)         for k = i to j - 1
11)             q = m[i, k] + m[k+1, j] + p_{i-1} . p_k . p_j
12)             if q < m[i, j]
13)                 m[i, j] = q
14)                 s[i, j] = k
15) return m and s
```



Page No.: 386
Date: / /

Subset - Sum problem -

I/P = { set[] = {3, 34, 4, 12, 5, 2} non-ve integers
sum = 9

Q. are there items (subset) in the set such that their sum = 9 → T/F
{4, 5} ⇒ 4+5 = 9

Brute force -

Set of n numbers & SUM
No. of the subsets = 2^n

Let's generate all subsets & find out if ∃ a subset where sum = SUM

Time Complexity - $O(2^n)$

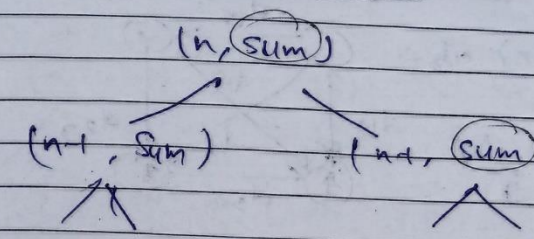
Recursion (optimal Sub-structure) -

S(n, sum) = { FALSE if n=0 & sum>0 } Base case
TRUE if sum=0
S(n-1, sum) OR S(n-1, sum - set[n])

S(n-1, sum) → Not using n^{th} item
S(n-1, sum - set[n]) → using n^{th} item

Page No.: 387
Date: / /

(ii) Overlapping Sub-problem -



Note - There may be some case where there is no overlapping sub problems then D.P. = Simple recursion.

Using DP - O(n*SUM).

for i = 0 to n
  for s = 0 to SUM
    S(i, s) = recursion formula
MatrixTable[n+1, SUM+1]

Time complexity in worst Case -

Brute face - O(2^n)

DP = O(n*SUM) if SUM = 2^n then DP is worst case more than brute force.

Note - If SUM is << 2^n use D.P.

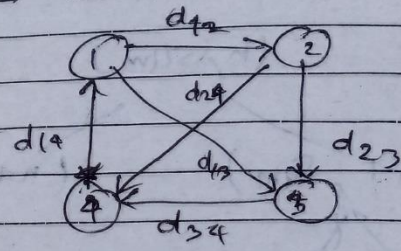
If SUM ≈ 2^n use Brute Force.

→ Cannot guarantee DP is better than brute force in worst case.

Page No.: 388
Date: 7, 12, 20

Traveling Salesman Problem (Graph Theory).

{1, 2, 3, — n} city
source: 1
(overall path-length dist minimum).



- **Brute-force -**

Permutation (ordering of n cities)
{1, 2, 3, 4} — {1, 2, 4, 3} — —
— n! permutations.

generate n! perm → Compute the path-length
pick the minimal.

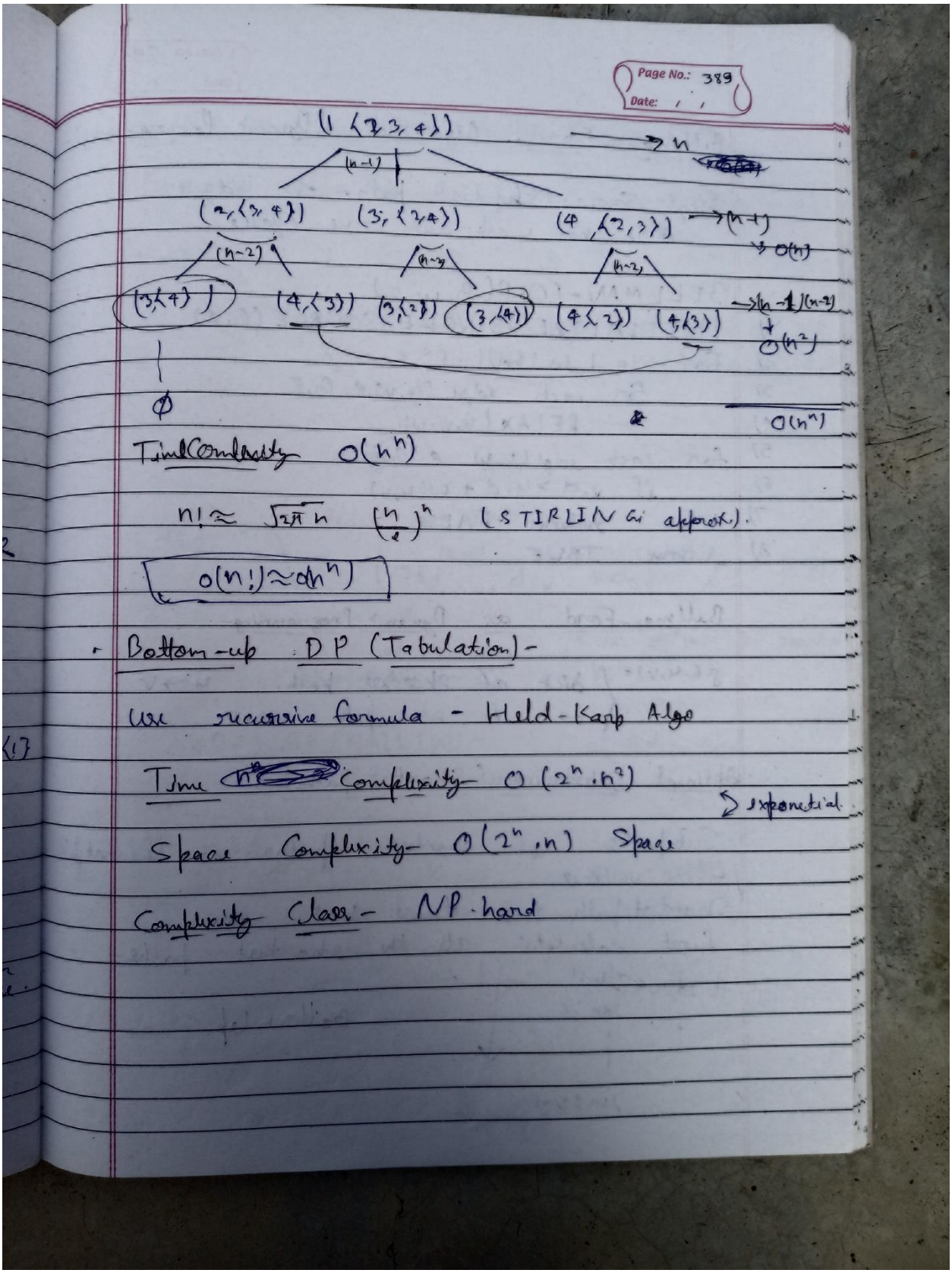
O(n!) Time Complexity

- **Recursive -**

V = {1, 2, 3, — n}
src = 1
d_{ij} = dist(i, j)
$S \subseteq \{2, 3, 4 — n\} = V - \{1\}$

$$TSP(i, S) = \begin{cases} d_{i1} & \text{if } S \neq \phi \rightarrow \text{Base case} \\ \min_{k \in S} \{ d_{ik} + TSP(k, S - \{k\}) \} & \text{otherwise} \end{cases}$$

↑ min cost/len path from i to every vertex in S & back to 1 = src.



Time Complexity $O(n^n)$

$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$ (STIRLING approx.)

$O(n!) \approx O(n^n)$

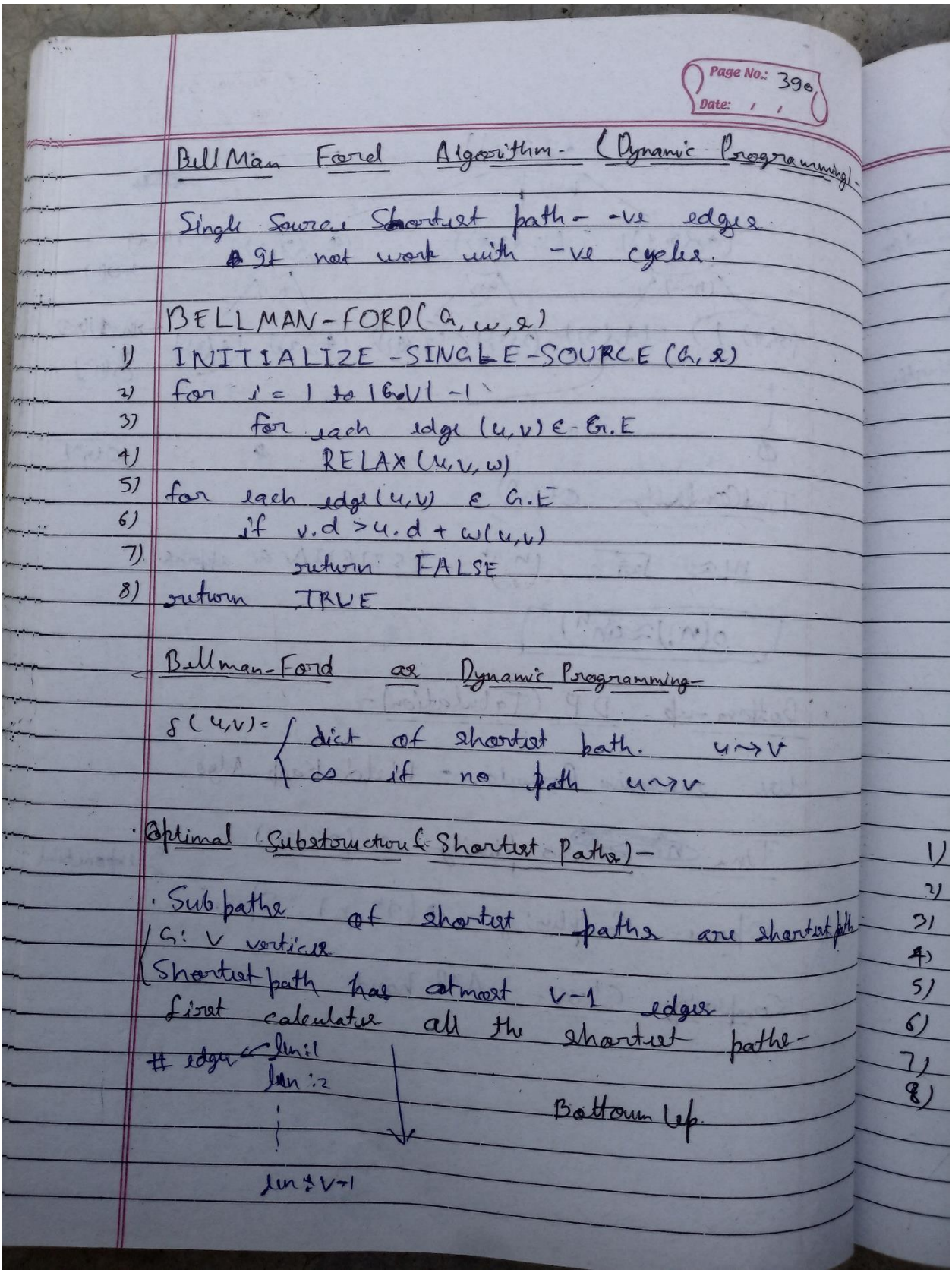
- Bottom-up DP (Tabulation) -

Use recursive formula - Held-Karp Algo

Time Complexity - $O(2^n \cdot n^2)$ > exponential

Space Complexity - $O(2^n \cdot n)$ Space

Complexity Class - NP-hard



Page No.: 390
Date: / /

Bellman Ford Algorithm - (Dynamic Programming)

Single Source Shortest path - -ve edges.
* It not work with -ve cycles.

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

Bellman-Ford as Dynamic Programming -

$$\delta(u,v) = \begin{cases} \text{dist of shortest path. } u \to v \\ \infty \text{ if no path } u \to v \end{cases}$$

Optimal Substructure (Shortest Paths) -

- Sub paths of shortest paths are shortest path

G: V vertices
Shortest path has atmost V-1 edges
First calculate all the shortest paths -

edges: len:1, len:2, ..., len: V-1

Bottom up.

1)
2)
3)
4)
5)
6)
7)
8)

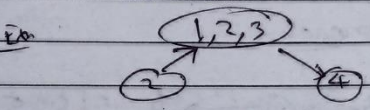
Page No.: 391
Date: / /

Floyd- Warshall Algorithm: (Dynamic Programming)

Find All pair shortest paths. $V = \{1, 2, 3, \ldots n\}$

Recursive —

$d_{ij}^{(k)}$ = dist of shortest path $i \rightsquigarrow j$ with $\{1, 2, 3, \ldots k\}$ as potential intermediate vertices



$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \rightarrow \text{if } k = 0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1 \end{cases}$$

$d_{ij}^{(k-1)}$: k is not in the shortest path $i \rightsquigarrow j$

$d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$: k is part of shortest path $i \rightsquigarrow j$

FLOYD-WARSHALL(W) — (Bottom-up)

```
1) n = W.rows
2) D^(0) = W
3) for k = 1 to n
4)     let D^(k) = (d_ij^(k)) be a new nxn matrix
5)     for i = 1 to n
6)         for j = 1 to n
7)             d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
8) return D^(n)
```

Page No.: 392
Date: / /

Multi-Stage Graph-

Problem- Minimization Optimization

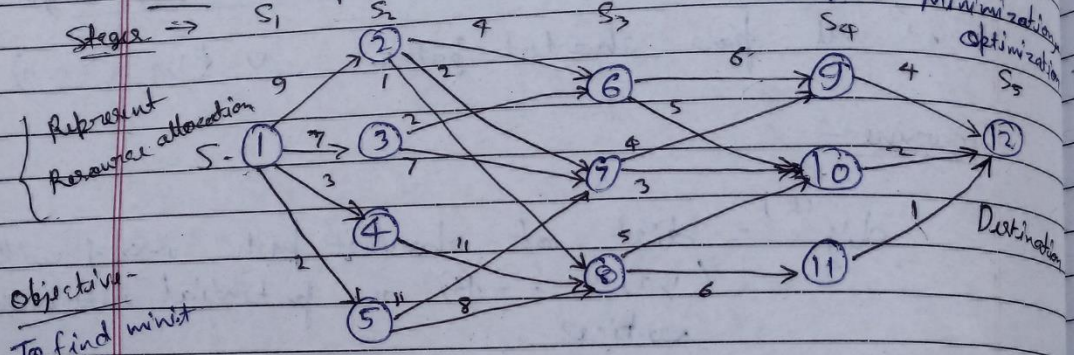
Stages ⇒ S_1 S_2 S_3 S_4 S_5

Represent Resource allocation

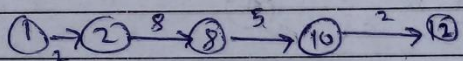
Source

Destination

Objective- To find minimum

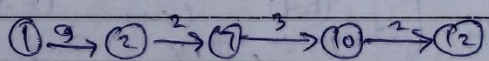


- Apply Greedy Approach-



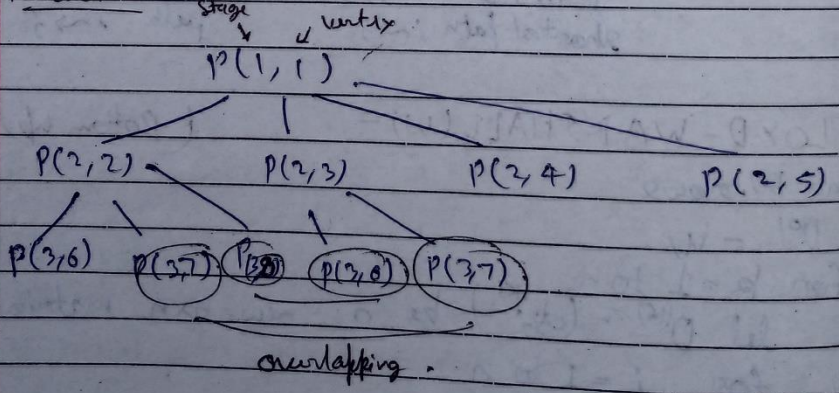
$= 10+5+2 = 17.$

But Minimum Path-



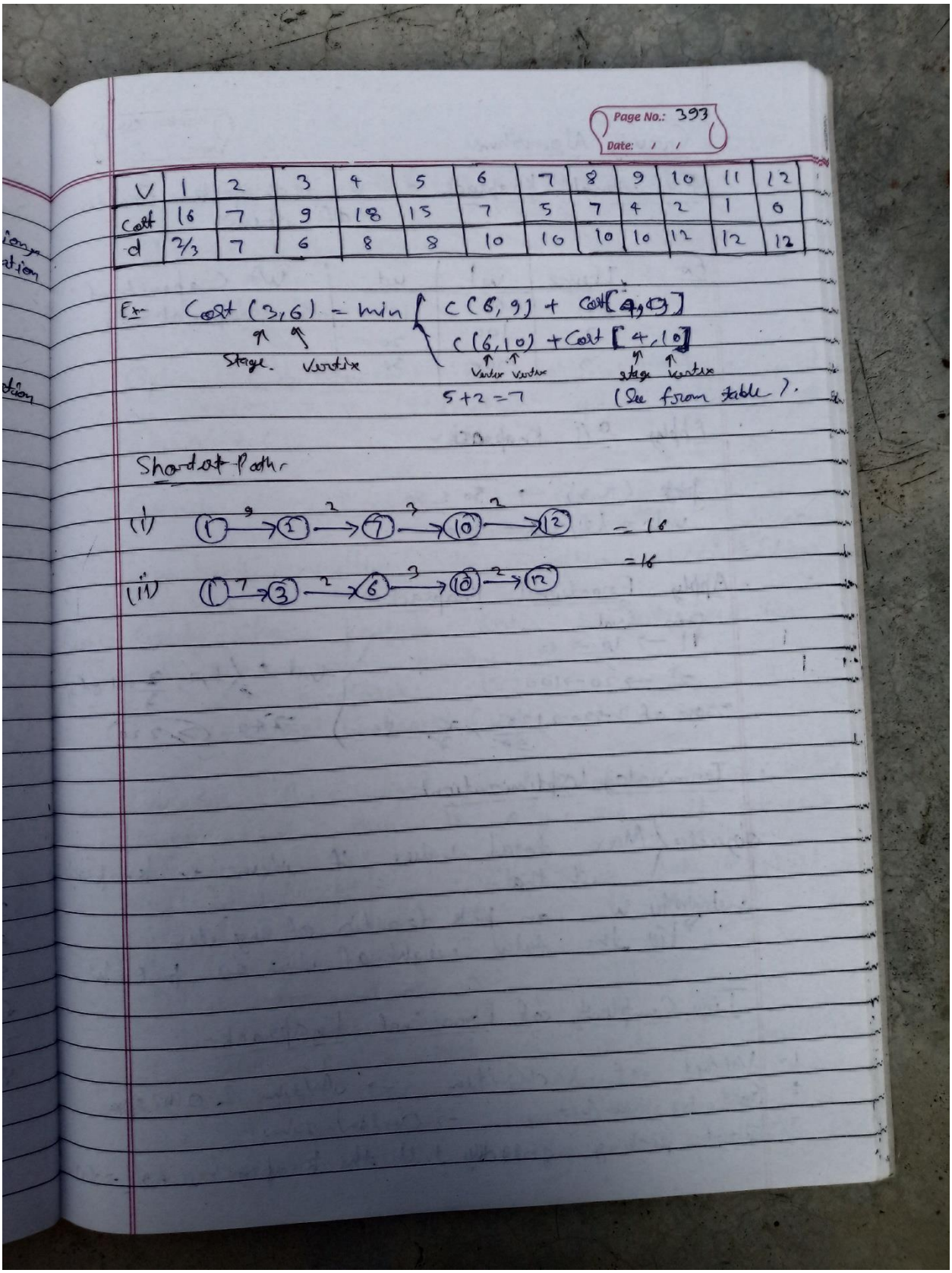
$= 16.$

- Recursion-



- Apply DP.-

$$Cost(i, j) = \min_{\langle j,l \rangle \in E,\ l \in S_{i+1}} \{ c(j, l) + Cost(i+1, l) \}$$



Page No.: 393
Date: / /

V	1	2	3	4	5	6	7	8	9	10	11	12
Cost	16	7	9	18	15	7	5	7	4	2	1	0
d	2/3	7	6	8	8	10	10	10	10	12	12	12

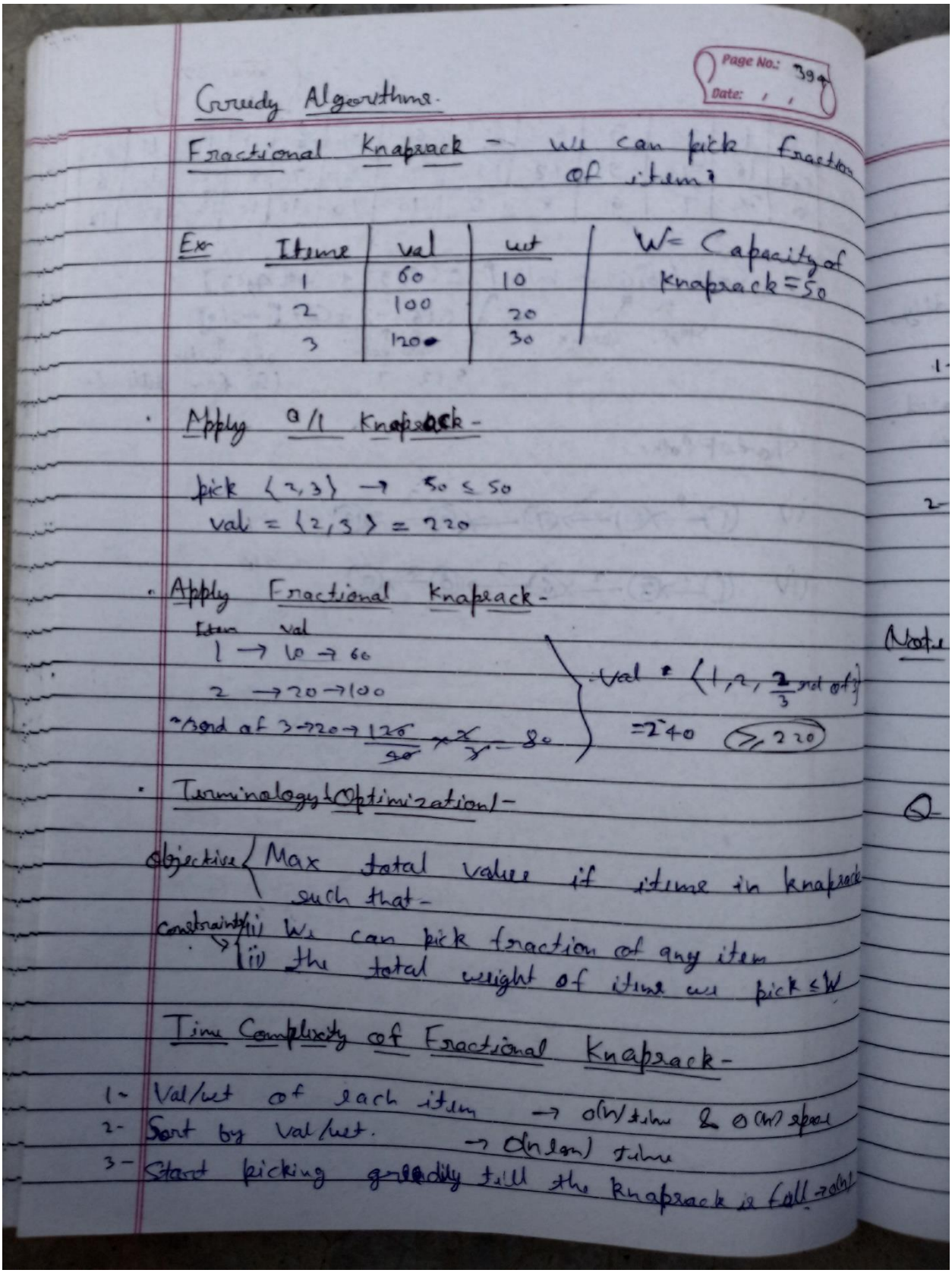
Ex- Cost (3,6) = min { c(6,9) + Cost[9,0]
c(6,10) + Cost[4,10]

Stage. Vertex
Vertex Vertex
stage Vertex
5+2=7
(See from table).

Shortest Path.

(i) 1 →(9) 4 →(2) 7 →(2) 10 →(2) 12 = 16

(ii) 1 →(7) 3 →(2) 6 →(2) 10 →(2) 12 = 16



Page No.: 394
Date: / /

Greedy Algorithms.

Fractional Knapsack - We can pick fraction of items

Ex	Items	val	wt
	1	60	10
	2	100	20
	3	120	30

W = Capacity of knapsack = 50

- Apply 0/1 Knapsack -

pick {2,3} → 50 ≤ 50
val = {2,3} = 220

- Apply Fractional Knapsack -

Item val
1 → 10 → 60
2 → 20 → 100
3 → 20 → 120 → 80

Val = {1, 2, 2/3 rd of 3}
= 240 (≥ 220)

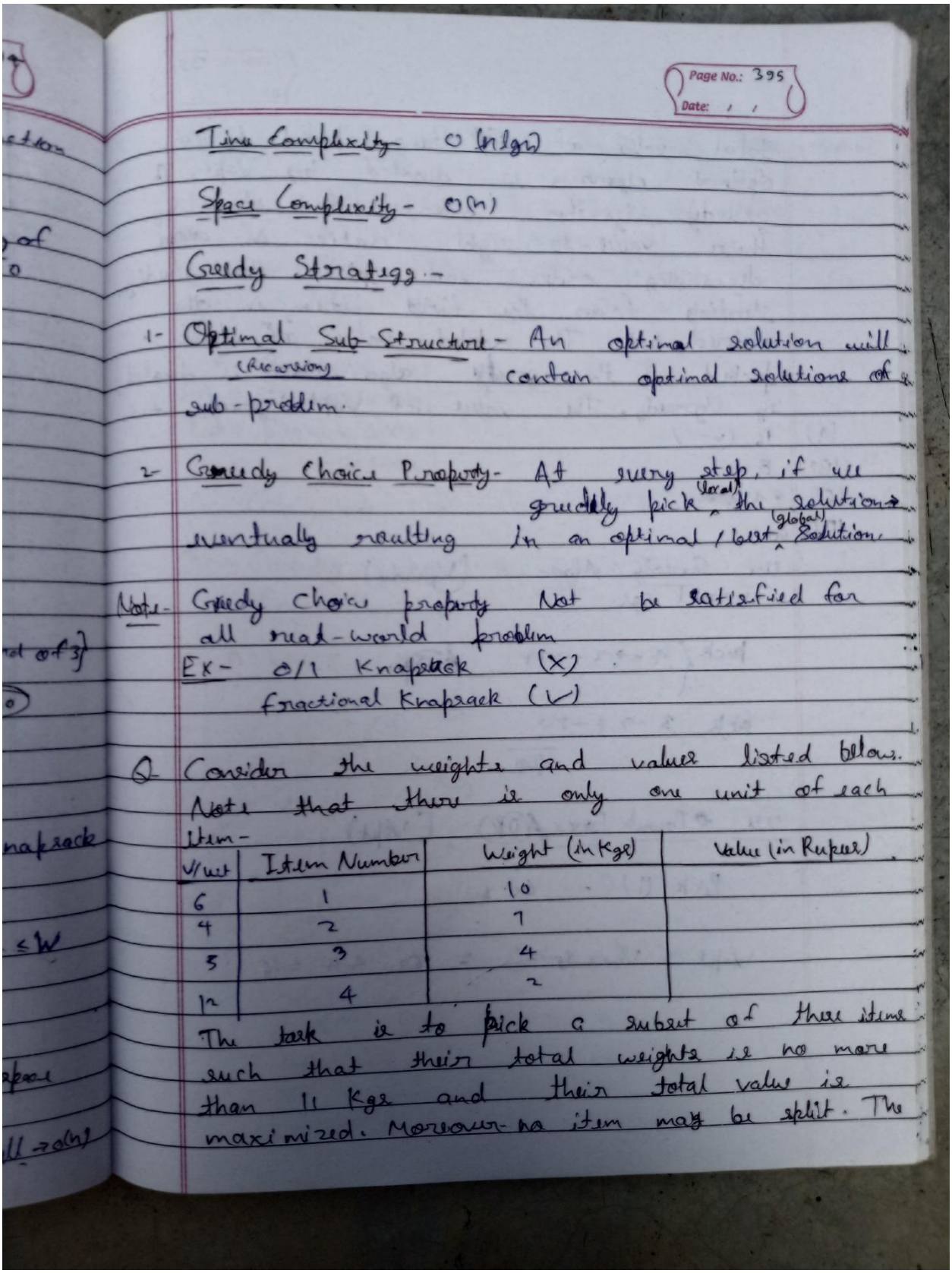
Note

- Terminology (Optimization) -

Objective { Max total value if items in knapsack such that -
Constraints { i) We can pick fraction of any item
ii) the total weight of items we pick ≤ W

Time Complexity of Fractional Knapsack -

1- Val/wt of each item → O(N) time & O(N) space
2- Sort by val/wt. → O(N log N) time
3- Start picking greedily till the knapsack is full → O(N)



Page No.: 395
Date: / /

Time Complexity - O(n log n)

Space Complexity - O(n)

Greedy Strategy -

1- Optimal Sub-Structure (Recursion) - An optimal solution will contain optimal solutions of sub-problem.

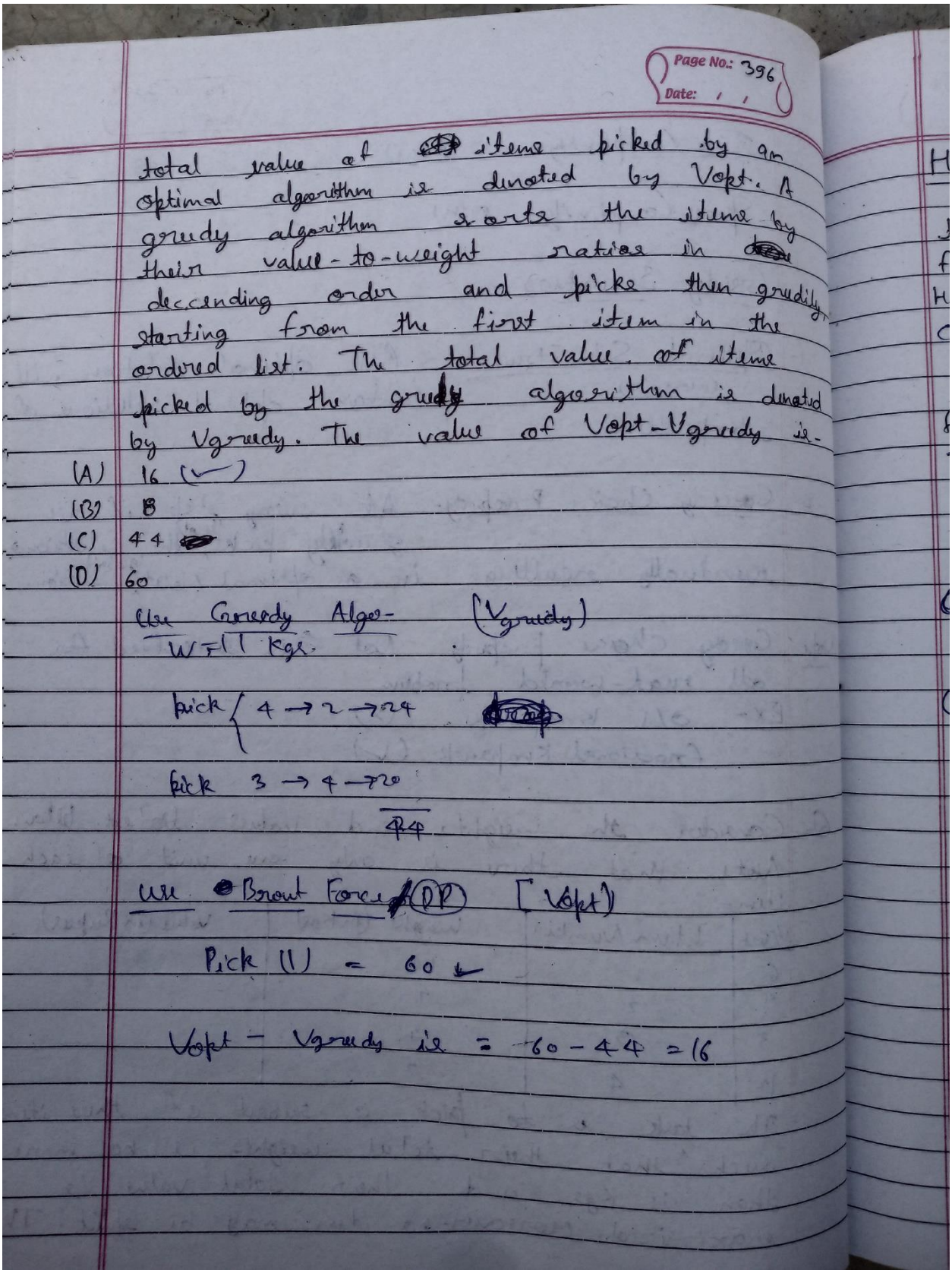
2- Greedy Choice Property - At every step, if we greedily pick (local) the solutions eventually resulting in an optimal / best (global) solution.

Note - Greedy choice property Not be satisfied for all real-world problem
Ex - 0/1 Knapsack (X)
Fractional Knapsack (✓)

Q- Consider the weights and values listed below. Note that there is only one unit of each item -

V/wt	Item Number	Weight (in Kgs)	Value (in Rupees)
6	1	10	
4	2	7	
5	3	4	
12	4	2	

The task is to pick a subset of these items such that their total weights is no more than 11 Kgs and their total value is maximized. Moreover no item may be split. The



Page No.: 396
Date: / /

total value of items picked by an optimal algorithm is denoted by Vopt. A greedy algorithm sorts the items by their value-to-weight ratios in descending order and picks them greedily starting from the first item in the ordered list. The total value of items picked by the greedy algorithm is denoted by Vgreedy. The value of Vopt - Vgreedy is-

(A) 16 ✓
(B) 8
(C) 44
(D) 60

Use Greedy Algo- (Vgreedy)
W = 11 Kgs.

pick { 4 → 2 → 24

pick 3 → 4 → 20

44

use Brout Force (DP) [Vopt]

Pick (1) = 60 ✓

Vopt - Vgreedy is = 60 - 44 = 16

Page No.: 397
Date: / /

Huffman Coding- (Data Compression) (Greedy Method)

Text →	A	B	C	D	E	F	No. of char in text
freq →	5	25	7	15	4	12	= 68
Huffman Code →	1011	11	100	01	1010	00	→ variable length representation
	4	2	3	2	4	2	← length)

Case 1- binary

6 Characters.

Fix length representation

Total Space = 68×3

= 204 bits

2 bits = 2^2 = 4 Char.

00, 01, 10, 11

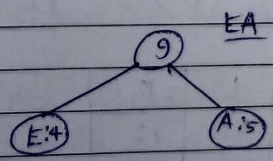
A D C D

3 bits = 2^3 = 8 char.

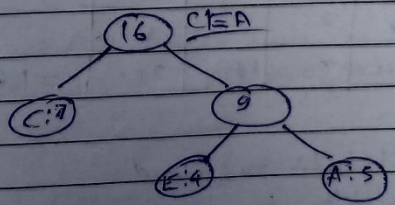
Q- Can we use fewer bits to store this info?

→ Greedily picked the char that occur least frequently

① B:25; C:7; D:15; F:12; EA:9

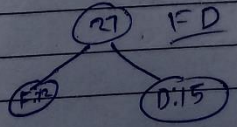


②

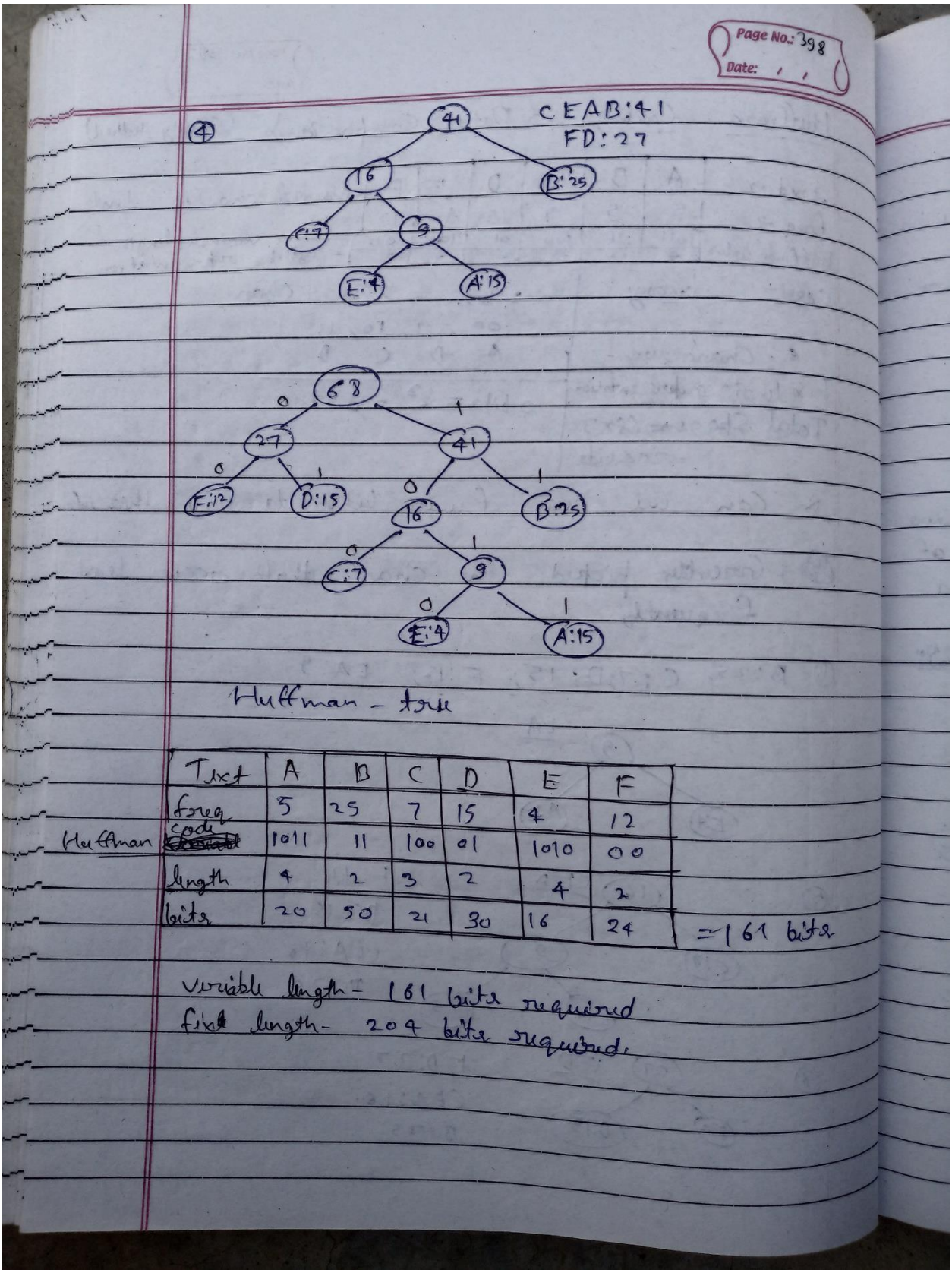


F:12
D:15
CEA:16
B:25

③



FD:27
CEA:16
B:25



Text	A	B	C	D	E	F
freq	5	25	7	15	4	12
Huffman code	1011	11	100	01	1010	00
length	4	2	3	2	4	2
bits	20	50	21	30	16	24

= 161 bits

variable length - 161 bits required
fixed length - 204 bits required.

Page No.: 399
Date: / /

Time Complexity -

n
↓
n-1
↓
n-2
↓
1
↓
1

n time
→ pick 2 least frequent characters

(i) When use array -

Time Complexity - $O(n^2)$
Space complexity - $O(n)$

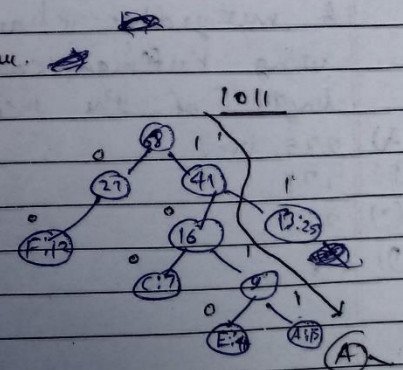
(ii) When use Min Heap -

Time Complexity - $O(n \log n)$
Space Complexity - $O(n)$

How to Decode -

text - A B B C D
encode - 10111111 0001
use Huffman tree.
decode - A B B C D

Ex -
10111111 0001



Page No.: 400
Date: / /

Greedy Strategy for Huffman coding-

(a) Greedy Choice Property - 2 least frequent character

(b) Optimal Sub Structure - Every subtree of huffman tree difines an optimal huffman tree.

Note- Huffman encoding is Lossless Data Compression Algorithm (And Jpg use Fourier Transformation)

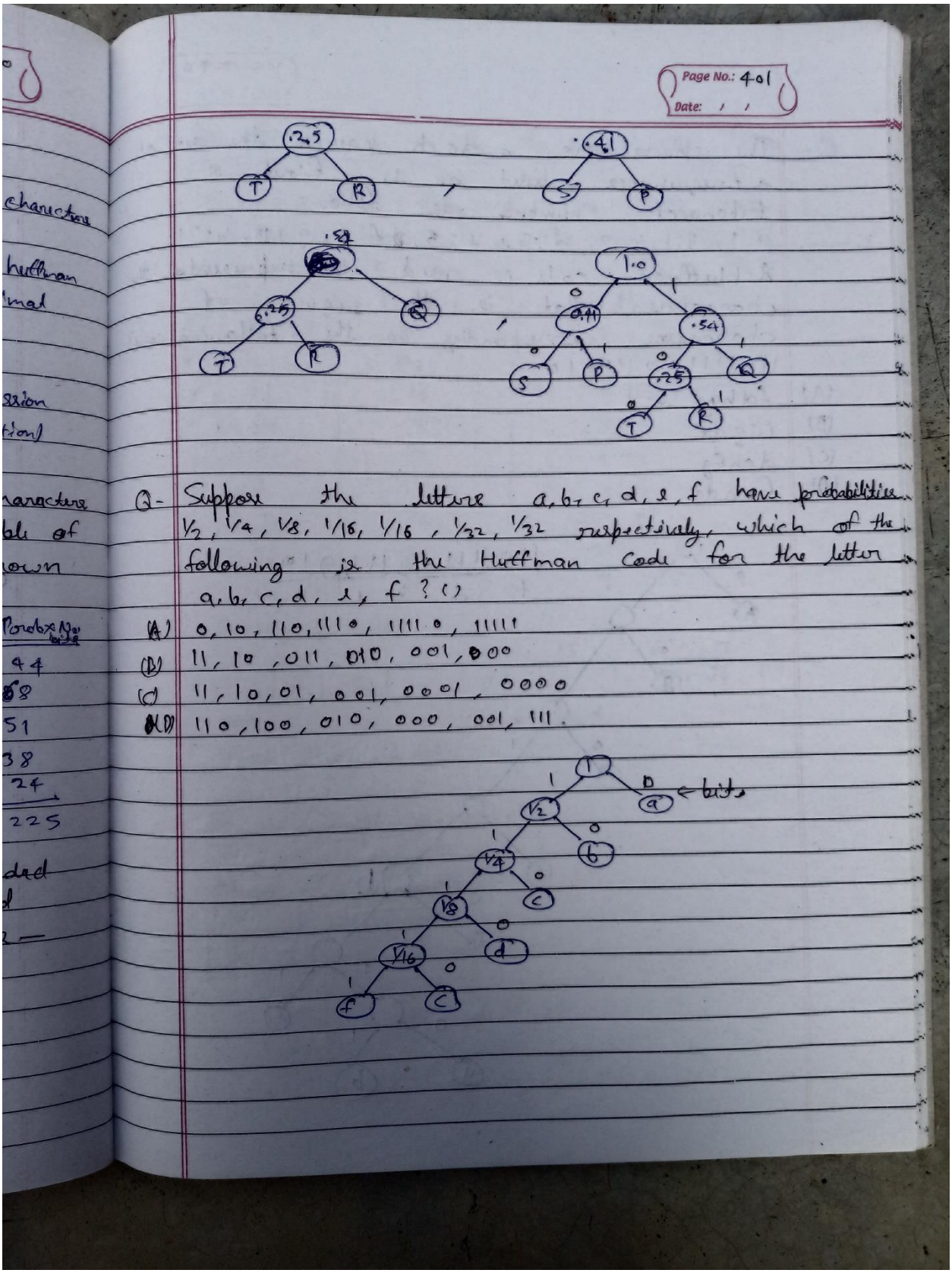
Q- A message is made up entirely of characters from the set $X = \{P, Q, R, S, T\}$. The table of probabilities of each character is shown below:-

Character	Probability	No. bits	Prob × No. bits
P	0.22	2	44
Q	0.34	2	68
R	0.17	3	51
S	0.19	2	38
T	0.08	3	24
Total	1.00		225

A message of 100 character over X is encoded using Huffman coding. Then the expected length of the encoded message in bits is-

(A) 225 (✓)
(B) 226
(C) 227
(D) 228

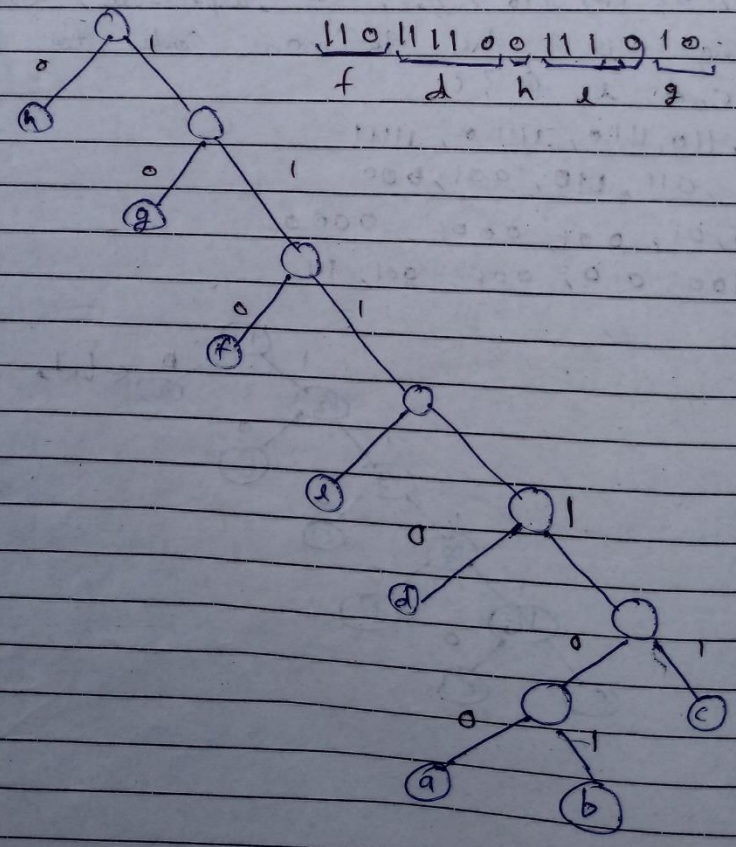
Q-
(A)
(B)
(C)
(D)


Page No.: 401
Date: / /
.25
T
R
.41
S
P
.84
.25
T
R
Q
1.0
0.44
.54
S
P
.25
Q
T
R
Q- Suppose the letters a, b, c, d, e, f have probabilities 1/2, 1/4, 1/8, 1/16, 1/16, 1/32, 1/32 respectively, which of the following is the Huffman code for the letter a, b, c, d, e, f ?
(A) 0, 10, 110, 1110, 11110, 11111
(B) 11, 10, 011, 010, 001, 000
(C) 11, 10, 01, 001, 0001, 0000
(D) 110, 100, 010, 000, 001, 111
1
1/2
1/4
1/8
1/16
1/16
1
0
a
b
c
d
e
f
← bits

Page No.: 402
Date: / /

Q- The characters a to h have the set of frequencies based on the first 8 Fibonacci numbers as follows -
a: 1, b: 1, c: 2, d: 3, e: 5, f: 8, g: 13, h: 21
A Huffman code is used to represent the characters. What is the sequence of characters corresponding to the following code?
11011110011110

(A) fdheg
(B) egdf
(C) dchfg
(D) fehdg



Page No.: 403
Date: 9/12/20

Job Sequencing with deadline -

Every Job take 1 unit of time to complete. objective to maximize the profit

	Jobs	Profit	Deadline
✓	J_1	85	5
X	J_2	25	4
X	J_3	16	3
✓	J_4	40	3
✓	J_5	55	4
X	J_6	19	5
✓	J_7	92	2
✓	J_8	80	3
✓	J_9	15	7

Picking greedily -

1	2	3	4	5	6	7
J_4	J_7	J_8	J_5	J_1		J_9

start

Gantt-Chart

Total Profit - 357

Time Complexity - $O(n^2)$

Space Complexity - $O(n)$

Page No.: 404
Date: / /

Q- We are given 9 task $T_1, T_2, \dots T_9$. The execution of each task requires one unit of time. We can execute one task at a time. Each task T_i has a profit P_i and a deadline d_i. Profit P_i is earned if the Task is completed.

Task	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
Profit	15	20	30	18	18	16	23	16	25
Deadline	7	2	5	1	3	4	5	7	3

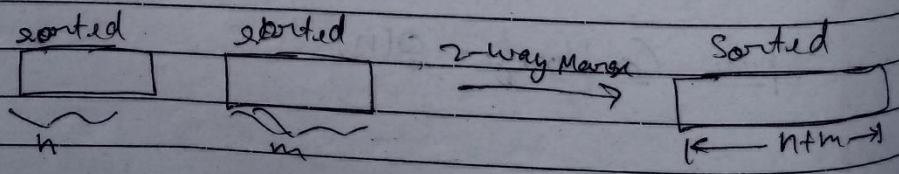
Are all task completed in the schedule that gives maximum profit ?

(A) All task are completed
(B) T_2 and T_6 are left out
(C) T_1 and T_8 are left out
(D) T_4 and T_6 are left out ✓

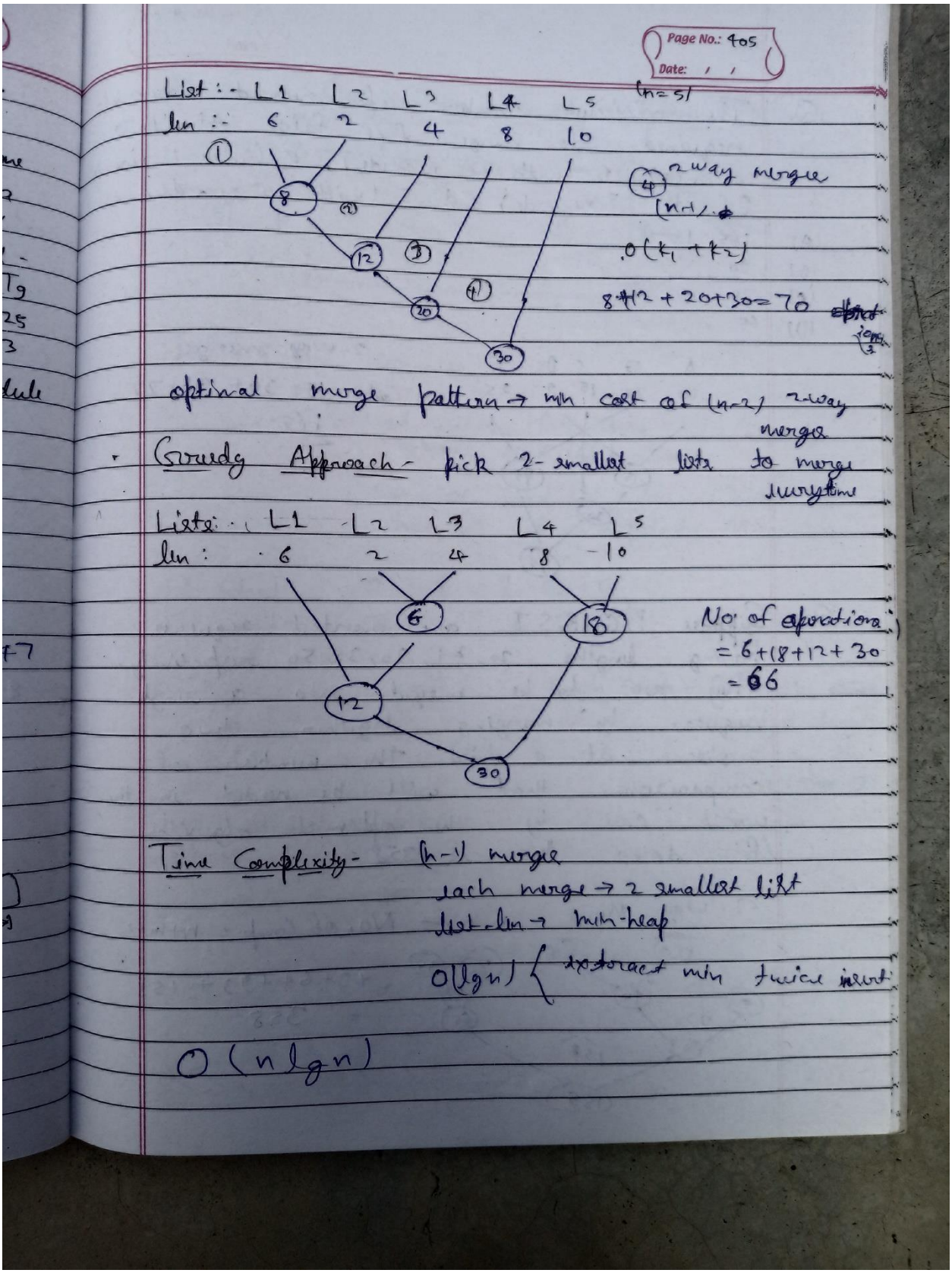
1	2	3	4	5	6	7	
T_2	T_7	T_9	T_5	T_3	T_1	T_8	Max Profit = 147

Optimal merge Pattern- (Greedy Algo^m).

- Merge Sort- $O(m+n)$.



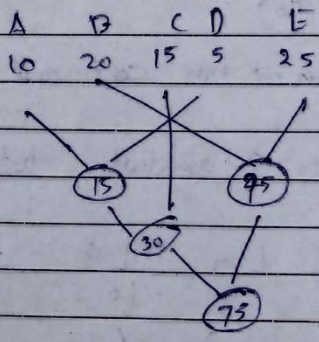
- Multiple - 2-sorted lists Merge-
  - n-way merge
  - multiple 2-way merge


Page No.: 405
Date: / /
List :- L1 L2 L3 L4 L5
len :- 6 2 4 8 10
(n=5)
8
12
20
30
2 way merge
O(k1+k2)
8+12+20+30=70
optimal merge pattern → min cost of (n-2) 2-way merges
Greedy Approach - pick 2 smallest lists to merge everytime
List :- L1 L2 L3 L4 L5
len : 6 2 4 8 10
6
18
12
30
No of operations
= 6+18+12+30
= 66
Time Complexity - (n-1) merge
each merge → 2 smallest list
list-len → min-heap
O(lgn) { extract min twice insert
O(nlgn)

Page No.: 406
Date: / /

Q- The minimum number of record movements required to merge five files A (with 10 records), B (with 20 records), C (with 15 records), D (with 5 records) and E (with 25 records) is-

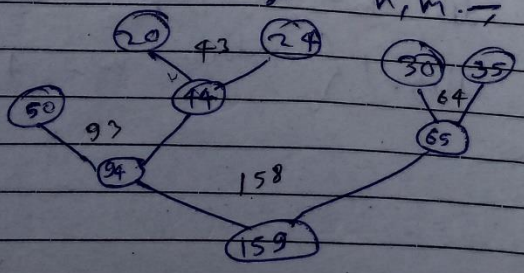
(A) 165 (✓)
(B) 90
(C) 75
(D) 65



2-way merge.
Cost - 15 + 30 + 45 + 75
= 165

Q- Suppose P, Q, R, S, T are sorted sequence having lengths 20, 24, 30, 35, 50 respectively. They are to be merged into a single sequence by merging together two sequence at a time. The number of comparisons that will be needed in the worst case by the optimal algorithm for doing this is 358.

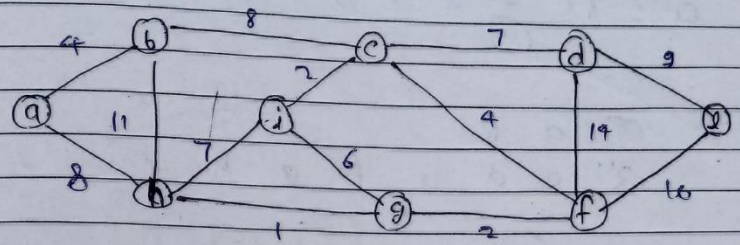
2-way Merge n, m → No. of Comp. = n+m-1



= 43 + 64 + 93 + 158
= 358

Page No.: 407
Date: / /

Minimum Spaning Tree - Greedy Algorithms - (Prim's Algo)

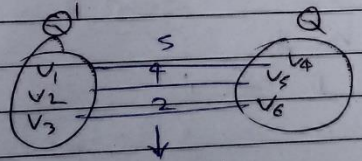


Start - Empty Spanning Tree

Tree Sets
- Set of vertices for which we already computed MST (Q')
- Set of vertices for which we haven't yet computed MST (Q)

At Start - $Q' = \phi$ & $Q = V$
At end - $Q' = V$ & $Q = \phi$

Ex-

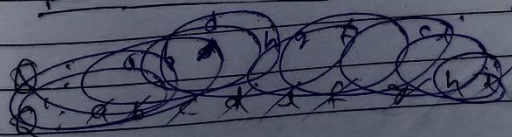


Cut set of vertices Q' & Q

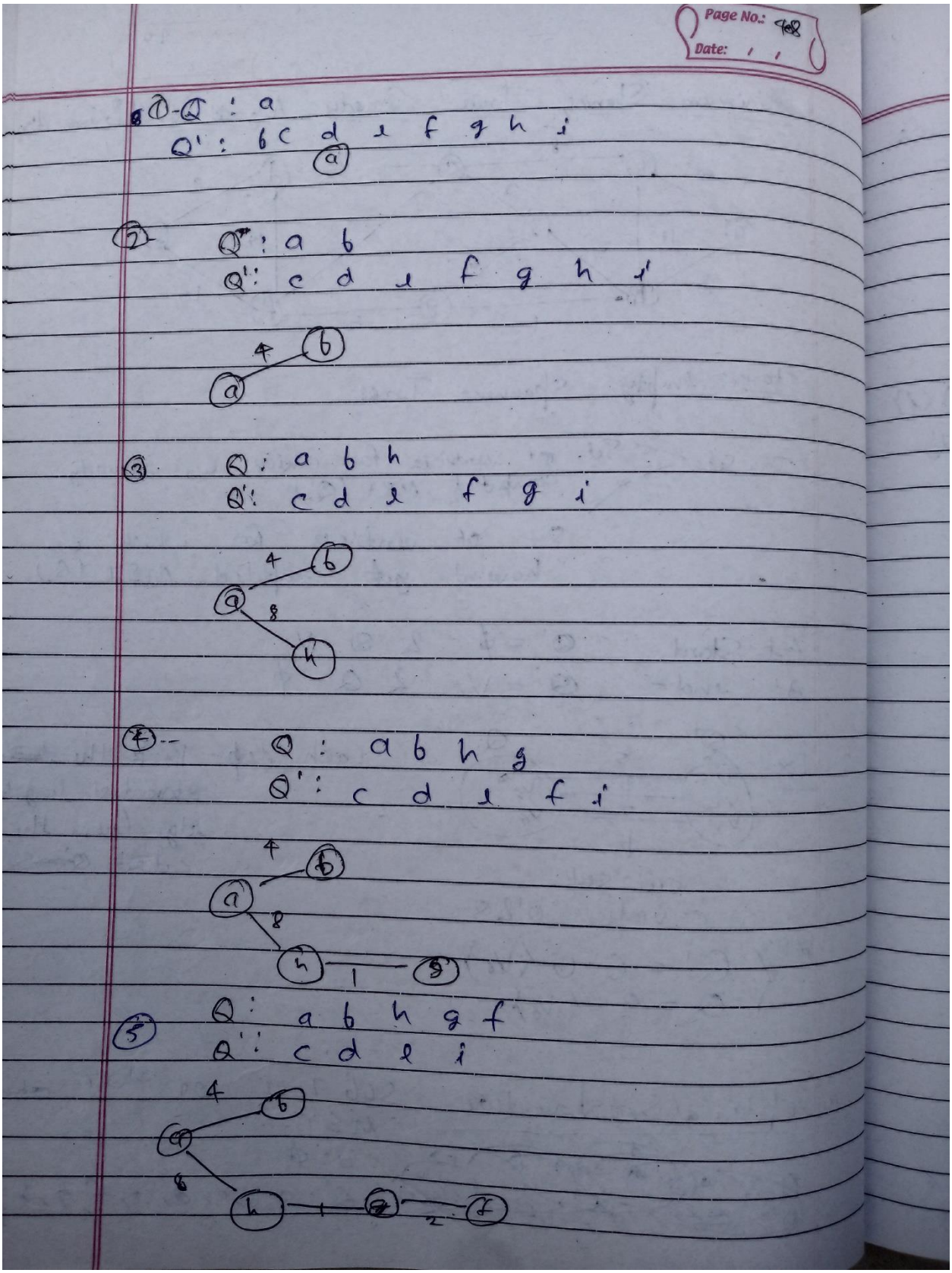
Each Step - Pick the shortest length edge from the cut set Q' — Q

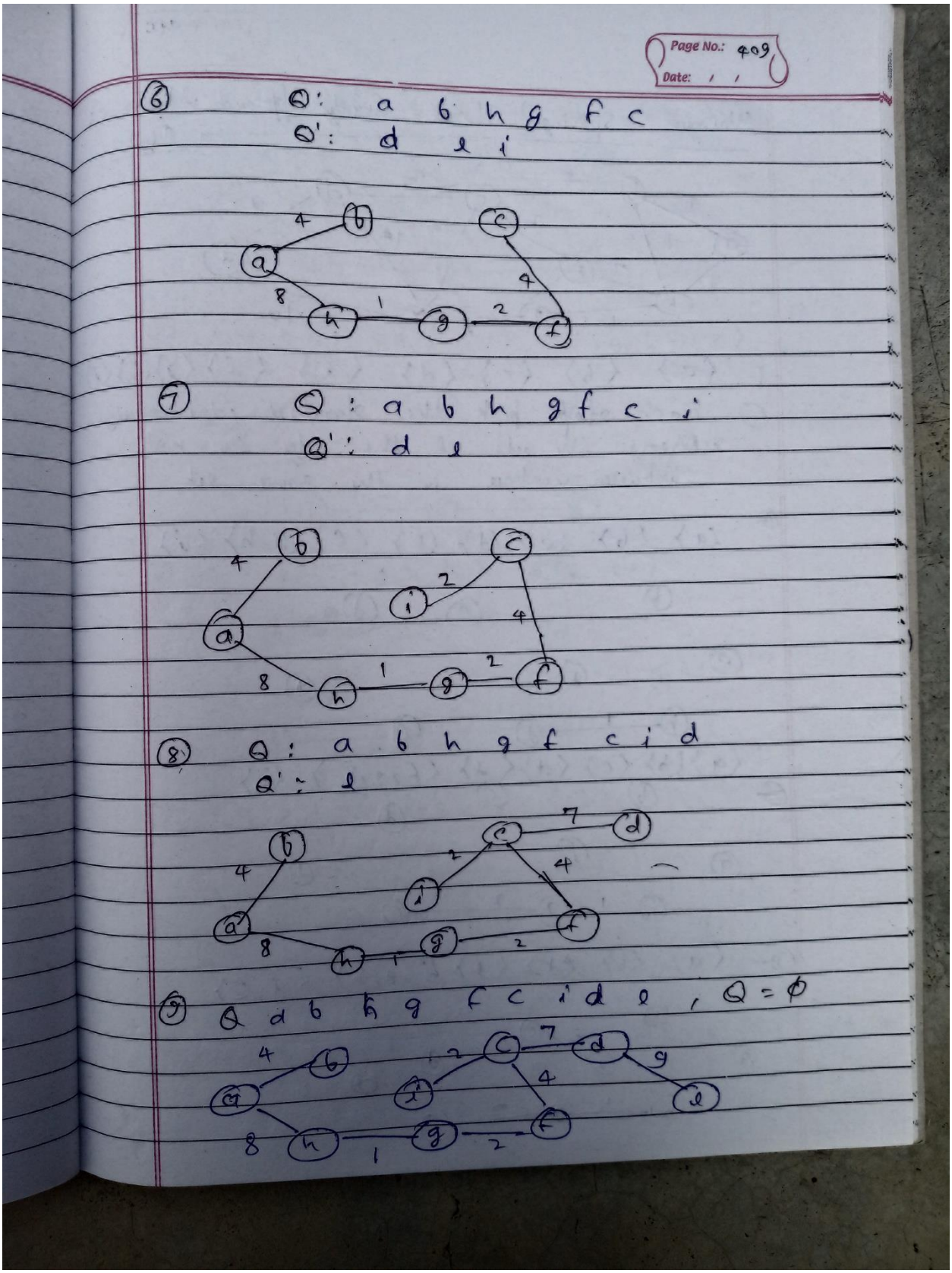
Pick 2 { $Q' = Q' \cup \{V_6\}$; $Q = Q - \{V_6\}$

- **Optimal Substructure** - Sub Tree of MST are MST.



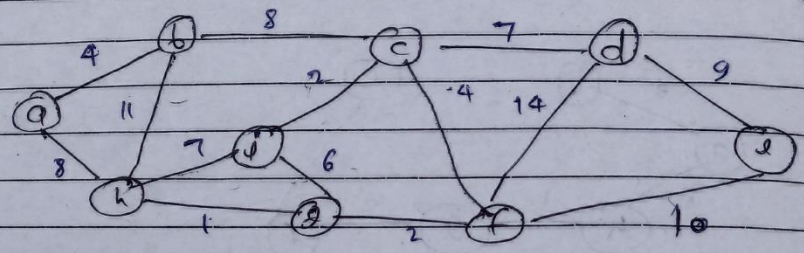
Q: ϕ
Q': a, b, c, d, e, f, g, h, i


Page No.:
Date:
① Q : a
Q' : b c d e f g h i
(a)
② Q : a b
Q' : c d e f g h i
4
③ Q : a b h
Q' : c d e f g i
4
8
④ Q : a b h g
Q' : c d e f i
4
8
1
⑤ Q : a b h g f
Q' : c d e i
4
6
1
2


Page No.: 409
Date: / /
6
Q: a b h g f c
Q': d e i
4
8
1
2
4
7
Q: a b h g f c i
Q': d e
2
4
8
1
2
4
8
Q: a b h g f c i d
Q': e
2
4
7
8
1
2
4
9
9
Q d b h g f c i d e , Q = ϕ

Page No.: 40
Date: / /

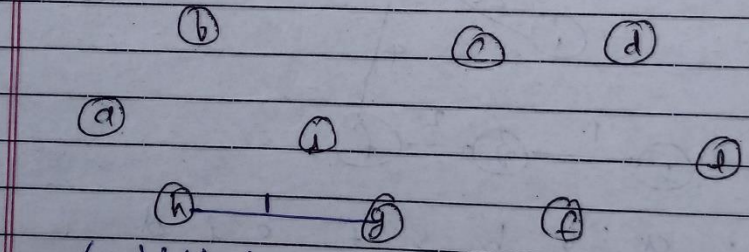
Minimum Spaning Trees - Greedy Approach → Kruskal's Algo -



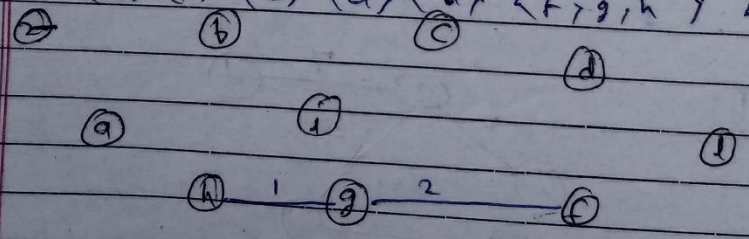
1- {a} {b} {c} {d} {e} {f} {g} {h} {i}

2- Each step- pick the shortest edge → greedy
→ Merge the sets if the edge is not between vertices in the same set

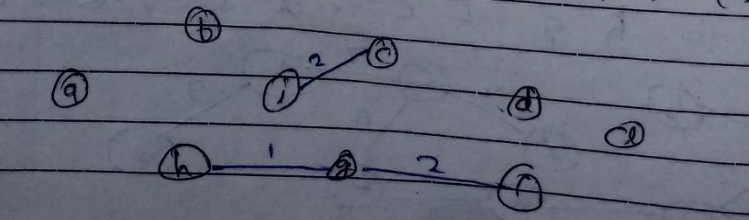
{a} {b} {c} {d} {e} {f} {g, h} {i}



{a} {b} {c} {d} {e} {f, g, h} {i}

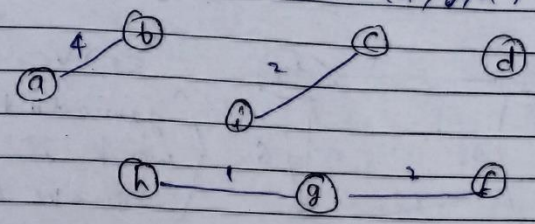


3- {a} {b} {d} {e} {f, g, h} {i, c}

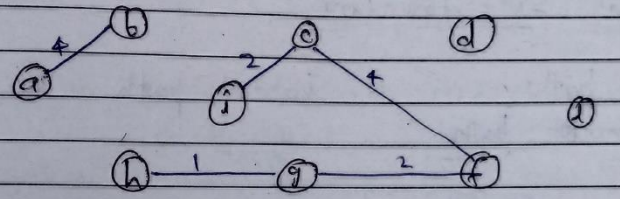


Page No.: 411
Date: / /

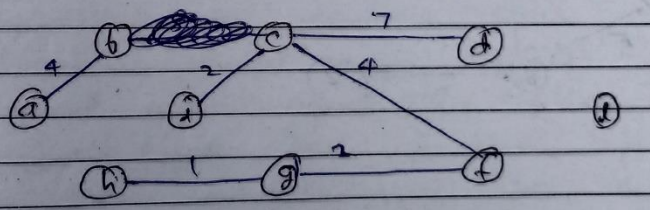
④ {a, b}, {d} {g} {f, g, h} {i, c}



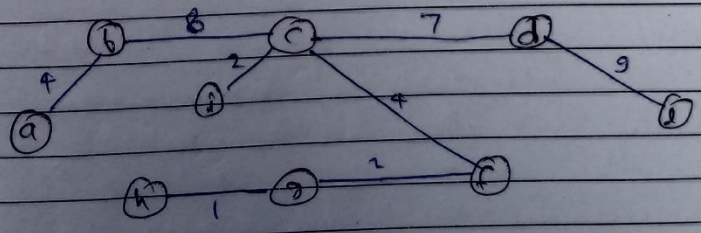
⑤ {a, b}, {d}, {i}, {f, g, h, i, c}



⑥ {a, b} {c, f g, h, di} {i}



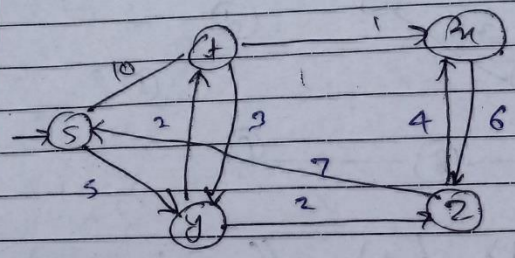
⑦ {a, b, c, d f, g, h, i} {i}.



1. Optimal Substructure - Sub tree of a MST or MST.

Page No.: 412
Date: / /

Single Source Shortest Path - Dijkstra's Algorithm



$s \rightsquigarrow V - \{s\}$

{ guaranteed to work if there is no -ve edges

- **Optimal Sub-Structure**

Sub paths in a shortest path are also shortest path